HITACHI
Inspire the Next

Hitachi Streaming Data Platform
アプリケーション開発ガイド

3000-3-E24-30

前書き

■ 著作権

■ 輸出時の注意

本製品を輸出される場合には、外国為替及び外国貿易法の規制並びに米国輸出管理規則など外国の輸出関連法規をご確認の上、必要な手続きをお取りください。

なお、不明な場合は、弊社担当営業にお問い合わせください。

■ 商標類

Regular expression support is provided by the PCRE library package, which is open source software, written by Philip Hazel, and copyright by the University of Cambridge, England. The original software is available from ftp://ftp.csx.cam.ac.uk/pub/software/programming/pcre/

This product includes software developed by Ralf S. Engelschall <rse@engelschall.com> for use in the mod_ssl project (http://www.modssl.org/).

This product includes software developed by IAIK of Graz University of Technology.

This product includes software developed by the Java Apache Project for use in the Apache JServ servlet engine project (http://java.apache.org/).

This product includes software developed by Daisuke Okajima and Kohsuke Kawaguchi (http://relaxngcc.sf.net/).

This product includes software developed by Andy Clark.

Java is a registered trademark of Oracle and/or its affiliates.

■ 発行

2019 年 11 月

変更内容

変更内容(3000-3-E24-30)　Hitachi Streaming Data Platform 03-30，Hitachi Streaming Data Platform software development kit 03-30

追加・変更内容	変更個所
次のインタフェース，クラス，および列挙型を追加した。 • SDPClientLogger インタフェース • SDPClientLoggerFactory クラス • LogKind 列挙型 これに伴い，次のサンプルプログラムの内容を変更した。 • インプロセス連携送受信制御 AP（Inprocess_Main.java） • インプロセス連携データ送信 AP（Inprocess_Sender.java） • インプロセス連携データ受信 AP（Inprocess_Receiver.java）	8.2，8.3，8.9，8.12，10.1.2，10.1.3，10.1.4
SDPClientException クラスのサブクラスにSDPClientInitializeException を追加した。	8.13.1

単なる誤字・脱字などはお断りなく訂正しました。

はじめに

このマニュアルは，Hitachi Streaming Data Platform （Streaming Data Platform）の設計・構築・運用方法，および構築時に設定できる機能の詳細について説明したものです。Hitachi Streaming Data Platform （Streaming Data Platform）の設計・構築・運用をすることで，ストリームデータの分析ができるようになることを目的としています。

■ 対象製品

- P-9W64-9F31 Hitachi Streaming Data Platform 03-30 （適用 OS：Red Hat(R) Enterprise Linux(R) Server 6（64-bit x86_64），Red Hat(R) Enterprise Linux(R) Server 7（64-bit x86_64））
- P-9W64-9K31 Hitachi Streaming Data Platform software development kit 03-30 （適用 OS：Red Hat(R) Enterprise Linux(R) Server 6（64-bit x86_64），Red Hat(R) Enterprise Linux(R) Server 7（64-bit x86_64））

■ 対象読者

このマニュアルは，ソリューションデベロッパー，およびインテグレーションデベロッパーを対象にしています。

■ このマニュアルで使用する記号

このマニュアルで使用する記号を次に示します。

記号	意味
斜体	可変値，強調，呼び出しを示します。
＜ ＞	可変値（斜体で変数を表現しきれない場合に使用）
[ ]	任意の値であることを示します。
{ }	必要な値，または期待される値を示します。
| （ストローク）	複数の項目または引数から選択することを示します。
… （点線）	この記号の直前に示された項目を繰り返して指定できることを示します。

■ このマニュアルで使用する略語

このマニュアルで使用する製品名の表記

このマニュアルでは，製品名を次のように表記しています。

表記	製品名
HSDP	Hitachi Streaming Data Platform
Streaming Data Platform	
SDP	
HSDP software development kit	Hitachi Streaming Data Platform software development kit
Streaming Data Platform software development kit	
SDP SDK	
JavaVM	Java Virtual Machine
Red Hat Enterprise Linux Server	Red Hat(R) Enterprise Linux(R) Server

このマニュアルで使用する英略語

このマニュアルで使用する英略語を次に示します。

英略語	正式名称
AP	Application Program
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
BOM	Byte Order Mark
CPU	Central Processing Unit
CQL	Continuous Query Language
CSV	Comma Separated Value
EADs	Hitachi Elastic Application Data Store
EUC	Extended UNIX Code
EUC-JP	Extended UNIX Code Packed Format for Japanese
G1GC	Garbage First Garbage Collection
GC	Garbage Collection
GCC	GNU Compiler Collection

英略語	正式名称
GMT	Greenwich Mean Time
HTTP	Hyper Text Transfer Protocol
ID	Identifier
IP	Internet Protocol
JDBC	Java Database Connectivity
JIS	Japanese Industrial Standards
JRE	Java Runtime Environment
JST	Japan Standard Time
JVM	Java Virtual Machine
NIC	Network Interface Card
OS	Operating System
SJIS	Shift JIS
SMTP	Simple Mail Transfer Protocol
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol/Internet Protocol
UDP/IP	User Datagram Protocol/Internet Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF	UCS Transformation Format
VM	Virtual Machine
VTL	Velocity Template Language
XML	Extensible Markup Language

■ ¥の表記について

本文中で使用されている¥は半角のバックスラッシュを意味しています。

■ このマニュアルで使用する KB（キロバイト）などの単位表記

1KB（キロバイト），1MB（メガバイト），1GB（ギガバイト），1TB（テラバイト）はそれぞれ 1,024 バイト，$1{,}024^2$ バイト，$1{,}024^3$ バイト，$1{,}024^4$ バイトです。

■ 関連マニュアル

関連マニュアルを次に示します。必要に応じてお読みください。

- 『Hitachi Streaming Data Platform 入門』 (3000-3-E21)
- 『Hitachi Streaming Data Platform 概説』 (3000-3-E22)
- 『Hitachi Streaming Data Platform システム構築ガイド』(3000-3-E23)
- 『Hitachi Streaming Data Platform メッセージ』 (3000-3-E25)

目次

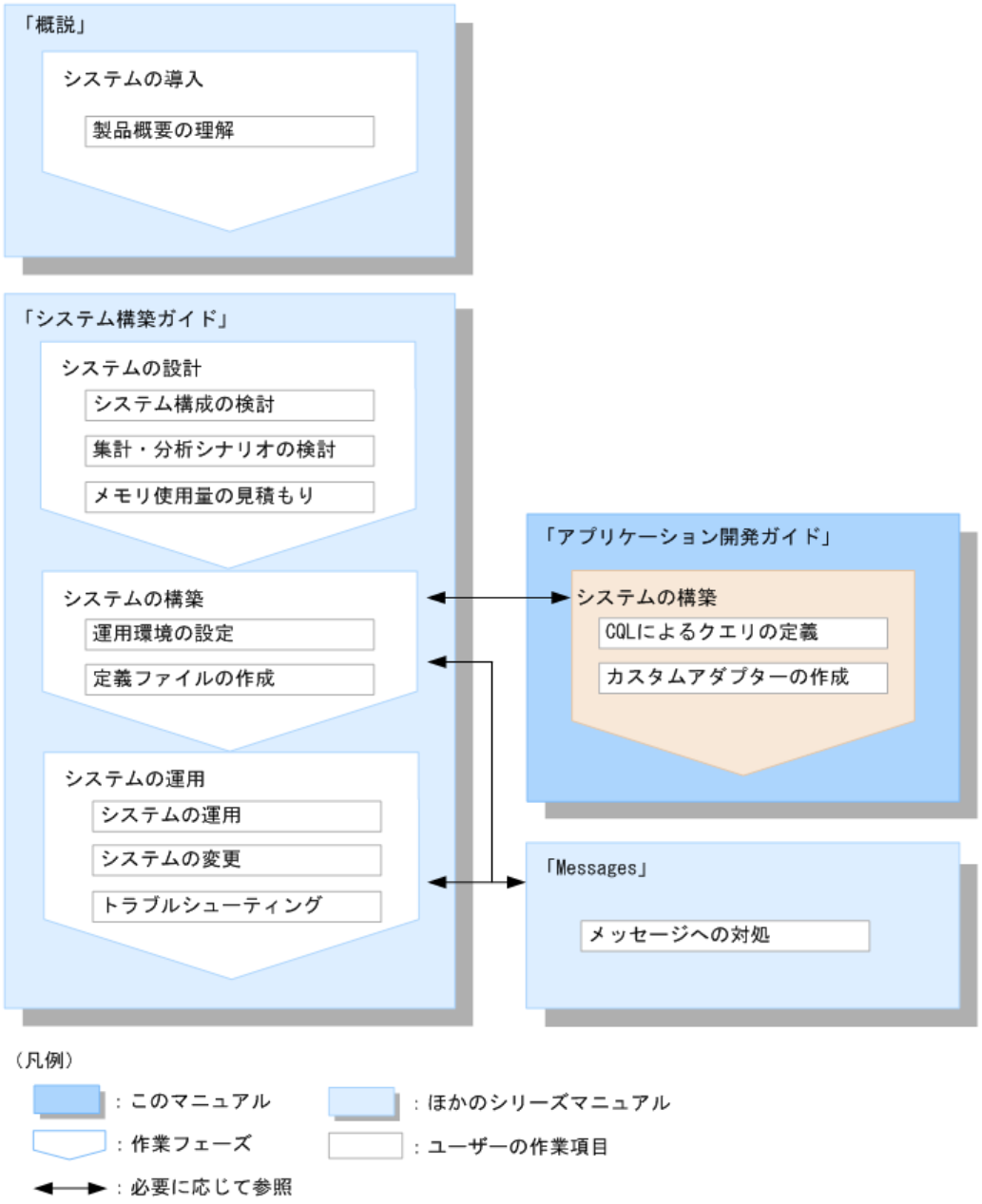
1 Hitachi Streaming Data Platform でのアプリケーション開発の概要

この章では，このマニュアルを読み進める前に知っておいていただきたい内容として，Hitachi Streaming Data Platform の導入から運用までの流れ，このマニュアルの構成，および Hitachi Streaming Data Platform でのアプリケーション開発の概要について説明します。

1.1　導入から運用までの流れ

Hitachi Streaming Data Platform の導入から運用までの流れと関連マニュアルとの関係を，次の図に示します。

図 1-1　導入から運用までの流れと関連マニュアルとの関係



このマニュアルでは，「システムの構築」作業のうち，「CQL によるクエリの定義」および「カスタムアダプターの作成」について説明します。

なお，このマニュアルを読む前に，マニュアル『Hitachi Streaming Data Platform 概説』を参照して，Hitachi Streaming Data Platform の製品概要を理解しておいてください。

また，必要に応じて，各作業フェーズで関連マニュアルを参照してください。

1.2　このマニュアルの構成

このマニュアルの構成を紹介します。

表 1-1　各章と付録の記載内容

章	記載内容
1.　Hitachi Streaming Data Platform でのアプリケーション開発の概要	この章です。このマニュアルを読み進める前に知っておいていただきたい内容について説明しています。
2.　CQL によるクエリの定義	CQL によるクエリの定義方法について説明しています。
3.　CQL の基本項目とデータ型	CQL の基本項目とデータ型について説明しています。
4.　CQL リファレンス	CQL の文法について説明しています。
5.　CQL で指定する組み込み関数	CQL で指定する組み込み関数の文法について説明しています。
6.　CQL デバッグツールの操作	クエリのデバッグ，テスト結果の作成，および CQL のチューニングで使用する開発支援ツールについて説明しています。
7.　カスタムアダプターの作成	カスタムアダプターの作成方法について説明しています。
8.　データ送受信用 API	カスタムアダプターの作成で使用するデータ送受信用 API の文法について説明しています。
9.　外部アダプター用 API	外部アダプターの作成で使用する外部アダプター用 API の文法について説明しています。
10.　データ送受信用 API を使用したサンプルプログラム	データ送受信 API を使用したサンプルプログラムについて説明しています。
11.　HSDP クライアントライブラリの使用方法	HSDP クライアントライブラリの使用方法について説明しています。
12.　外部定義関数の作成（Java）	Java の外部定義関数の作成方法について説明しています。
13.　外部定義関数の作成（C）	C の外部定義関数の作成方法について説明しています。
14.　外部定義関数インタフェース（Java）	外部定義関数の作成で使用する外部定義関数インタフェースの文法について説明しています。
15.　外部定義関数インタフェース（C）	C の外部定義関数について説明しています。
16.　コマンド	コマンドについて説明しています。

1.3 アプリケーション開発の概要

Hitachi Streaming Data Platform でのアプリケーション開発の概要について説明します。

アプリケーション開発では，次の作業を実施します。

- 分析シナリオの定義（クエリの定義）
- ストリームデータ処理エンジンとのデータ送受信用のアプリケーションの作成（カスタムアダプターの作成）

分析シナリオは，分析するデータや分析内容に合わせて定義する必要があります。

カスタムアダプターは，標準提供アダプターを使用するシステムでは不要なアプリケーションです。必要に応じて作成してください。

1.3.1 クエリの定義でできること

Hitachi Streaming Data Platform では，ストリームデータ処理エンジンに入力されたストリームデータに対して，あらかじめ登録されたシナリオに沿った分析を実行し，目的に応じた分析結果を出力します。

シナリオは，クエリ定義ファイルに，クエリとして定義します。クエリに定義する内容を次に示します。

- **分析対象の特定方法**

  ストリームデータは，途切れることなく続く時系列順のデータです。分析するためには，分析対象とする範囲を特定する必要があります。

  Hitachi Streaming Data Platform では，ウィンドウという概念を使用して，ストリームデータを時間またはタプルの個数で区切られた有限のデータとして扱い，分析対象の範囲を特定します。

- **分析処理の内容**

  ウィンドウで区切られたストリームデータに対して，目的に沿った分析処理を実施します。特定の列について値の推移を分析したり，複数のストリームデータの値を組み合わせて新たなストリームデータを生成して分析したりします。

  分析処理の内容は，選択，結合などの関係演算や，集合関数を使用した集合演算などによって定義します。

- **分析結果の出力方法**

  分析した結果を新たなストリームデータとして出力します。

  分析結果のストリームデータは，分析結果が増減したときに出力したり，一定間隔ですべての分析結果を出力したりできます。

  分析結果の出力には，ストリーム化演算という演算を使用します。

メモ

ストリーム間演算を使用すると，リレーションを生成しないで，直接ストリームデータに対して演算を実行し，演算の結果を別のストリームデータに変換できます。

ストリーム間演算では，入出力がストリームデータであること以外は特に規定がなく，入力されたストリームデータに対して実行する処理は任意です。ストリーム間演算関数の処理ロジックを，ユーザーが Java で記述して作成するクラスファイルにメソッドとして実装することで，任意の処理を実行できます。

ストリーム間演算を使用する場合は，CQL でストリーム間演算関数を定義するほかに，外部定義関数の作成が必要です。外部定義関数の作成については，「12.　外部定義関数の作成（Java）」を参照してください。

クエリは，CQL という SQL に類似したクエリ言語を使用して定義します。

CQL を使用したクエリの定義方法については，「2.　CQL によるクエリの定義」で説明します。また，CQL の記述方法の詳細は，「3.　CQL の基本項目とデータ型」，「4.　CQL リファレンス」，および「5.　CQL で指定する組み込み関数」で説明します。なお，クエリ定義ファイルについては，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

また，クエリ定義ファイルは，サンプルを基に定義できます。サンプルの内容は，「6.　CQL デバッグツールの操作」で説明します。

1.3.2　カスタムアダプターの作成でできること

Hitachi Streaming Data Platform では，ストリームデータの入出力に使用する機能として，標準提供アダプターを提供しています。標準提供アダプターを使用する場合，API を使用したアプリケーションの開発は不要です。

ただし，標準提供アダプターには次のような制限があります。

標準提供アダプターの制限事項

- 扱えるストリームデータの形式は，TCP データ，ファイルおよび HTTP パケットだけです。
- 定義できるシナリオ（クエリグループ）の個数に制限があります。
- ストリームデータ処理エンジンに対するデータ送信とデータ受信に使用するアダプター（アプリケーション）のプロセス構成を任意に決定できません（データ送信 AP をインプロセスで開始する場合は，データ受信 AP もインプロセスで開始する必要があるなど）。

標準提供アダプターについては，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

任意の形式のストリームデータを分析したい場合など，標準提供アダプターを使用できないときに，カスタムアダプターを作成してください。

カスタムアダプターは，Streaming Data Platform が提供するデータ送受信 API を使用した Java アプリケーションとして作成します。データ送受信 API の使用方法，アプリケーション開発時の留意事項，コンパイル方法などについては，「7. カスタムアダプターの作成」で説明します。データ送受信 API の詳細は，「8. データ送受信用 API」で説明します。

2 CQL によるクエリの定義

この章では，CQL によるクエリの定義方法について説明します。

なお，この章で示す CQL の記述方法の詳細は，「4. CQL リファレンス」を参照してください。

2.1　CQL の体系

Hitachi Streaming Data Platform では，ストリームデータ処理エンジンに入力されたストリームデータに対して，あらかじめ登録されたシナリオに沿った分析を実行し，目的に応じた分析結果を出力します。

シナリオは，CQL というクエリ言語を使用して定義します。

Hitachi Streaming Data Platform で使用する CQL の体系を次の表に示します。

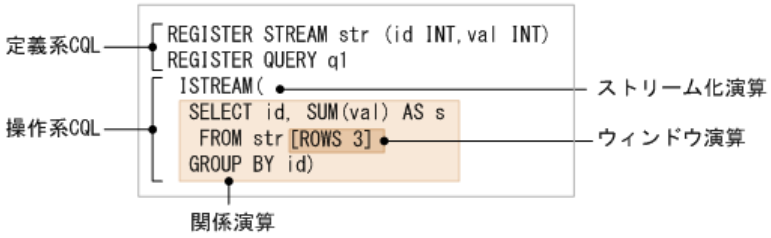
表 2-1　Hitachi Streaming Data Platform で使用する CQL の体系

項番	CQL の分類	用途	該当する CQL の例
1	定義系 CQL	ストリームやクエリを定義して，ストリームデータ処理エンジンに登録します。	REGISTER STREAM REGISTER QUERY など
2	操作系 CQL	クエリで実行するストリームデータの処理内容について定義します。REGISTER QUERY 句に続けて記述します。 操作系 CQL では，次の 4 種類の演算実行をクエリに定義できます。 • ウィンドウ演算 • 関係演算 • ストリーム化演算 • ストリーム間演算	SELECT FROM WHERE GROUP BY HAVING UNION など

CQL の詳細については，「4.　CQL リファレンス」で説明します。

CQL によるクエリの指定例を次の図に示します。

図 2-1　CQL によるクエリの指定例



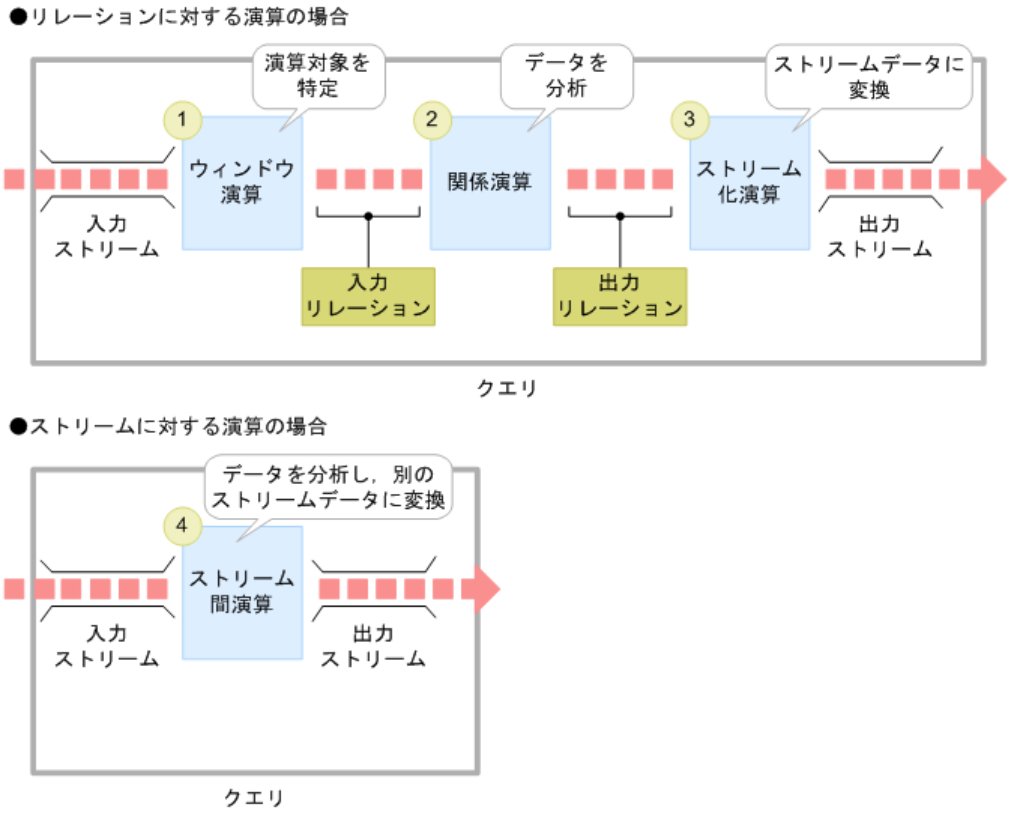
定義系 CQL では，ストリームデータ処理エンジンに，次の情報を登録します。

- 処理の対象になるストリームの情報
- 操作系 CQL によって処理内容を記載したクエリ

操作系 CQL では，入力ストリームキューから入力されたデータに対してシナリオに沿った分析処理を実行するための，演算処理を定義します。

クエリで定義するウィンドウ演算，関係演算，ストリーム化演算，およびストリーム間演算の関係を次の図に示します。

図 2-2　演算の関係



各演算での実施内容について説明します。

1. ウィンドウ演算によって，演算対象を特定します。時系列集合である入力ストリームデータまたは出力ストリームデータは，そのままの状態では演算の対象にできません。ウィンドウ演算によって，生存期間を持つ，*n* 項組のタプルの集合（**リレーション**）として切り取ることで，演算の対象にできます。演算の対象として特定したリレーションを**入力リレーション**といいます。

   ウィンドウ演算では，タプルの個数や時間などによって演算対象を特定するために，ROWS ウィンドウ，RANGE ウィンドウ，PARTITION BY ウィンドウなどを使用します。

2. 関係演算によって，分析の目的に沿った処理を実行します。

   関係演算では，選択，結合，集合関数などの演算を実行して，結果データをリレーションとして抽出します。このリレーションを**出力リレーション**といいます。

3. ストリーム化演算によって，関係演算の結果データをストリームデータに変換します。出力リレーションの変化内容に応じて，追加分，削除分，集合などのデータをストリームデータとして出力できます。

4. ストリーム間演算によって，ストリームデータに対して演算を実行し，別のストリームデータに変換します。

   ストリーム間演算では，リレーションを生成しません。入力ストリームデータに対して演算を実行し，演算の結果を出力ストリームデータとして出力します。

   また，入力されたストリームデータに対して実行する処理は任意です。ストリーム間演算関数の処理ロジックを，ユーザーが Java で記述して作成するクラスファイルにメソッドとして実装することで，任意の処理を実行できます。

ストリーム間演算を使用する場合は，CQL でストリーム間演算関数を定義するほかに，外部定義関数の作成が必要です。外部定義関数の作成については，「12.　外部定義関数の作成（Java）」を参照してください。

2.2　ウィンドウ演算による入力リレーションの生成

この節では，ウィンドウ演算による入力リレーションの生成について説明します。

ウィンドウ演算は，ストリームデータの一部を切り取り，データ処理の対象を特定する演算です。タプルの数，時間など，異なる軸のウィンドウを組み合わせてデータ処理の対象を特定することもできます。

2.2.1　ウィンドウ演算の種類

ウィンドウ演算では，4 種類のウィンドウによって，処理対象を特定します。

それぞれのウィンドウによって生成される入力リレーションについて説明します。

(1)　データの数による指定（ROWS ウィンドウ）

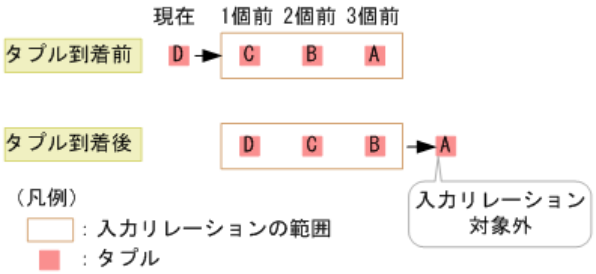
ROWS ウィンドウは，タプルの数によって入力リレーションの範囲を特定するウィンドウです。

例えば，ストリームs1 の入力ストリームデータの直前 3 個分を入力リレーションとして保持する場合，CQL は次のように定義します。

```
SELECT … FROM s1 [ROWS 3]…
```

この場合の入力リレーションの内容を次の図に示します。

図 2-3　ROWS ウィンドウの例



すでにリレーションに A，B，C の 3 つのタプルがある場合に新たなタプル D が到着した場合，到着タプルを含む 3 つのタプルが入力リレーションの内容になります。このとき，いちばん古いタプル A は，入力リレーションの対象から外れます。

(2)　時間による指定（RANGE ウィンドウ）

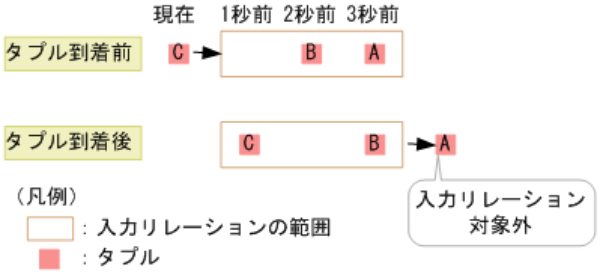
RANGE ウィンドウは，時間によって入力リレーションの範囲を特定するウィンドウです。

例えば，ストリームs2 の入力ストリームデータの直前 3 秒分を入力リレーションとして保持する場合，CQL は次のように定義します。

```
SELECT … FROM s2 [RANGE 3 SECOND]…
```

この場合の入力リレーションの内容を次の図に示します。

図 2-4　RANGE ウィンドウの例



タプル C の到着時，現在から直前 3 秒間分のタプルだけが入力リレーションに残ります。タプル C 到着後，タプル B が 3 秒前のタプルとなり，それよりも前に到着していたタプル A は入力リレーションの対象から外れます。

(3)　到着したタプルのタイムスタンプ時刻による指定（NOW ウィンドウ）

NOW ウィンドウは，その時点で到着したタプルのタイムスタンプ時刻だけを演算の対象にするためのウィンドウです。

ほかのウィンドウで特定する入力リレーションが個数や時間の範囲を持つ「線」のリレーションであるのに対して，NOW ウィンドウは到着したタプルと同時刻という「点」のリレーションになります。

例えば，ストリームs3 の入力ストリームデータについて，到着したタプルと同時刻のタプルを演算の対象とする場合，CQL は次のように定義します。

```
SELECT … FROM s3 [NOW]
```

(4)　データのグループによる指定（PARTITION BY ウィンドウ）

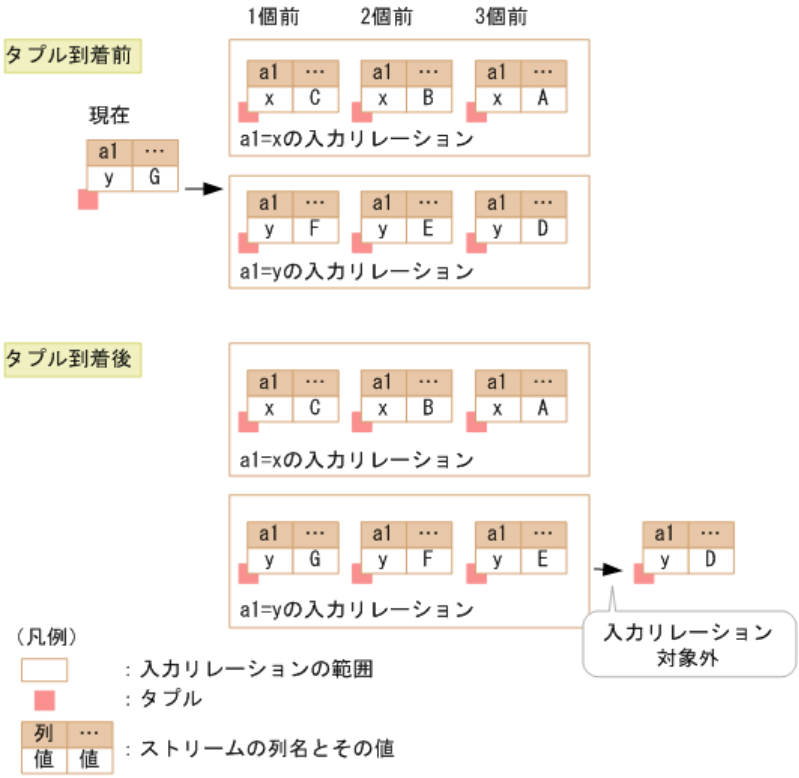
PARTITION BY ウィンドウは，タプルの種類ごとに，タプルの個数によって入力リレーションの範囲を特定するウィンドウです。

例えば，ストリームs4 の入力ストリームデータについて，列指定リストa1 の各値について直前 3 個分を入力リレーションとして保持する場合，CQL は次のように定義します。

```
SELECT … FROM s4 [PARTITION BY a1 ROWS 3]
```

この場合の入力リレーションの内容を次の図に示します。

図 2-5　PARTITION BY ウィンドウの例



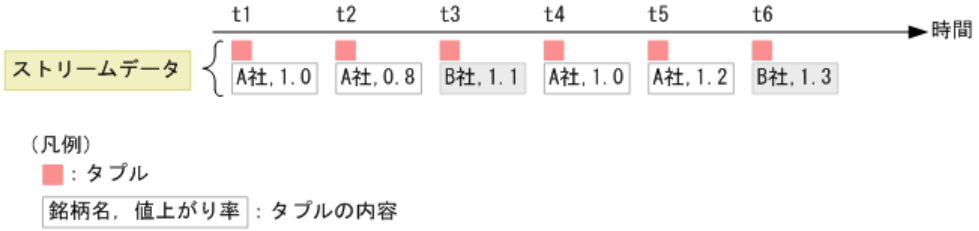
a1 列の値として，x と y があり，それぞれ直前 3 個分を入力リレーションとして保持しています。a1 が y のタプルが到着すると，a1 の値が y のタプル（タプル D）が 1 つ，入力リレーションの対象から外れます。

2.2.2　ウィンドウ演算の例

それぞれのウィンドウ演算の例を示します。

ここでは，次の図に示す構成のストリームデータが到着した場合を例に，それぞれのウィンドウの指定によって作成される入力リレーションについて説明します。

図 2-6　ウィンドウ演算の例で使用するタプルの構成



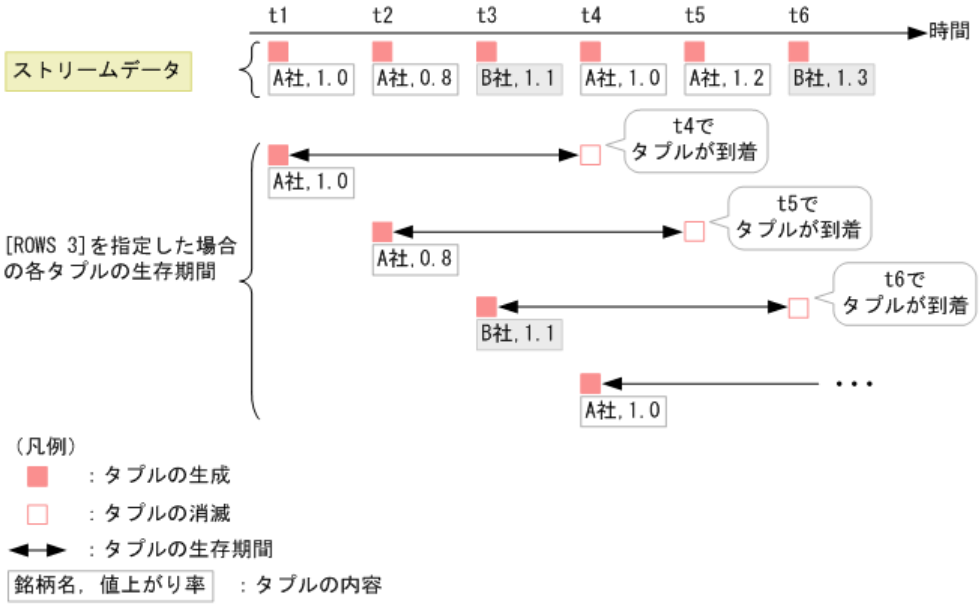
図の横軸は時間軸です。右に行くほど時間が経過しています。t1～t6 は，タプルが到着した時刻を示します。また，各タプルは，「銘柄名，値上がり率」で構成されています。

(1) データの数による指定の例（[ROWS 3]の例）

ここでは，[ROWS 3]を指定した場合のウィンドウ演算の例について説明します。この指定は，入力リレーション内に同時に生存するタプル数が3 であることを示します。

[ROWS 3]を指定した場合の入力リレーション内の各タプルの生存期間について，次の図に示します。

図 2-7 [ROWS 3]を指定した場合の入力リレーション内の各タプルの生存期間



時刻が t1，t2，t3 と経過していく中で，「(A社,1.0)」「(A社,0.8)」「(B社,1.1)」のタプルが順次到着し，入力リレーションに生成されていきます。

時刻 t4 に「(A社,1.0)」のタプルが到着して，同時生存タプル数が3 を超えた時点で，入力リレーション内でいちばん古いタプルの生存期間が終了します。この例では，時刻 t1 に生成された「(A社,1.0)」のタプルが入力リレーションから削除されて，時刻 t4 に到着した「(A社,1.0)」のタプルが新たに入力リレーションに生成されます。

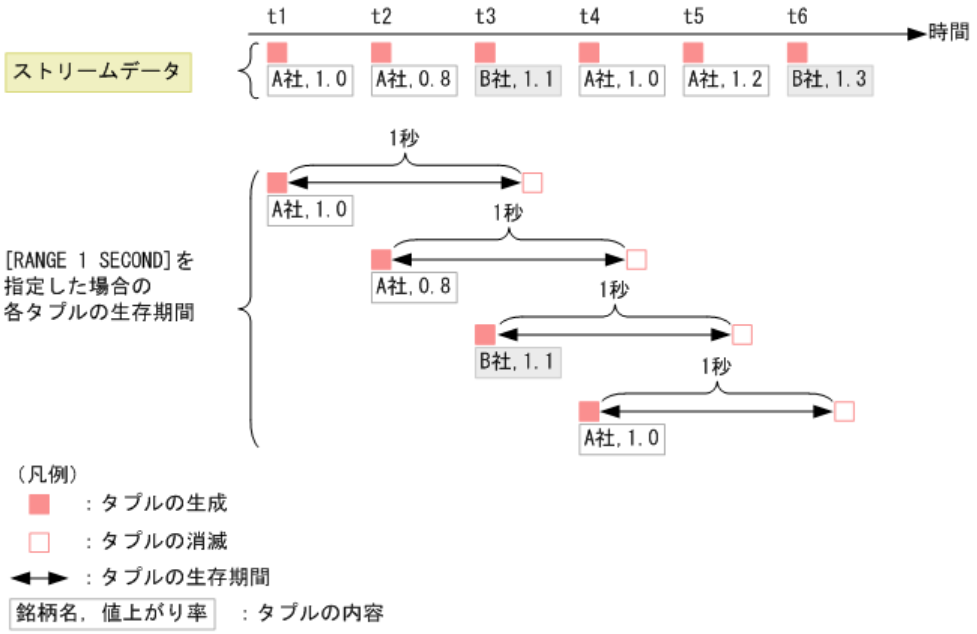
(2) 時間による指定の例（[RANGE 1 SECOND]の例）

ここでは，[RANGE 1 SECOND]を指定した場合のウィンドウ演算の例について説明します。この指定は，入力リレーション内の各タプルの生存期間を1 秒とすることを示します。

リレーション内に生成された各タプルは，その後到着するタプルの個数には関係なく，1 秒たつと消滅します。

[RANGE 1 SECOND]を指定した場合の入力リレーション内の各タプルの生存期間について，次の図に示します。

図 2-8　[RANGE 1 SECOND]を指定した場合の入力リレーション内の各タプルの生存期間



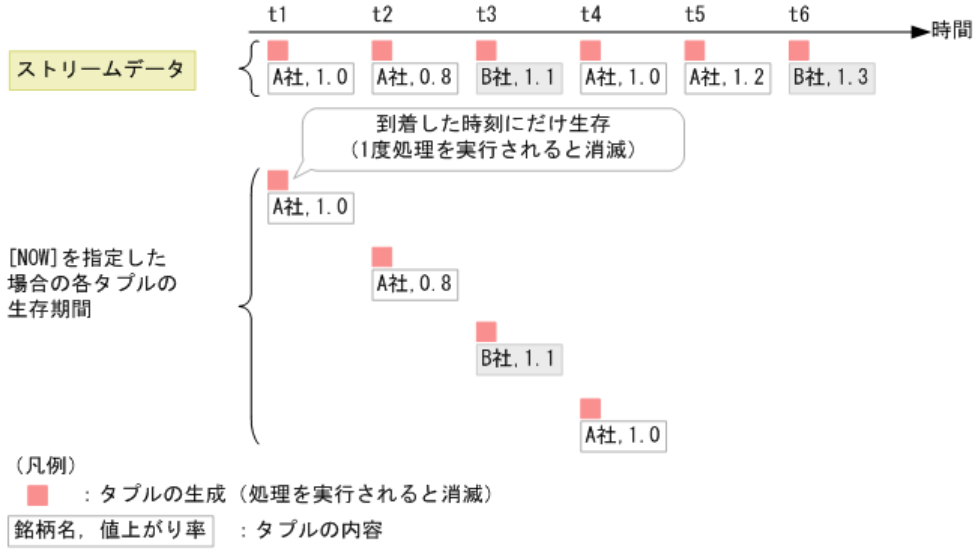
t1～t4 の時刻に到着したタプルは，ほかのタプルの到着状況に関係なく，1 秒たつと消滅します。

(3)　到着したタプルのタイムスタンプ時刻の指定による例（NOW の例）

ここでは，[NOW]を指定した場合のウィンドウ演算の例について説明します。この指定は，入力リレーション内に現時刻のタプルだけを保持することを示します。

[NOW]を指定した場合の入力リレーション内の各タプルの生存期間について，次の図に示します。

図 2-9　[NOW]を指定した場合の入力リレーション内の各タプルの生存期間



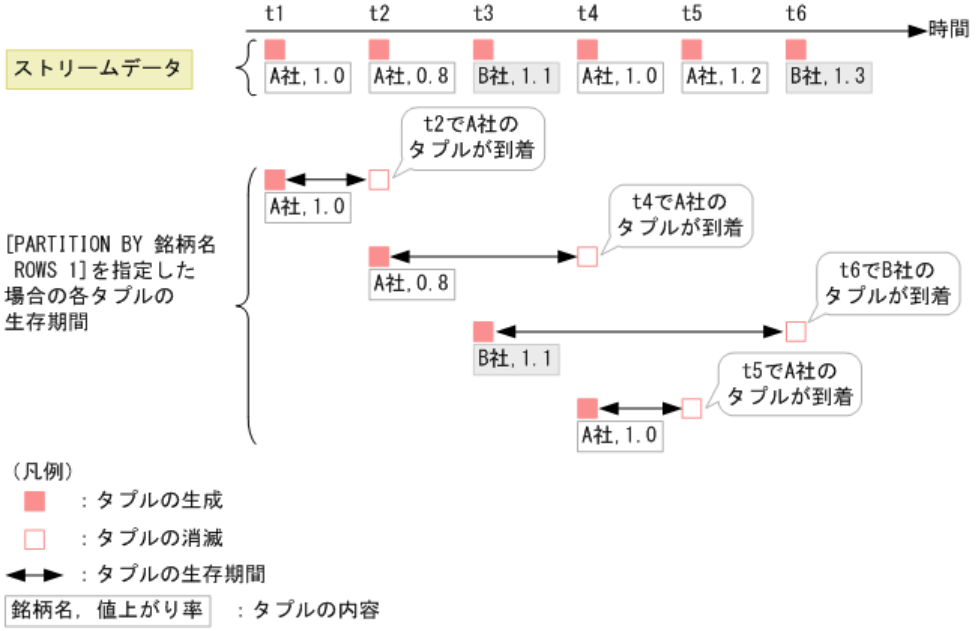
[NOW]を指定した場合，入力リレーション内のタプルは到着した時刻にだけ生存しています。一度演算処理が実行されたタプルは入力リレーションから消滅します。

(4) データのグループによる指定の例（[PARTITION BY 銘柄名 ROWS 1]の例）

ここでは，[PARTITION BY *銘柄名* ROWS 1]を指定した場合のウィンドウ演算の例について説明します。この指定は，入力リレーション内に銘柄ごとに，ROWS に指定した個数ずつのタプルを保持することを示します。

[PARTITION BY *銘柄名* ROWS 1]を指定した場合の入力リレーション内の各タプルの生存期間について，次の図に示します。

図 2-10 [PARTITION BY 銘柄名 ROWS 1]を指定した場合の入力リレーション内の各タプルの生存期間



この指定は，銘柄名 A 社および B 社ごとに，タプルを1 個ずつ保存する指定です。

時刻 t1 に到着した「(A社,1.0)」のタプルが入力リレーションに生存している状態で時刻 t2 に「(A社,0.8)」のタプルが到着すると，t1 に到着した「(A社,1.0)」のタプルが消滅して「(A社,0.8)」のタプルが入力リレーション内に生成されます。続いて時刻 t3 に「(B社,1.1)」のタプルが到着した場合は，銘柄名が異なるため，「(A社,0.8)」のタプルは消滅しないまま，新たに「(B社,1.1)」のタプルが生成されます。時刻 t2 に到着した「(A社,0.8)」のタプルは時刻 t4 に「(A社,1.0)」のタプルが到着するまで，時刻 t3 に到着した「(B社,1.1)」のタプルは時刻 t6 に「(B社,1.3)」のタプルが到着するまで，それぞれ入力リレーション内で保持されます。

2.3　関係演算によるデータの抽出

この節では，関係演算によるデータの抽出について説明します。

関係演算では，選択，結合，および集合関数によって入力リレーション内のストリームデータを操作して，結果のデータを出力リレーションとして抽出します。

2.3.1　関係演算の種類

関係演算では，次の 3 種類の操作によって結果を抽出します。

- 選択

  n 個のタプルを含む入力リレーションから，特定の条件を満たすタプルを抽出します。

  例を示します。

  ```
  REGISTER QUERY q1 SELECT s1.a FROM s1 [ROWS 10] WHERE s1.a > 10;
  ```

  この例では，s1 の入力リレーションから，WHERE 句以下で指定した条件に該当するタプルを抽出します。

- 結合

  n 個のタプルを含む複数の入力リレーションから，同じデータの組み合わせを結合した結果を抽出します。

  例を示します。

  ```
  REGISTER QUERY q2 SELECT s1.a, s1.b, s2.b FROM s1 [ROWS 10], s2 [ROWS 10] WHERE s1.a = s2
  .a;
  ```

  この例では，s1 およびs2 の入力リレーションから，WHERE 句以下で指定した条件に該当するタプルを抽出します。

- 集合関数

  n 個のタプルを含む入力リレーションに対して集合関数による演算を実行した結果を抽出します。

  例を示します。

  ```
  REGISTER QUERY q3 SELECT SUM(s1.a) AS c1 FROM s1 [ROWS 10];
  ```

  この例では，s1 の入力リレーションに対して，集合関数SUM による演算を実行した結果を抽出します。

関係演算での処理内容の詳細については，「4.　CQL リファレンス」を参照してください。

以降では，関係演算で実行する処理のうち，結合処理の例，および集合関数による演算処理の例を示します。

2.3.2　結合処理の例

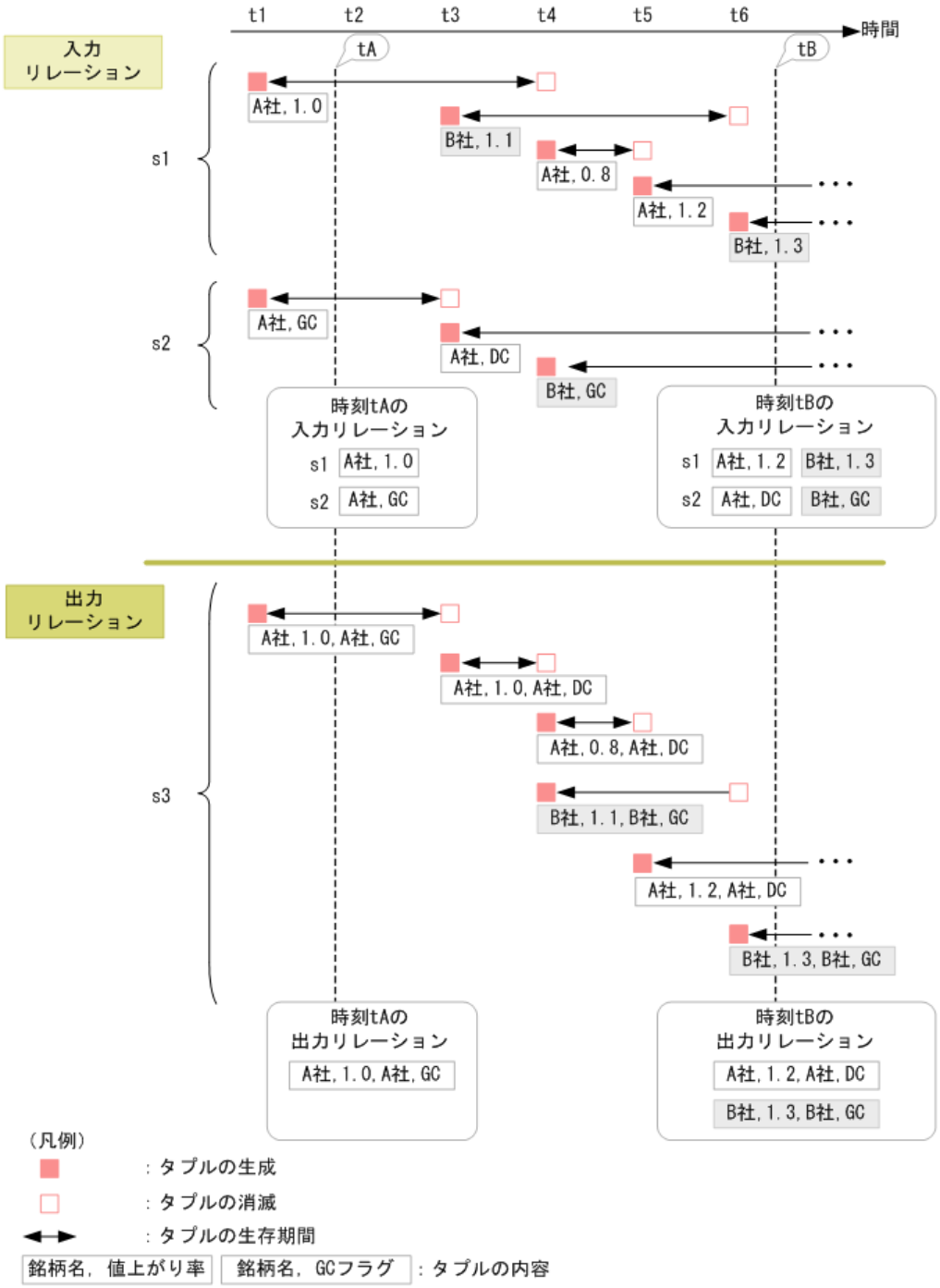
ここでは，s1，s2 の 2 種類の入力リレーションを銘柄ごとに結合した結果を出力リレーションとして抽出する例を示します。

CQL の定義例を次に示します。なお，実際の CQL の定義では，列名は半角英数字で指定してください。

```
REGISTER QUERY q
   SELECT s1.銘柄名, s1.値上がり率, s2.銘柄名, s2.GCフラグ
   FROM s1[PARTITION BY s1.銘柄名 ROWS 1],
        s2[PARTITION BY s2.銘柄名 ROWS 1]
   WHERE s1.銘柄名=s2.銘柄名;
```

この CQL を定義した場合の入力リレーションと出力リレーションの例を次の図に示します。

図 2-11　結合処理の例



図の横軸は時間軸です。右に行くほど時間が経過しています。t1～t6 は，タプルが到着した時刻を示します。また，入力リレーションのs1 のタプルは，「銘柄名，値上がり率」で構成されており，s2 のタプルは，「銘柄名，GCフラグ」で構成されています。なお，GC フラグの値は，「DC（Dead Cross）」，「GC（Golden Cross）」のどちらかです。

この例では，銘柄名によって結合処理を実行して出力します。

時刻 *tA* では，入力リレーションのs1 にはタプル「(A社,1.0)」，s2 にはタプル「(A社,GC)」があります。これらを銘柄名で結合した結果として，出力リレーションには，タプル「(A社,1.0,A社,GC)」が出力されます。

同様に，時刻 *tB* では，入力リレーションのs1 のタプルとして「(A社,1.2)」「(B社,1.3)」があり，s2 のタプルとして「(A社,DC)」「(B社,GC)」があります。これらを銘柄名で結合した結果として，出力リレーションには，タプル「(A社,1.2,A社,DC)」「(B社,1.3,B社,GC)」が出力されます。

このように，特定時刻での結合処理の結果は，入力リレーションのタプルの生存期間に応じたタプルとして，出力リレーションに抽出されます。

2.3.3 集合関数による演算処理の例

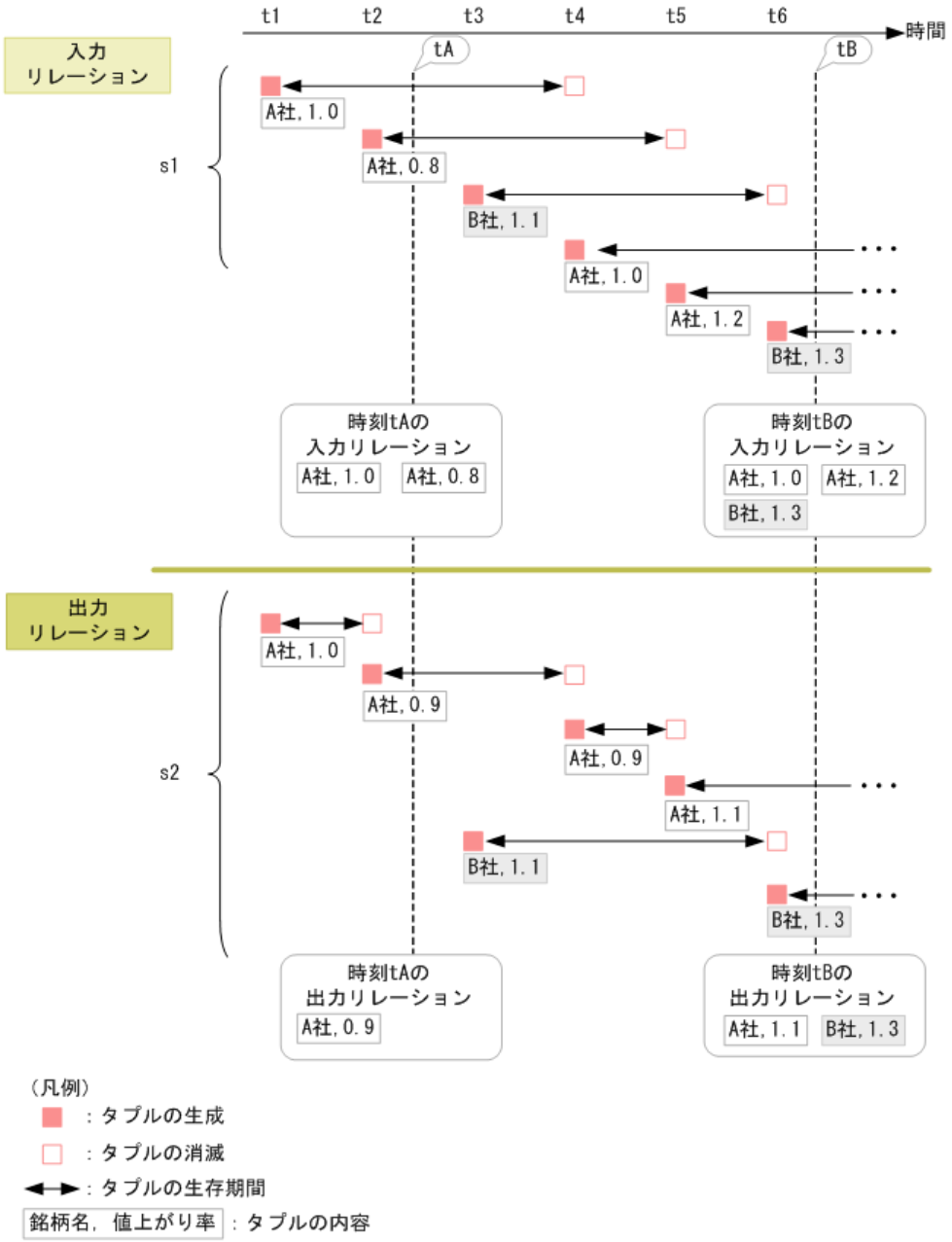
ここでは，「銘柄名」と「値上がり率」という内容を持つタプルに対して，GROUP BY 句と集合関数AVG を使用して，銘柄名別に値上がり率の平均値を算出する関係演算の例を示します。

CQL の定義例を次に示します。なお，実際の CQL の定義では，列名は半角英数字で指定してください。また，AVG には相関名を指定してください。

```
REGISTER QUERY q
   SELECT AVG(値上がり率)
   FROM s1[ROWS 3] GROUP BY 銘柄名;
```

この CQL を定義した場合の入力リレーションと出力リレーションの例を次の図に示します。

図 2-12　集合関数による演算処理の例



図の横軸は時間軸です。右に行くほど時間が経過しています。t1～t6 は，タプルが到着した時刻を示します。

この例は，GROUP BY 句で指定した銘柄別に，値上がり率を対象に集合演算AVG（平均値の算出）を指定しています。なお，入力リレーションは，[ROWS 3]の指定による，期間付きのリレーションです。

入力リレーションの時刻 *tA* での集合は「(A社,1.0)(A社,0.8)」となり，銘柄名別の平均値は「(A社,0.9)」となります。

同様に，時刻 *tB* での集合は「(A社,1.0)(A社,1.2)(B社,1.3)」となり，銘柄名別の平均値は，「(A社,1.1)(B社,1.3)」となります。

このように，特定時刻での銘柄名別の値上がり率の平均値は，入力リレーションのタプルの生存期間に応じたタプルとして，出力リレーションに抽出されます。

2.3.4 結合処理とROWSウィンドウを併用する場合の注意事項

ここでは，結合処理とROWSウィンドウを併用する場合の注意事項について説明します。

結合処理の結果として出力されるタプルの数がROWSウィンドウで指定したタプル数よりも多い場合，処理結果はROWSウィンドウで指定した数分だけ出力され，すべての処理結果が出力されません。

この問題は，次の2つの条件を満たすクエリを指定した場合に発生します。

1. あるクエリで，結合演算によって同一時刻を持つ複数のデータを生成する。
2. 別のクエリで，1.のクエリの結果を入力して処理を実行するが，このクエリのROWSウィンドウで指定した生存タプル数が1.で生成されるデータ数よりも少ない。

この条件に該当するクエリの例を示します。

```
REGISTER STREAM s1 (c1 INT);
REGISTER STREAM s2 (c2 INT);
REGISTER QUERY q1 ISTREAM(        条件1.のクエリ
  SELECT * FROM s1[ROWS 3], s2[ROWS 3]);
REGISTER QUERY q2 ISTREAM(        条件2.のクエリ
  SELECT * FROM q1[ROWS 1]);
```

このクエリに対して，時刻*t*に，ストリームs1またはs2へのタプルが到着した場合を想定します。クエリq1では，2つのストリームデータを入力して，結合演算を実施します。入力ストリームデータとして[ROWS 3]を指定しているため，時刻*t*での結合演算の結果としては，同一時刻を持つ3つのタプルが生成されます。

クエリq2でROWSウィンドウに指定した同時生存タプル数（1）は，クエリq1で生成されるデータ数（3）よりも少ないため，時刻*t*での生存タプルは，ウィンドウに到着したタプルのうち最後の1つのタプルだけになります。クエリq1の処理結果のうち，残り2つに対するq2の処理結果は出力されません。

この例の場合，クエリq2のROWSウィンドウで3以上の数を指定するなど，CQLを見直す必要があります。

2.4　ストリーム化演算による出力ストリームデータへの変換

この節では，ストリーム化演算による出力ストリームデータへの変換について説明します。

ストリーム化演算は，出力リレーション内のデータの変化に応じて，分析結果をストリームデータとして出力する方法を指定する演算です。ストリーム化演算によって，次の 3 種類のタプルをストリームデータとして出力できます。

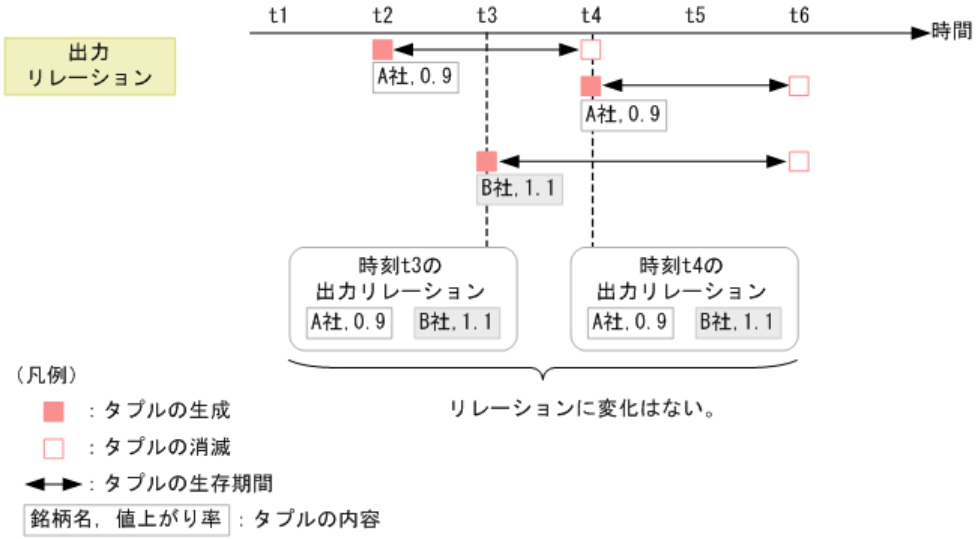
- 出力リレーションに追加されたタプル
- 出力リレーションから削除されたタプル
- 出力リレーション内のタプルの集合

出力リレーションの変化

出力リレーション内のデータの変化とは，リレーションとしての変化を意味します。出力リレーション内のタプルが変化しても，結果の内容が同じであれば，変化したとは見なされません。

例を示します。

図 2-13　リレーションが変化しない例



図の場合，時刻 t3 では，時刻 t2 の出力リレーション「(A社,0.9)」に対してタプル「(B社,1.1)」が追加されたことに伴い，出力リレーションが「(A社,0.9)(B社,1.1)」に変化します。一方，時刻 t4 では，t2 に生成された「(A社,0.9)」が消滅しますが，同時に別のタプルとして「(A社,0.9)」が生成されるため，出力リレーションの内容は「(A社,0.9)(B社,1.1)」のままになります。この場合，時刻 t4 で出力リレーションは変化していないものと見なされます。

2.4.1　ストリーム化演算の種類

ストリーム化演算の種類と指定方法について説明します。

(1) 出力リレーションに追加されたタプルを出力する指定（ISTREAM）

ISTREAM は，出力リレーションにタプルが追加されたときに，そのタプルを出力する演算です。指定例を次に示します。

```
ISTREAM( SELECT … FROM s1 [ROWS 10] … )
```

(2) 出力リレーションから削除されたタプルを出力する指定（DSTREAM）

DSTREAM は，出力リレーションからタプルが削除されたときに，そのタプルを出力する演算です。指定例を次に示します。

```
DSTREAM( SELECT … FROM s2 [ROWS 10] … )
```

(3) 出力リレーション内のタプルの集合を一定間隔で出力する指定（RSTREAM）

RSTREAM は，出力リレーション内のタプルの集合を一定間隔で出力する演算です。指定例を次に示します。

```
RSTREAM[1 SECOND]( SELECT … FROM s3 [ROWS 10] … )
```

重要

タイムスタンプモードをデータソースモードにしている場合，RSTREAM を指定しても意図した内容が出力されないおそれがあります。

データソースモードでは，タプルが到着したタイミングでリレーションの時刻を進めます。例えば，RSTREAM で出力間隔を 1 分とした場合でも，「09:01:00」にタプルが到着したあと，次のタプルが「10:00:00」に到着した場合，09:02 から09:59 の間，ストリームデータは出力されません。「10:00:00」のタプルが到着したタイミングで，1 時間分のタプルがまとめて出力されてしまいます。

データソースモードでRSTREAM を使用する場合は，この動作を理解した上で，注意して使用してください。

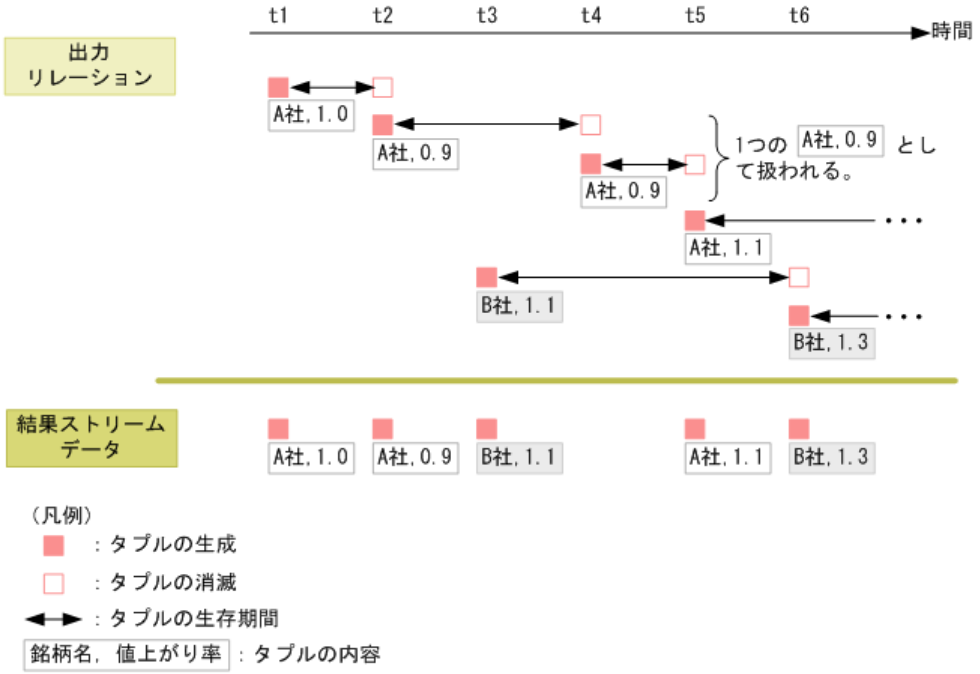
2.4.2 ストリーム化演算の例

ここでは，それぞれのストリーム化演算を指定した場合に，結果として出力されるストリームデータの例について説明します。なお，ここで示すのは，ウィンドウ演算に「PARTITION BY *銘柄名* ROWS 1」を指定した場合の例です。

(1) 出力リレーションに追加されたタプルを出力する指定の例（ISTREAM）

ISTREAM を指定した場合の出力リレーションと結果ストリームデータの例を次の図に示します。

図 2-14 ISTREAM を指定した場合の出力リレーションと結果ストリームデータの例



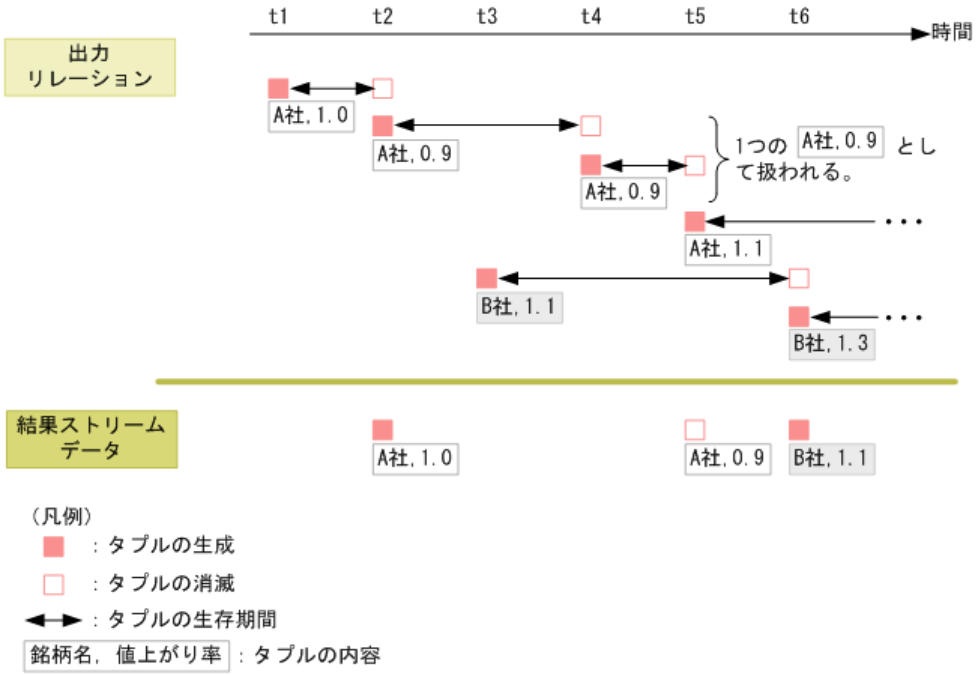
出力リレーションにタプルが追加されたときに，結果ストリームデータとして対応するタプルが出力されます。この例の場合は，出力リレーションの時刻 t1 にタプル「(A社,1.0)」が追加された時点で，結果ストリームデータにも時刻 t1 にタプル「(A社,1.0)」が出力されます。以降，時刻 t2，時刻 t3 にタプルが追加されるたびに，結果ストリームデータに対応するタプルが出力されます。

なお，時刻 t4 では，タプル「(A社,0.9)」が追加されるのと同時に時刻 t2 で追加されたタプル「(A社,0.9)」が削除されるため，リレーションとしては変化しません。このため，結果ストリームデータには何も出力されません。

(2) 出力リレーションから削除されたタプルを出力する指定の例（DSTREAM）

DSTREAM を指定した場合の出力リレーションと結果ストリームデータの例を次の図に示します。

図 2-15　DSTREAM を指定した場合の出力リレーションと結果ストリームデータの例



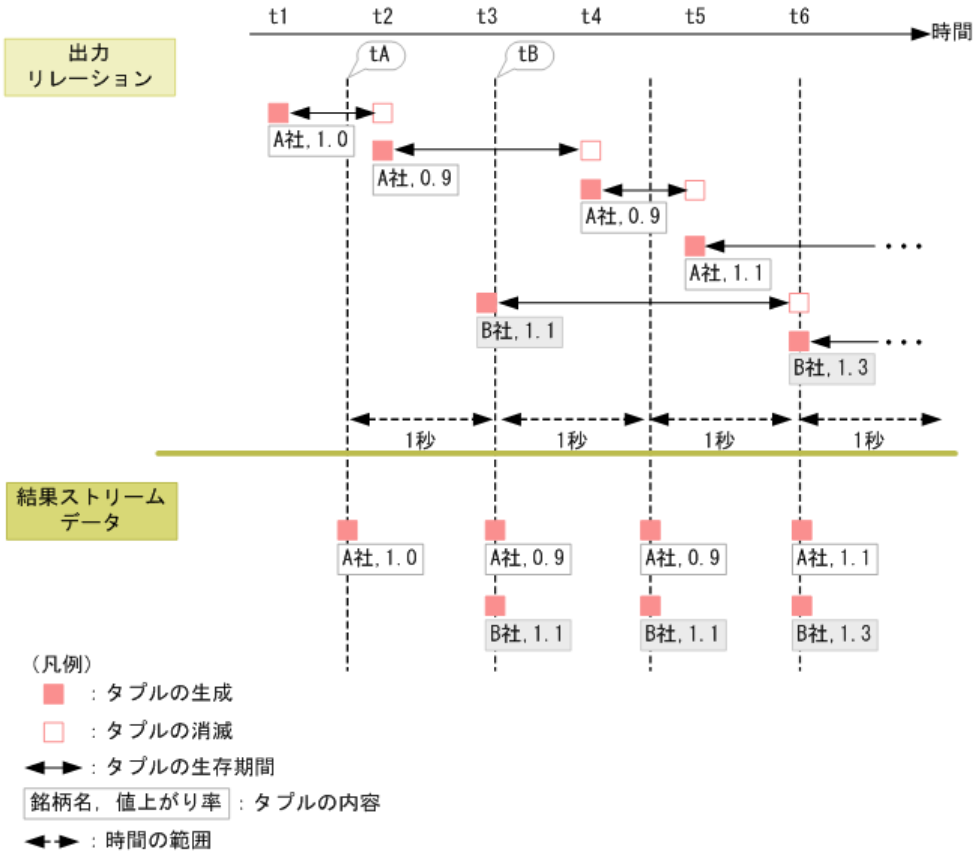
出力リレーションからタプルが削除されたときに，結果ストリームデータとしてタプルが出力されます。この例の場合は，出力リレーションのタプル「(A社,1.0)」が時刻 t2 で削除された時点で，結果ストリームデータにタプル「(A社,1.0)」が出力されます。

なお，時刻 t4 では，タプル「(A社,0.9)」が削除されるのと同時にタプル「(A社,0.9)」が追加されるため，リレーションとしては変化しません。このため，ISTREAM を指定した場合と同様に，結果ストリームデータには何も出力されません。

(3)　出力リレーション内のタプルの集合を一定間隔で出力する指定の例 (RSTREAM)

RSTREAM を指定した場合の出力リレーションと結果ストリームデータの例を次の図に示します。

図 2-16　RSTREAM を指定した場合の出力リレーションと結果ストリームデータの例



出力リレーション内の集合が，結果ストリームデータのタプルとして 1 秒間隔で出力されます。時刻 tA には，出力リレーション内には「(A社,1.0)」だけしかないため，結果ストリームデータには「(A社,1.0)」だけが出力されます。1 秒経過後の時刻 tB では，出力ストリームの集合が「(A社,0.9)(B社,1.1)」になっているため，結果ストリームデータには「(A社,0.9)(B社,1.1)」が出力されます。

以降，1 秒ごとの出力リレーションの内容が，結果ストリームデータに出力されます。

2.5 時刻解像度の指定によるメモリ使用量増加の抑止

この節では，RANGE ウィンドウを指定して大量の受信データを処理する場合などに，時刻解像度を指定することによってメモリ使用量の増加を抑止する方法について説明します。

RANGE ウィンドウを指定したクエリの場合，一定時間内に受信したデータがすべて処理の対象になります。このため，指定する時間やデータの量によっては，メモリ使用量が増加して，システムの動作に影響が出るおそれがあります。

これに対して，時刻解像度を指定することで，RANGE ウィンドウの処理対象となるデータの量を一定に抑えられます。

時刻解像度の指定とは，RANGE ウィンドウで指定した範囲のリレーションを任意の単位時間に分割して，分割した時間ごとに演算処理を実行する機能のことです。なお，分割したリレーションのことをメッシュといいます。分割する間隔（ミリ秒単位）をメッシュ間隔といいます。

RANGE ウィンドウで指定した時間の範囲で扱う受信データが多い場合，あらかじめメッシュ単位で演算を実行し，その結果を擬似タプルとして保持します。RANGE ウィンドウでは，この擬似タプルを処理対象とすることで，RANGE ウィンドウ内のタプル数を一定に保ちます。これによって，メモリ使用量の増加を抑止できます。

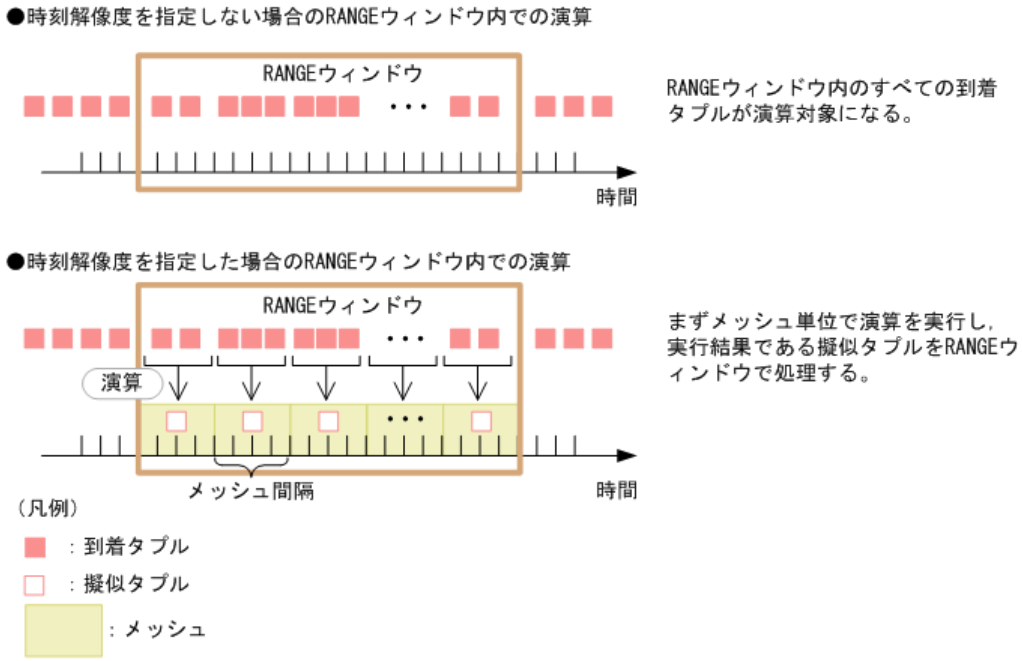
例えば，RANGE ウィンドウ内のタプル数が 10 万個ある場合に，時刻解像度を指定して，10 万個のタプルをあらかじめ 8 個の擬似タプルにしておけば，メモリ内に保持しておくタプルの個数は 8 個に抑えられます。

時刻解像度は，次の 2 つの条件を満たすクエリで指定できます。

- RANGE ウィンドウを使用している。
- 関係演算として，次のどれかの集合関数を使用している。
  SUM（合計値の算出）
  MAX（最大値の算出）
  MIN（最小値の算出）

時刻解像度を指定しない場合と指定した場合の演算の違いを次に示します。

図 2-17　時刻解像度を指定しない場合と指定した場合の演算の違い



時刻解像度を指定しない場合，演算の対象は，RANGE ウィンドウ内のすべてのタプルになります。例えば，RANGE ウィンドウの範囲として 1 分間を指定している場合に，1 分間に到着するタプルの総数が 10 万個ある場合，演算の対象は 10 万個になります。最大値を求める演算の場合，10 万個のタプルをすべて比較して，最大値を求める必要があります。

一方，時刻解像度を指定した場合，演算の対象は，擬似タプルになります。擬似タプルは，あらかじめ指定したメッシュ間隔に応じて演算処理が実行された結果です。例えば，メッシュ間隔の指定によってRANGE ウィンドウを 8 個に分割していた場合，擬似タプルは 8 個作成されます。最大値を求める演算の場合，8 個の擬似タプルの内容を比較すれば，最大値を求められます。このように，時刻解像度を指定することで，RANGE ウィンドウ内に保持するタプル数を削減し，メモリ使用量の増加を抑えられます。

メモ

時刻解像度は，ストリームデータの送信レートが高く，RANGE ウィンドウ内に生存する到着タプルが多い場合に指定すると効果的です。

時刻解像度を指定することで，Java ヒープ領域内のメモリ使用量を一定に保つことができるので，メモリ使用量の単調増加を抑止できます。

時刻解像度は，単位時間当たりの最大値（MAX），最小値（MIN），合計値（SUM）を求める処理ロジックを持つクエリに適用できます。

時刻解像度の指定例については，「2.6.2　時刻解像度を指定した定義例」を参照してください。

2.6　定義例

この節では，CQL によるクエリの定義例について説明します。

2.6.1　基本的なクエリの定義例

ここでは，ウィンドウ演算，関係演算およびストリーム化演算を使用した，基本的なクエリの定義例として，株価情報を対象としたクエリの定義例を示します。

この例で扱う株価情報ストリームデータの例を次の図に示します。

図 2-18　株価情報ストリームデータの例

株価情報ストリームデータ（stock）

時刻	銘柄コード (stockID)	銘柄名 (stockName)	株価 (currentPrice)	出来高 (tradingVolume)
10:00:00	6501	A社	780	12442000
10:00:05	1468	B社	1650	3318000
10:00:10	2282	C社	1518	1347000
…	…	…	…	…

タイムスタンプ　　株価情報

この例で扱うストリームデータは，銘柄コード（stockID），銘柄名（stockName），株価（currentPrice）および出来高（tradingVolume）という株価情報を持つデータに対して，Hitachi Streaming Data Platform でタイムスタンプを設定したストリームデータです。

このストリームデータを使用して，株価の値上がり率を計算します。値上がり率は，現在のデータと 1 分前のデータを比較して算出します。

値上がり率を算出するためには，次の 2 種類のクエリを使用します。

1 分間データ算出クエリ

```
REGISTER QUERY stockDStream
DSTREAM ( SELECT * FROM stock[RANGE 1 MINUTE] );
```

- 範囲を 1 分間と指定したRANGE ウィンドウによって，入力リレーションのタプルを 1 分間保持します。
- DSTREAM を指定して，1 分間の生存期間が終了したタプルを出力します。これによって，現在から 1 分前のデータが，ストリームデータとして出力されます。

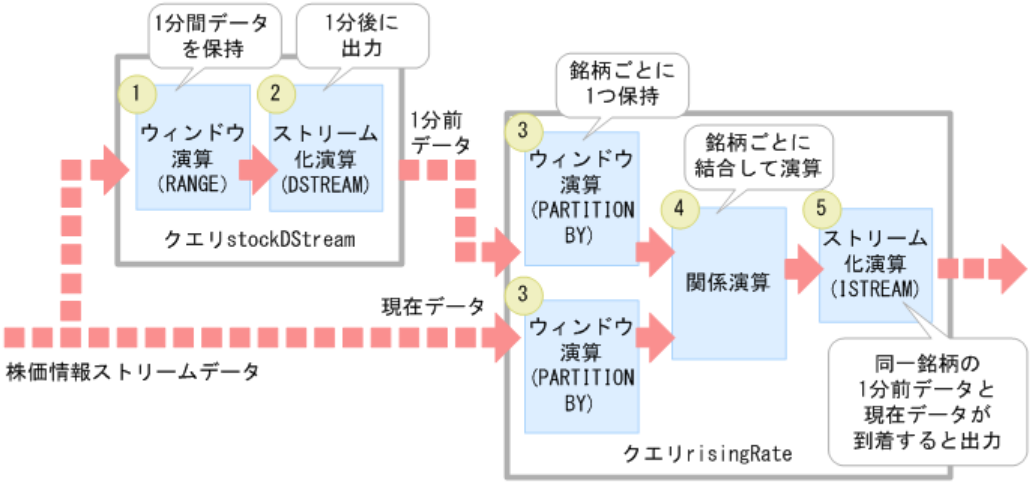
値上がり率計算クエリ

```
REGISTER QUERY risingRate
ISTREAM ( SELECT stock.stockID, stock.stockName,
          stock.currentPrice / stockDStream.currentPrice AS stockRate
          FROM stock[PARTITION BY stock.stockID ROWS 1],
          stockDStream[PARTITION BY stockDStream.stockID ROWS 1]
          WHERE stock.stockID = stockDStream.stockID );
```

- PARTITION BY ウィンドウを使用して，銘柄ごとに最新の 1 件のタプルを保持します。
- 現在の株価情報ストリームデータと 1 分前のストリームデータを統合して，値上がり率を計算します。

これらのクエリを使用した演算処理の流れを次の図に示します。

図 2-19　株価値上がり率を算出する演算処理の流れ



処理の流れについて説明します。説明の番号は図中の番号と対応しています。

1. [クエリstockDStream のウィンドウ演算]

   ウィンドウ演算を実行します。

   「FROM stock」の指定によって入力したストリームstock を，「[RANGE 1 MINUTE]」の指定によって 1 分間保持します。

2. [クエリstockDStream のストリーム化演算]

   1 分経過したデータは，ストリーム化演算「DSTREAM」によってストリームデータ（1 分前データ）として出力されます。

3. [クエリrisingRate のウィンドウ演算]

   ストリームstock と，2.で出力した結果ストリーム（1 分前データ）の 2 種類のデータを「FROM stock…, stockDStream…」の指定によって入力します。

   「[PARTITION BY… ROWS 1]」の指定によって，入力データを銘柄コード（stockID）ごとにグルーピングした上で，銘柄コードごとの最新 1 件のデータを保持します。これによって，銘柄コードごとに，次のデータが保持されます。

   - 入力ストリームstock による現在の株価データのうち，最新の 1 件
   - 入力ストリームstockDStream（クエリstockDStream の結果ストリーム）による 1 分前の株価データのうち，最新の 1 件

4. [クエリstockDStream の関係演算]

   銘柄コードごとに，銘柄コード，銘柄名，および値上がり率（*現在の株価/1 分前の株価*）を結合します。

5. [クエリstockDStream ストリーム化演算]

同一銘柄の 1 分前データと現在データが到着したタイミングで，4.の結合データを結果ストリームとして出力します。

2.6.2　時刻解像度を指定した定義例

ここでは，時刻解像度を指定したクエリの定義例として，次の 2 種類の例について説明します。

- 1 つの集合関数を定義する例
- 複数の集合関数を定義する例

この例では，1 分間のRANGE ウィンドウ内にある到着タプルに対して，集合関数による関係演算を実行します。この例で扱うストリームは次のとおりです。

時刻解像度の指定例で扱うストリーム

- ストリーム名は「stock」である。
- 「price」という名称の列がある。

1 つの集合関数を定義する例

ストリームstock に対して，RANGE ウィンドウ内にある到着タプルから列名price の合計値を求める場合の定義例を次に示します。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));    …(1)
REGISTER QUERY_ATTRIBUTE q1 STREAM_NAME=stock PERIOD=300MS TARGETS=SUM(price);    …(2)
REGISTER QUERY q1    …(3)
ISTREAM(
  SELECT name, SUM (price) AS s1
  FROM stock[RANGE 1 MINUTE]
  GROUP BY name
);
```

注

(1)～(3)は次に示す説明と対応している番号です。実際の定義には不要です。

時刻解像度を指定するクエリの名称は，(3)で指定しているq1 です。このため，(2)のQUERY_ATTRIBUTE 文でも，クエリ名としてq1 を指定します。

この例は，(1)で指定しているストリーム名stock について，時刻解像度を指定して列名price の合計値を求める例です。このため，(2)のSTREAM_NAME=のデータ識別子にはstock を指定して，TARGETS=には集合関数SUM(price)を指定します。

合計値ではなく，最大値または最小値を求める場合には，集合関数MAX またはMIN を指定してください。

複数の集合関数を定義する例

ストリームstock に対して，RANGE ウィンドウ内にある到着タプルから列名price の合計値，最大値および最小値を求める場合の定義例を次に示します。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
ISTREAM(
  SELECT name, price AS price0, price AS price1, price AS price2
  FROM stock[NOW]
);
REGISTER QUERY_ATTRIBUTE q1 STREAM_NAME=q0 PERIOD=300MS
TARGETS=SUM(price0),MAX(price1),MIN(price2);
REGISTER QUERY q1
ISTREAM(
  SELECT q0.name, SUM(q0.price0) AS sum_price,
    MAX(q0.price1) AS max_price, MIN(q0.price2) AS min_price
  FROM q0[RANGE 1 MINUTE]
GROUP BY q0.name
);
```

時刻解像度を指定して複数の集合演算を実行する場合，同じデータ識別子を持つ同一名称の列名に対して，複数の集合演算は実行できません。例えば，データ識別子stock の列名price のクエリに対して，「TARGETS=SUM(price),MAX(price),MIN(price)」のような指定はできません。

複数の集合関数を指定する場合は，集合関数の対象とする列名をそれぞれ用意する必要があります。この例では，SUM，MAX，MIN それぞれの対象とする列名として，列名price をprice0，price1，price2 という列名に置き換えた上で，それぞれの集合関数による演算を実行しています。

3 CQL の基本項目とデータ型

この章では，CQL の基本項目とデータ型について説明します。

3.1　CQL の記述規則

クエリは，CQL を使用して定義します。CQL は，次の項目（**基本項目**）を組み合わせて処理を記述します。

- キーワード
- 数値
- 区切り文字
- 名前
- 定数

キーワードは，「SELECT」や「WHERE」のように，データ処理のための機能を示す文字列です。キーワードに対して，処理対象や処理範囲などを定義するために指定する数値，区切り文字，名前，定数などは，**オペランド**といいます。

これらの基本項目を指定する場合は，使用できる文字，データ型などに留意する必要があります。

ここでは，CQL でクエリを定義する際の，基本項目の記述形式および使用できる文字について説明します。また，以降の CQL の文法説明で使用する記号についても説明します。

3.1.1　CQL の記述形式

CQL は，次の内容に従って記述してください。

- CQL は，フリーフォーマット形式です。オペランドは，各 CQL の形式で記述している順序に従って指定します。CQL の形式については，「4.　CQL リファレンス」を参照してください。
- CQL の文は，セミコロン（;）で区切って記述します。

  1 つの文は複数行にわたって記述できます。1 行に複数の文を記述することはできません。

  正しい記述例と誤った記述例を次に示します。

  （正しい記述例）

  ```
  REGISTER STREAM
      s1(id INT,name VARCHAR(10));
  REGISTER QUERY q1
      SELECT s1.name FROM s1[ROWS 10];
  ```

  （誤った記述例）

  ```
  REGISTER STREAM s1(id INT); REGISTER QUERY q1 SELECT * FROM s1[ROWS 10];
  ```

  2 つのREGISTER 文を 1 行に記載しているので，誤りとなります。
- 注釈は，ダブルスラッシュ（//）で指定します。ダブルスラッシュを指定した場合，ダブルスラッシュを含めた以降の文は注釈として扱われます。この場合，注釈の有効範囲は 1 行です。1 つの定義文の途中に注釈を挿入することはできません。

正しい記述例と誤った記述例を次に示します。

（正しい記述例）

```
// 注釈
REGISTER STREAM
    s1(id INT,name VARCHAR(10));   // 注釈
```

（誤った記述例）

```
REGISTER QUERY q1   // 注釈
    SELECT s1.name FROM s1[ROWS 10];
```

REGISTER QUERY 文の途中に注釈を挿入しているので，誤りとなります。

なお，文字列中に注釈以外の用途で「//」を指定する場合，「//」が行の先頭にならないように記述してください。正しい記述例と誤った記述例を次に示します。

（正しい記述例）

```
REGISTER QUERY q1 SELECT * FROM s1[NOW] WHERE s1.c1='ab//cd';
```

（誤った記述例）

```
REGISTER QUERY q1 SELECT * FROM s1[NOW] WHERE s1.c1='ab
//cd';
```

行の先頭が「//」となっているため，誤りとなります（注釈行と見なされてしまいます）。

3.1.2 CQL 指示文

(1) 説明

CQL 指示文では，CQL を処理するエンジンを指定します。

C で作成した外部定義関数を使用する場合，外部定義関数を使用する範囲を CQL 指示文で指定する必要があります。CQL 指示文で指定した範囲に含まれる操作系 CQL は，acceleration CQL エンジンで処理されます。

(2) 形式

CQL 指示文の指定形式を次に示します。

```
Δ0#SDPOPTIONΔ1(指示文)Δ0(改行コード)
```

（凡例）

Δ_0：0 個以上の半角スペース

Δ_1：1 個以上の半角スペース

次の指示文を使用できます。

表 3-1　指示文

指示文	説明
EXTERNAL C BEGIN	C 外部定義関数の先頭に指定します。 この指示文以降のクエリは，acceleration CQL エンジンに登録されます。
EXTERNAL C END	C 外部定義関数の末尾を指定します。 この指示文よりもあとに指定したクエリは，CQL エンジンで処理されます。

(3)　例

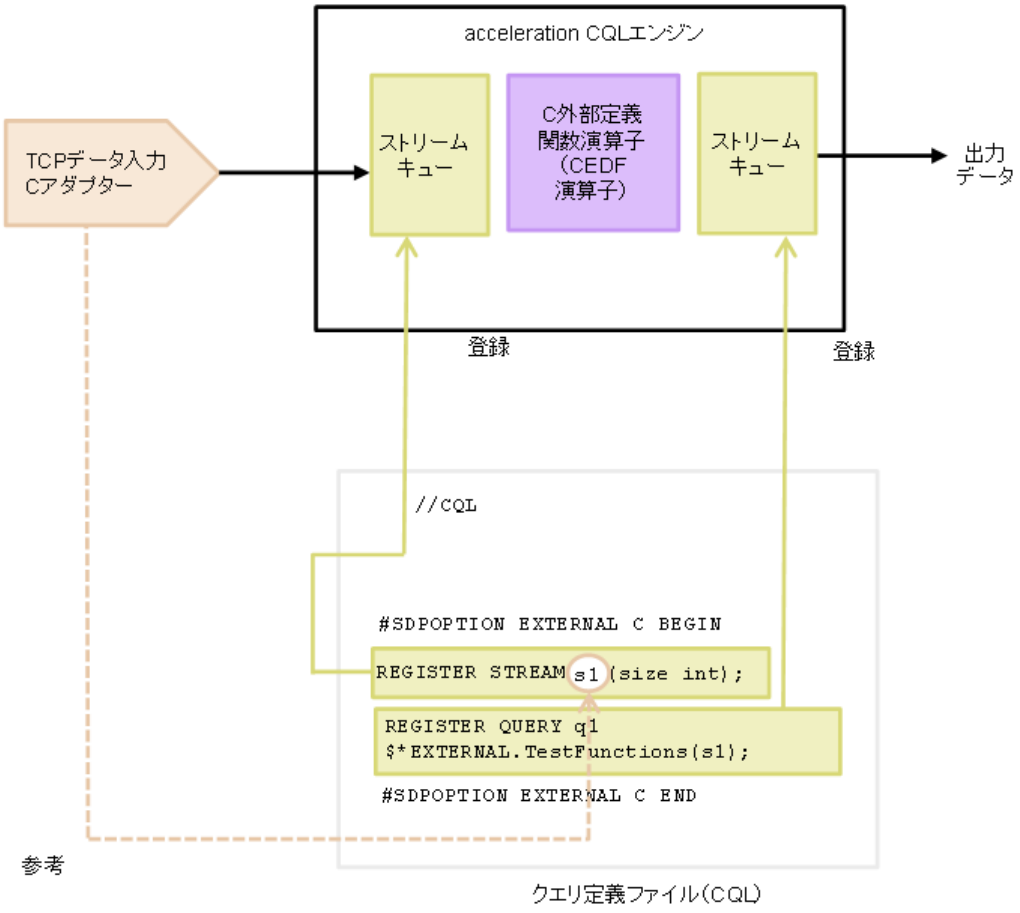
C 外部定義関数の CQL を入力するには，CQL を指示文（#SDPOPTION EXTERNAL C BEGIN, #SDPOPTION EXTERNAL C END）で囲みます。

CQL を指示文で囲む例を次に示します。

```
#SDPOPTION EXTERNAL C BEGIN
REGISTER STREAM s1 (size int);
REGISTER QUERY q1 $*EXTERNAL.TestFunction(s1);
#SDPOPTION EXTERNAL C END
```

REGISTER STREAM 句を指示文で囲むと，ストリームキューが acceleration CQL エンジンに挿入されます。「TCP データ入力 C アダプター」を使用するには，外部領域（CQL 指示文で囲まれた範囲）にストリームを定義します。アダプター，エンジン，および CQL の関係を次の図に示します。

図 3-1　acceleration CQL エンジンの概要



ポイント

次の規則に従います。

- C 外部定義関数の CQL には，入力ストリームと出力ストリームを最大で 1 つずつ指定できます。

  誤ったコーディングの例を次に示します。

```
#SDPOPTION EXTERNAL C BEGIN
...
REGISTER QUERY output1 $*EXTERNAL.TestFunction(input1, input2);
#SDPOPTION EXTERNAL C END
```

  複数の入力ストリームを入力するには，1 つのREGISTER STREAM 句に各入力ストリームのスキーマ指定文字列を定義します。

- 外部領域には，外部定義ストリーム間演算関数以外の操作系 CQL は定義できません。

  誤ったコーディングの例を次に示します。

```
#SDPOPTION EXTERNAL C BEGIN
...
REGISTER QUERY q istream(select * from input[now]);
#SDPOPTION EXTERNAL C END
```

- クエリ定義ファイルには，外部領域は 1 つだけ出現します。

  誤ったコーディングの例を次に示します。

```
#SDPOPTION EXTERNAL C BEGIN
...
REGISTER QUERY q1 $*EXTERNAL.TestFunction(s1);
#SDPOPTION EXTERNAL C END

#SDPOPTION EXTERNAL C BEGIN
...
REGISTER QUERY q2 $*EXTERNAL.TestFunction(q1);
#SDPOPTION EXTERNAL C END
```

- 外部領域には，クエリは 1 つだけ出現します。

  誤ったコーディングの例を次に示します。

```
#SDPOPTION EXTERNAL C BEGIN
...
REGISTER QUERY q2 $*EXTERNAL.TestFunction(q1);
REGISTER QUERY q3 $*EXTERNAL.TestFunction(q2);
#SDPOPTION EXTERNAL C END
```

  複数のクエリを使用するには，ファイルを分割し，カスケーディングアダプターを使用してファイルを接続します。

- すべての CQL 文は，外部領域内に指定する必要があります。

  誤ったコーディングの例を次に示します。

```
REGISTER STREAM s1 (c1 int);
#SDPOPTION EXTERNAL C BEGIN
REGISTER QUERY q1 $*EXTERNAL.TestFunction(s1);
#SDPOPTION EXTERNAL C END
```

3.1.3　CQL で使用できる文字

ここでは，CQL で使用できる文字について説明します。

(1)　使用できる文字コード

CQL で使用できる文字コードは UTF-8 です。

(2)　使用できる文字

CQL で使用できる文字を次の表に示します。

表 3-2　CQL で使用できる文字

項番	種別	使用できる文字
1	名前	半角英数と下線（_）が使用できます。ただし，先頭に使用できる文字は，半角英文字だけです。名前の指定については，「3.2.4　名前の指定」を参照してください。

項番	種別	使用できる文字
2	文字列定数	半角文字コードおよび全角文字コードが使用できます。
3	上記以外	次に示す半角文字コードの文字および特殊記号が使用できます。 • 半角文字コード 英大文字（A～Z），英小文字（a～z），数字（0～9），空白，下線文字（_） • 特殊記号 コンマ（,）：選択リスト，値式など ピリオド（.）：浮動小数点定数，列指定，日付／時刻データなど コロン（:）：時刻／時刻印データ ハイフン（-）：日付／時刻印データ 引用符（'）：文字列の囲み 左括弧（(），右括弧（)）：リレーション式，キャスト指定の囲みなど 小なり演算子（<），大なり演算子（>），等号演算子（=），感嘆符（!）：比較演算子 正符号（+），負記号（-），斜線（/）：算術演算子 アスタリスク（*）：算術演算子，すべての列出力など セミコロン（;）：文の区切り 角括弧（[ ]）：時間指定の囲み，タブコード，改行コード，復帰コード 斜線（/）＋斜線（/）：注釈 ドル記号，アスタリスクの組み合わせ（$*）：外部定義関数

3.1.4 CQL 文法説明で使用する記号

CQL の文法説明で使用する記号の用法を次の表に示します。以降，3 章および 4 章では，これらの記号を使用して CQL の文法を説明します。

表 3-3 CQL の文法説明で使用する記号

記号	意味	例
{ }	この記号で囲まれた複数の項目から，1 つを選択することを示します。	{*文字列定数* | *数定数*} 文字列定数，または数定数のどちらかを選択して記述します。
〔 〕	この記号で囲まれた項目は省略できることを示します。複数の項目が並べて記述されている場合は，すべてを省略するか，記号 { } と同じくどれか 1 つを選択します。	〔SECOND | MILLISECOND〕 すべてを省略するか，SECOND またはMILLISECOND のどちらかを選択して記述します。
…	この記号の直前に示された項目を繰り返し複数個指定できることを示します。	*リレーション参照*〔,*リレーション参照*〕… リレーション参照を繰り返し複数個記述します。
'(' ')'	この記号で囲まれた項目は，( )を省略しないでそのまま記述することを示します。	'('TINYINT')' TINYINT の前後を( )で囲んで記述します。
'[' ']'	この記号で囲まれた項目は，[ ]を省略しないでそのまま記述することを示します。	'['*ウィンドウ指定*']' ウィンドウ指定の前後を[ ]で囲んで記述します。

記号	意味	例
::=	この記号の左にあるものを右にあるもので定義することを示します。	*選択リスト*::=*選択式*〔,*選択リスト*〕
▲*n*	*n* 個以上の区切り文字を示します。*n* が省略されている場合，*n*=1 を仮定します。	NOT｛▲0'(*'探索条件'*)'｜▲*比較述語*｝ NOT と(*探索条件*)の間に 0 個以上の区切り文字，またはNOT と比較述語の間に 1 個以上の区切り文字を記述します。

3.2　CQL の基本項目の指定方法

この節では，CQL の基本項目の指定方法について説明します。

3.2.1　キーワードの指定

CQL 文の名称（SELECT，UNION など）のように，機能を使用するために指定する語を**キーワード**といいます。

キーワードの意味については，「4.　CQL リファレンス」を参照してください。

キーワードの指定例を次の図に示します。

図 3-2　キーワードの指定例

ほとんどのキーワードは，システムの予約語として登録されているため，決められた位置以外には指定できません。

ただし，予約語として登録されていないキーワードの文字列は，名前としても使用できます。その場合，大文字，小文字の区別はされません。

予約語としてシステムに登録されている文字列を，先頭文字のアルファベットごとに次の表に示します。

表 3-4　予約語としてシステムに登録されている文字列

先頭文字	予約語
A	ALL，AND，ANY，AS，ASC，AST，AVG
B	BETWEEN，BIGINT，BIGINT_TOSTRING，BINARY，BOOLEAN，BY
C	CASE，CHAR，CHARACTER，CON，CONCAT，CORREL，COUNT，COVAR，COVAR_POP
D	DABS，DACOS，DASIN，DATAN，DATAN2，DATE，DAY，DCEIL，DCOS，DCOSH，DDISTANCE，DDISTANCE3，DEC，DECIMAL，DECIMAL_TOSTRING，DEFAULT，DELETABLE，DELETE，DESC，DEXP，DFLOOR，DISTINCT，DLN，DLOG，DMOD，DNAN，DNEGATIVE_INFINITY，DOUBLE，DOUBLE_TOSTRING，DPI，DPOSITIVE_INFINITY，DPOWER，DROUND，DSIN，DSINH，DSQRT，DSTREAM，DTAN，DTANH，DTODEGREES，DTORADIANS
E	ELSE，END，EOF，EQ，EXISTS
F	FALSE，FLOAT，FROM
G	GE，GT，GROUP
H	HAVING，HOUR

先頭文字	予約語
I	IABS, IDSTREAM, IMOD, IN, INSERT, INSTANT, INT, INT_TOSTRING, INTEGER, IPOWER, IS, ISTREAM
J	予約語はありません。
K	予約語はありません。
L	LABS, LATEST, LE, LENGTH, LEQ, LGE, LGT, LIKE, LIMIT, LLE, LLP, LLT, LMOD, LP, LPOWER, LRP, LT
M	MAX, MILLISECOND, MIN, MINUTE, MIS
N	NE, NOT, NOW, NROUND, NULL, NUMBER_TODATE, NUMBER_TOTIME, NUMERIC
O	OR, ORDER, OTHER
P	P_INT, PARTITION, PLS, POM, PR, PREV
Q	予約語はありません。
R	RABS, RACOS, RANGE, RANKING, RASIN, RATAN, RATAN2, RCEIL, RCOS, RCOSH, RDISTANCE, RDISTANCE3, REAL, REAL_TOSTRING, RECENT, REGEXP_FIRSTSEARCH, REGEXP_REPLACE, REGEXP_SEARCH, RELATIONAL, REXP, RFLOOR, RLN, RLOG, RMOD, RNAN, RNEGATIVE_INFINITY, ROWS, RPI, RPOSITIVE_INFINITY, RPOWER, RROUND, RSIN, RSINH, RSQRT, RSTREAM, RTAN, RTANH, RTODEGREES, RTORADIANS
S	SECOND, SELECT, SMALLINT, SOME, SPLENGTH, SQU, STDDEV, STDDEV_POP, SUB, SUM
T	THEN, TIME, TIME_TONUMBER, TIMEDIFF, TIMESTAMP, TIMESTAMP_TONUMBER, TIMESTAMPDIFF, TINYINT, TRUE
U	UNBOUNDED, UNION, UNKNOWN, UPDATE
V	VAR, VAR_POP, VARBINARY, VARCHAR
W	WHEN, WHERE, WITH, WQU
X	予約語はありません。
Y	予約語はありません。
Z	予約語はありません。

3.2.2　数値の指定

CQL 中に記述する値式以外の数値（データ型の括弧中の指定，時間，行数など）は，符号なし整数の表記法に従って指定します。

3.2.3　区切り文字の挿入

(1)　区切り文字の種類

区切り文字として指定できるのは，次の文字です。

- 半角空白

- 復帰コード
- 改行コード
- タブコード

なお，区切り文字に空白を指定する場合は，半角空白を使用してください。全角空白は区切り文字と見なされません。

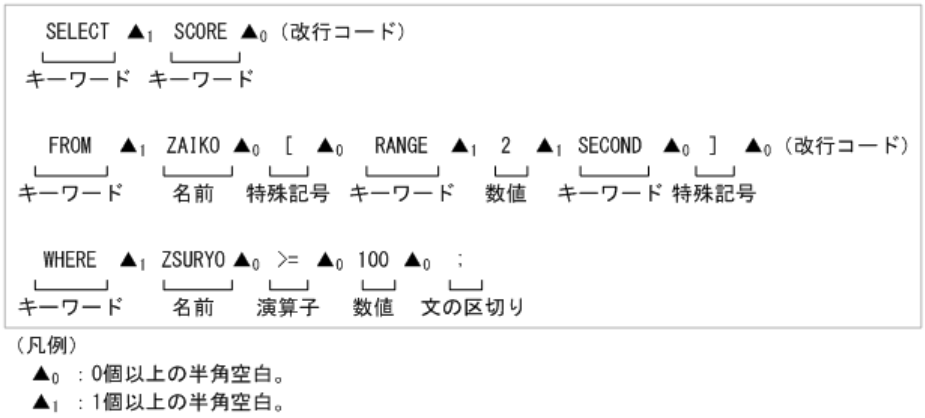
(2)　区切り文字を挿入する位置

区切り文字を挿入する位置を次に示します。

- キーワードとキーワードの間
- キーワードと名前の間
- キーワードと数値の間

区切り文字の挿入例を次の図に示します。

図 3-3　区切り文字の挿入例



(3)　区切り文字を挿入できる位置

区切り文字は，次に示す特殊文字の前後で，かつ「3.2.3　区切り文字の挿入」の「(4)　区切り文字を挿入できない位置」で挿入を禁止されていない位置に挿入できます。

「,」「.」「-」「+」「*」「'」「(」「)」「<」「>」「=」「^」「!」「#」「$」「/」「[」「]」「復帰コード」「改行コード」「タブコード」「;」

ただし，「'」を使用する文字列定数部分，および「>=」などの比較演算子の場合は，特殊記号と特殊記号の間に区切り文字を挿入できません。

区切り文字を挿入できる位置の例を次に示します。

(例 1) 特殊文字 (.) の前に空白を挿入

```
SCORE▲.NAME
```

（例 2）特殊文字（=）のあとに空白を挿入

```
SCORE=▲1
```

(4) 区切り文字を挿入できない位置

区切り文字を挿入できない位置を次に示します。

- キーワードの途中
  （例）S▲ELECT
- 名前の途中
  （例）ZA▲IKO
- 数定数の途中
  （例）678▲9
- 演算子の途中
  （例）<▲=
- 外部定義関数の記号「$*」の特殊記号と特殊記号の間
  （例）$▲*

3.2.4 名前の指定

名前の指定についての規則を次に示します。

- 名前に使用できる文字は，半角英数字と下線（_）です。
- 名前の先頭に使用できる文字は，半角英文字だけです。数字や下線（_）は使用できません。
- 名前の指定では，大文字と小文字は区別されません。すべて半角大文字として扱われます。
- 名前に空白を含めることはできません。
- CQL の予約語と重複した名前は指定できません（予約語でないキーワードとは重複できます）。
- 名前を引用符（”）で囲むことはできません。
- 名前として指定できる文字数は1～100 文字です。

3.2.5 名前の修飾

名前の修飾は，「*データ識別子.列名*」のように，ピリオド（.）によってデータ識別子と列名を連結して，名前を一意にするときに使用します。なお，データ識別子と列名を連結した名前の指定方法を，**列指定**といいます。

(1) データ識別子指定

データ識別子指定とは，1 つの CQL 文中に 2 つ以上のデータ識別子を指定する場合に，指定する列名がどのストリーム，リレーションまたは相関名に対応するかを一意にするための修飾子を使用した指定方法です。

次の指定形式に示すように，修飾子としてストリーム名，リレーション名，または相関名を指定します。

データ識別子の指定形式

```
データ識別子::={ストリーム名 | リレーション名 | 相関名}
```

ストリーム名およびリレーション名については，「4.4.13　リレーション参照」を参照してください。また，相関名については，「4.4.11　選択式」および「4.4.13　リレーション参照」を参照してください。

(2) 列指定

列指定とは，データ識別子で修飾された列名のことです。

複数のデータ識別子を指定した検索（2 つ以上のデータ識別子を結合した検索）をする場合に，検索対象の複数のデータ識別子に同じ名称の列名があるとき，列指定によって指定した列がどのデータ識別子に対応する列なのかを明確にします。

指定形式を次に示します。

列指定の指定形式

```
列指定::=データ識別子.列名
```

データ識別子で列名を修飾する場合，指定するデータ識別子が有効な範囲内の列名を指定する必要があります。**データ識別子の有効範囲**は，FROM 句に指定したリレーション参照のリレーション，ストリーム，または相関名で定義されている列名です。定義されていない列名を指定した場合，エラーとなります。

問い合わせの結果として取得されるストリームおよびリレーションの名前には，問い合わせを定義しているクエリ名と同じ名前が付けられます。

列名は，次のどちらかに含まれる必要があります。

- 列名を指定する位置が有効範囲であるデータ識別子によって示されるストリームまたはリレーション
- 問い合わせ結果として取得されるストリームまたはリレーション

ストリームおよびリレーションの列名は，次の規則に従って問い合わせ結果として取得されます。

- UNION 句を持つリレーション式で取得されるリレーションの場合，1 番目に指定したリレーション式（UNION 句を除く）の結果として取得されるリレーションの列名が，問い合わせ後の結果として取得されるリレーションの列名になります。

次の例では，q2 のリレーション参照に指定したq1 から導出される列名はc1 だけです。d1 およびe1 は取得されません。

```
REGISTER QUERY q1
    SELECT s1.c1 FROM s1[NOW] …
    UNION
    SELECT s2.d1 FROM s2[NOW] …
    UNION
    SELECT s3.e1 FROM s3[NOW] …;

REGISTER QUERY q2
    SELECT q1.c1 FROM q1 …;
```

- UNION 句を持たないリレーション式で取得されるリレーションの場合，*i* 番目の選択式によって取得される列名が，*i* 番目に取得されるリレーションの列名になります。

  次の例では，q2 のリレーション参照に指定したq1 から取得される列名は，q1 の選択式に指定した順番に従い，列名c1，d1，e1 の順に取得されます。

```
REGISTER QUERY q1
    SELECT s1.c1, s2.d1, s3.e1 FROM s1[NOW], s2[NOW], s3[NOW];

REGISTER QUERY q2
    SELECT * FROM q1;
```

- ストリーム句で取得されるストリームの場合，ストリーム句で囲ったリレーション式の結果として取得されるリレーションの列名が，問い合わせ後の結果として取得されるストリームの列名になります。
- ストリーム間演算関数で導出されるストリームの場合，ストリーム間演算の結果として導出されるストリームの列名が，問い合わせ後の結果として導出されるストリームの列名になります。

ストリーム句については，「4.4.2　ストリーム句」を参照してください。リレーション式については，「4.4.3　リレーション式」を参照してください。また，ストリーム間演算関数については，「4.4.22　ストリーム間演算関数」を参照してください。

(3)　範囲変数

列指定の修飾子となる識別子を**範囲変数**といいます。範囲変数は，有効範囲と名称を持ちます。また，範囲変数で指定されたデータ識別子は，**表識別子**ともいいます。

相関名は範囲変数です。相関名が指定された場合，相関名で指定された元のデータ識別子は，列名の有効範囲ではなくなります。

リレーション参照，選択式に相関名を指定した場合の列名の有効範囲について，例を次に示します。

- リレーション参照に相関名を指定した場合

  （正しい記述例）

```
REGISTER QUERY q1 SELECT a1.a FROM s1[NOW] AS a1 WHERE a1.a > 1;
```

（誤った記述例）

```
REGISTER QUERY q2 SELECT s1.a FROM s1[NOW] AS a1 WHERE s1.a > 1;
```

s1 は相関名a1 として定義されているため，列名a の有効範囲ではなくなっています。このため，SELECT 句やWHERE 句の列指定にs1 は使用できません。

- 選択式に相関名を指定した場合

（正しい記述例）

```
REGISTER QUERY q1 SELECT s1.a AS a1 FROM s1[NOW];
REGISTER QUERY q2 SELECT a1 FROM q1 WHERE a1 > 1;
```

（誤った記述例）

```
REGISTER QUERY q1 SELECT s1.a AS a1 FROM s1[NOW];
REGISTER QUERY q3 SELECT q1.a FROM q1 WHERE q1.a > 1;
REGISTER QUERY q4 SELECT a FROM q1 WHERE a > 1;
```

s1.a は相関名a1 として定義されているため，列名a の有効範囲ではなくなっています。このため，SELECT 句やWHERE 句の列指定にq1.a やa は使用できません。

3.2.6　定数の指定

定数は，プログラム中で値を変更できないデータです。ここでは定数の指定について説明します。なお，定数で指定する値のデータ型については，「3.3　CQL のデータ型」を参照してください。

(1)　定数の種類と表記方法

定数には，数値を表す**数定数**と文字列を表す**文字列定数**があります。

数定数には，次の種類があります。

- 整数定数
- 浮動小数点定数
- 10 進定数

文字列定数には，次の種類があります。

- 文字列
- 日付データ
- 時刻データ
- 時刻印データ

なお，日付データおよび時刻データを指定する場合，指定値は，Hitachi Streaming Data Platform が動作している JavaVM のタイムゾーンに従った表現で指定する必要があります。

定数の表記方法を次の表に示します。

表 3-5　定数の表記方法

項番	定数		表記方法		ストリームデータ処理エンジンが解釈するデータ型
1	数定数	整数定数	INTEGER の場合 〔*符号*〕*整数* BIGINT の場合 〔*符号*〕*整数*〔L|l〕 (例) -123 45 6789L	整数は，*数字*の並びで表します。 符号は，+または-で表します。 文字列L またはl を末尾に付けた定数のデータ型はBIGINT 型になります。	INTEGER，BIGINT
		浮動小数点定数	小数点形式 〔*符号*〕*整数部*.*小数部* (例) -12.3 456.0 0.789	整数部と小数部を整数で表して，小数点(.)で区切ります。整数部，小数部，および小数点(.)は必ず付けてください。 符号は，+または-で表します。	DOUBLE
		10 進定数	小数点形式 〔*符号*〕*整数部*〔.*小数部*〕{D|d} (例) -12.3D 456.0d 0.789d -123D 45d	整数部と小数部を整数で表して，小数点(.)で区切ります。 符号は+，または-で表します。 整数部，末尾の文字D，またはd は必ず付けてください。 小数部および小数点(.)は，省略できます。	DECIMAL NUMERIC
2	文字列定数		'*文字列*' (例) 'HITACHI' '98' 'DB0002'	*文字列*は半角文字，および全角文字の列で表し，アポストロフィ（'）で囲みます※。 文字列にアポストロフィ（'）を含める場合は，1 個のアポストロフィ（'）を表すのにアポストロフィ（'）を続けて 2 個記述してください。 例えば，文字列としてアポストロフィ'を指定する場合は，「''''」となります。 日付データ，時刻データおよび時刻印データの指定形式の詳細について	VARCHAR

項番	定数	表記方法		ストリームデータ処理エンジンが解釈するデータ型
2	文字列定数	*'文字列'* (例) 'HITACHI' '98' 'DB0002'	は，「3.2.6　定数の指定」の「(2)　日付データの規定の文字列表現」「(3)　時刻データの規定の文字列表現」および「(4)　時刻印データの規定の文字列表現」を参照してください。	VARCHAR
注※ 文字の囲みに引用符（"）は使用できません。また，文字列中で引用符（"）は使用できません（文字列中の引用符（"）はエスケープできません）。				

(2)　日付データの規定の文字列表現

日付データは，次の形式で指定します。

```
'YYYY-MM-DD'
```

YYYY

0001～9999（年）

MM

01～12（月）

DD

01～*MM* で指定した月の最終日（日）

日付を既定の文字列表現の定数にする場合，年（*YYYY*），月（*MM*），日（*DD*）をハイフン（-）でつないで表現します。年（*YYYY*），月（*MM*），日（*DD*）の桁数に満たない場合は，足りない桁数分だけ左側に0 を補ってください。また，数字とハイフンの間には空白を挿入しないでください。

日付を規定の文字列表現の定数として指定した例を次に示します。

2010 年 3 月 25 日の場合の指定例

'2010-03-25'

日付データを指定する際は，次の点に注意してください。

- 形式に従わないで指定した文字列は，日付データとして認識されません。単なる文字列定数として認識されます。
- 年，月，日に，範囲外の値を指定した場合，クエリグループ登録時にメッセージ KFSP32014-E がメッセージログファイルに出力されます。

(3) 時刻データの規定の文字列表現

時刻データは，次の形式で指定します。

```
'hh:mm:ss'
```

hh

00～23（時）

mm

00～59（分）

ss

00～59（秒）※

注※

うるう秒の場合は00～61（秒）です。

時刻を既定の文字列表現の定数にする場合，時（*hh*），分（*mm*），秒（*ss*）をコロン（:）でつないで表現します。時（*hh*），分（*mm*），秒（*ss*）の桁数に満たない場合は，足りない桁数分だけ左側に0 を補ってください。また，数字とコロンの間には空白を挿入しないでください。

時刻を規定の文字列表現の定数として指定した例を次に示します。

午後 10 時 8 分 26 秒の場合の指定例

'22:08:26'

時刻データを指定する際は，次の点に注意してください。

- 形式に従わないで指定した文字列は，時刻データとして認識されません。単なる文字列定数として認識されます。
- 時，分，秒に，範囲外の値を指定した場合，次の例に示すように，値が繰り上げられて認識されます。

  範囲外の値を指定した場合の例

  表記：'25:08:26'

  認識される時刻：午前 1 時 8 分 26 秒

(4) 時刻印データの規定の文字列表現

時刻印データは，次の形式で指定します。

```
'YYYY-MM-DD▲hh:mm:ss〔.SSSSSSSSS〕'
```

YYYY

0001～9999（年）

MM

01～12（月）

DD

01～*MM* で指定した月の最終日（日）

hh

00～23（時）

mm

00～59（分）

ss

00～59（秒）※

注※

うるう秒の場合は00～61（秒）です。

SSSSSSSSS

0～9 桁の小数秒（*S*：0～9）

時刻印データを既定の文字列表現の定数にする場合，年（*YYYY*），月（*MM*），日（*DD*）をハイフンでつなぎ，半角空白を指定して，時（*hh*），分（*mm*），秒（*ss*）をコロン（：）でつないで表現します。その際，年（*YYYY*），月（*MM*），日（*DD*），時（*hh*），分（*mm*），秒（*ss*）の桁数に満たない場合，足りない数分だけ左側に0 を補ってください。数字とハイフン，数字とコロンの間に空白を入れてはいけません。ただし，日（*DD*）と時（*hh*）の間には，半角空白を必ず入れてください。

時刻印データを文字列表現の定数として指定した例を次に示します。

2008 年 8 月 1 日 午後 10 時 8 分 26 秒の場合の指定例

'2008-08-01 22:08:26'

時刻印データを指定する際は，次の点に注意してください。

- 形式に示した書式に従わないで指定した文字列は，時刻印データとして認識されません。単なる文字列定数として認識されます。
- 小数秒の精度を表現する場合，秒（*ss*）と小数秒（*SSSSSSSSS*）の間を，ピリオドでつないで指定してください。小数秒精度を省略した場合，小数秒精度が0 のデータとして扱われます。小数秒が 3 桁の場合は1 ミリ秒単位，6 桁の場合は1 マイクロ秒単位，9 桁の場合は1 ナノ秒単位の指定を意味します。
- 形式で示した範囲外の値を指定した場合，次の例に示すように，値が繰り上げられて認識されます。

  範囲外の値を指定した場合の例

  表記：'2008-02-30 25:08:26'

  認識される時刻：2008 年 3 月 2 日 01 時 08 分 26 秒

3.3 CQL のデータ型

ここでは，CQL のデータ型について説明します。

3.3.1 CQL のデータ型と Java のデータ型のマッピング

CQL のデータ型は，Java のデータ型にマッピングされます。

CQL のデータ型と Java のデータ型のマッピングについて，次の表に示します。

表 3-6 CQL のデータ型と Java のデータ型のマッピング

分類	CQL のデータ型	データ形式	Java のデータ型	データの範囲	備考
数データ	INT〔EGER〕	整数型4 バイト	Integer クラス	-2,147,483,648 ～ 2,147,483,647	－
	SMALLINT	整数型2 バイト	Short クラス	-32,768～ 32,767	－
	TINYINT	整数型1 バイト	Byte クラス	-128～127	－
	BIGINT	整数型8 バイト	Long クラス	-9,223,372,036 ,854,775,808～ 9,223,372,036, 854,775,807	－
	DEC〔IMAL〕〔('*m*')〕 ※1	10 進形式	java.math.BigDecimal クラス	$-10^{38}+1$～ $10^{38}-1$	精度（全体の桁数）が *m* 桁（'+'，'-'の符号は含みません）の 10 進数です。 *m* は正整数で，1≦*m*≦38 です。*m* を省略すると，15 が仮定されます。
	NUMERIC〔('*m*')〕 ※1				
	REAL	実数型4 バイト	Float クラス	• -3.402823466E+38 ～-1.175494351E-38 • 0 • 1.175494351E-38～ 3.402823466E+38	入力データに指数表現は使用できません。
	FLOAT	実数型8 バイト	Double クラス	• -1.7976931348623157E+308 ～-2.2250738585072014E-308	入力データに指数表現は使用できません。
	DOUBLE				

分類	CQL のデータ型	データ形式	Java のデータ型	データの範囲	備考
数データ	DOUBLE	実数型8 バイト	Double クラス	• 0 • 2.2250738585072014E-308～1.7976931348623157E+308	入力データに指数表現は使用できません。
文字データ	CHAR〔ACTER〕〔('*n*')〕	固定長文字列（文字数*n* 個）	java.lang.String クラス	1～255	*n* は正整数です。1≦n≦255 です。*n* を省略すると，1 が仮定されます。列には，列を埋めるために必要なだけ空白が追加されます。データの長さが指定文字数*n* を超えた場合は，エラーになります。
	VARCHAR'('*n*')'	可変長文字列（最大文字数*n* 個）		1～32,767	*n* は正整数です。1≦*n*≦32,767 です。列には，列を埋めるための空白は追加されません。データの長さが指定文字数*n* を超えた場合は，エラーになります。
日付データ	DATE※2	日付（年月日）	java.sql.Date クラス	*YYYYMMDD* • *YYYY*：0001～9999（年） • *MM*：01～12（月） • *DD*：01～該当年月の最終日（日）	－
時刻データ	TIME※2	時間（時分秒）	java.sql.Time クラス	*hhmmss* • *hh*：00～23（時） • *mm*：00～59（分） • *ss*：00～59（秒）	－
時刻印データ	TIMESTAMP〔('*p*')〕※2	日時（年月日+時分秒+ナノ秒）	java.sql.Timestamp クラス	*YYYYMMDDhhmmss*〔*nn…n*〕 • *YYYY*：0001～9999（年） • *MM*：01～12（月）	*p* は正整数です。*p* を省略すると，3 が仮定されます。 *p* の桁は，3 桁でミリ秒，6 桁でマイクロ秒，9 桁でナノ秒単位となります。*p* を超えた場合は，エラーになります。

分類	CQL のデータ型	データ形式	Java のデータ型	データの範囲	備考
時刻印データ	TIMESTAMP〔('*p*')〕※2	日時（年月日+時分秒+ナノ秒）	java.sql.Timestamp クラス	• *DD*：01～該当年月の最終日（日） • *hh*：00～23（時） • *mm*：00～59（分） • *ss*：00～59（秒） • *nn…n*：*p* 桁の小数秒（*n*：0～9）	*p* は正整数です。*p* を省略すると，3 が仮定されます。 *p* の桁は，3 桁でミリ秒，6 桁でマイクロ秒，9 桁でナノ秒単位となります。*p* を超えた場合は，エラーになります。

（凡例）

－：該当しません。

注※1

DECIMAL 型およびNUMERIC 型についての注意事項は，「3.3.3　DECIMAL 型および NUMERIC 型についての注意事項」を参照してください。

注※2

DATE 型，TIME 型およびTIMESTAMP 型のデータには範囲外の値を指定しないように注意してください。これらのデータ型に範囲外の値を指定した場合，例外などは発生しないで，意図しない処理が実行されるおそれがあります。

なお，「データ型(*n*)」のように，データ型の後ろに( )で囲んで記載する数値は，型の長さを示します。

メモ

CQL のデータ型と Java のデータ型とのマッピングは，SQL のデータ型と Java のデータ型とのマッピングを定義した JDBC API に従っています。

3.3.2　CQL のデータ型と C のデータ型のマッピング

CQL のデータ型は，C のデータ型にマッピングされます。

次の表に，CQL のデータ型と C のデータ型のマッピングを示します。

表 3-7　CQL のデータ型と C のデータ型のマッピング

クラス	CQL のデータ型	データ形式	C のデータ型	データの範囲	備考
数値データ	INT[EGER]	整数，4 バイト	(signed) int	-2,147,483,648～2,147,483,647	－

クラス	CQL のデータ型	データ形式	C のデータ型	データの範囲	備考
数値データ	SMALLINT	整数，2 バイト	(signed) short	-32,768～32,767	－
	TINYINT	整数，1 バイト	(signed) char	-128～127	－
	BIGINT	整数，8 バイト	(signed) long long	-9,223,372,036,854,775,807～9,223,372,036,854,775,807	long long 型を使用するには，64 ビットプロセッサが必要です。
	DEC[IMAL]['('*m*')']	10 進形式	－	－	－
	NUMERIC['('*m*')']		－	－	－
	REAL	実数，4 バイト	float	• -3.402823466E+38～-1.175494351E-38 • 0 • 1.175494351E-38～3.402823466E+38	指数表現を使用したデータの入力はできません。
	FLOAT	実数，8 バイト	double	• -1.797693134862315 7E+308～-2.2250738585072014E-308 • 0 • 2.2250738585072014E-308～1.7976931348623157E+308	指数表現を使用したデータの入力はできません。
	DOUBLE				
文字データ	CHAR[ACTER]['('*n*')']	固定長文字列（*n* 文字）	char *	ASCII コードがサポートされています。	1 文字は 1 バイトです。 *n* は正の整数で，1≦*n*≦255 です。*n* を省略すると，1 を指定したと見なされます。文字列を埋めるために必要な数のスペースが追加されます。データの長さが指定された文字数 *n* を超えると，エラーが発生します。
	VARCHAR'('*n*')'	可変長文字列（最大 *n* 文字）	HSDPVarCharType	ASCII コードがサポートされています。	*n* は正の整数で，1≦*n*≦32,767 です。実際の長さは 0 以上です。文字列を埋めるためのスペースは追加されません。データ

クラス	CQL のデータ型	データ形式	C のデータ型	データの範囲	備考
文字データ	VARCHAR('*n*')	可変長文字列（最大*n*文字）	HSDPVarCharType	ASCII コードがサポートされています。	の長さが指定された文字数*n* を超えると，エラーが発生します。
日付データ	DATE	日付（年月日）	－	－	－
時刻データ	TIME	時間（時分秒）	－	－	－
時刻印データ	TIMESTAMP['('*p*')'][※1]	日時（年月日＋時分秒＋ナノ秒）	HSDPTimestampType[※2]	*YYYYMMDDhhmmss*[*nn....n*] *YYYY*： 0001～9999（年） *MM*： 01～12（月） *DD*： 01～該当する月の最終日（日） *hh*： 00～23（時） *mm*： 00～59（分） *ss*： 00～59（秒） *nn....n*： *p* 桁の小数秒（*n*：0～9）	*p* は正の整数です。*n* は 0≦*n* ≦9 です。*p* を省略すると，3 を指定したと見なされます。 *p* が 3 桁の場合はミリ秒単位，6 桁の場合はマイクロ秒単位，9 桁の場合はナノ秒単位で時間が指定されます。指定したデータが*p* を超えると，エラーが発生します。

（凡例）

－：該当しません。

注※1

TIMESTAMP 型のデータの場合，範囲外の値を指定しないでください。このデータ型に範囲外の値を指定すると，エラーは発生しませんが，意図しない日に望ましくない処理が実行される場合があります。

括弧内の数字は，*データ型*（*n*）のように，型の長さを示します。

注※2

時刻印データの詳細については，「15.3.4　HSDPTimestampType 構造体」を参照してください。

サポートされていないデータ型

次のデータ型はサポートされていません。

- DEC[IMAL]['('*m*')']
- NUMERIC['('*m*')']
- DATE
- TIME

3.3.3 DECIMAL 型および NUMERIC 型についての注意事項

DECIMAL 型およびNUMERIC 型のデータで，送信データが指定した桁数を超えた場合，エラーになります。

また，次の処理を実行した場合に，結果の桁数がquery.decimalMaxPrecision パラメーターで指定した桁数を超えた場合，指定した桁数に丸められて表現されます（これらの処理を実行しない場合，値は丸められません）。

- 四則演算（2 項演算）の演算項，または集合関数SUM もしくはAVG の引数にDECIMAL 型（NUMERIC 型）が含まれる演算を実行した場合
- DECIMAL 型（NUMERIC 型）以外のデータ型からDECIMAL 型（NUMERIC 型）へのキャスト（暗黙的なキャストを含む）を実行した場合

値をどのように丸めて表現するかは，query.decimalRoundingMode パラメーターで指定します。パラメーターの詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

値の丸め方の例を次の表に示します。なお，この例はquery.decimalMaxPrecision パラメーターに「1」（桁数は 1 に丸める）と設定している場合です。

表 3-8　値の丸め方の例（query.decimalMaxPrecision=1 の場合）

演算結果（桁数 2）	query.decimalRoundingMode パラメーターの指定値	
	DOWN	HALF_UP
5.5	5	6
2.5	2	3
1.6	1	2
1.1	1	1
1.0	1	1
-1.0	-1	-1
-1.1	-1	-1
-1.6	-1	-2
-2.5	-2	-3
-5.5	-5	-6

なお，DECIMAL 型（NUMERIC 型）の数定数（10 進定数）の場合，桁数のチェックは実行されません。値がデータの範囲外の場合，query.decimalMaxPrecision パラメーターおよびquery.decimalRoundingMode パラメーターの指定値に関係なく，CQL に記述した値として表現されます。

DECIMAL 型（NUMERIC 型）の送信データ，演算結果，数定数について，桁数の指定方法と指定した桁数を超えた場合の動作を次の表に示します。

表 3-9　DECIMAL 型（NUMERIC 型）の桁数の指定方法と桁数を超えた場合の動作

種類		桁数の指定方法	指定した桁数を超えた場合の動作
送信データ		スキーマ指定文字列の型名	データ送信時に桁数を超えていた場合，エラーになります。
演算結果	• 四則演算 • 集合関数SUM／AVG • キャスト	query.decimalMaxPrecision パラメーター	query.decimalMaxPrecision パラメーターで指定した桁数になるよう，query.decimalRoundingMode パラメーターで指定した方法で丸められます。
	上記以外		演算結果がそのまま表示されます（値は丸められません）。
数定数（10 進定数）		指定なし	CQL に記述した値で表現されます。

3.4　データの比較

この節では，クエリ内でのデータの比較について説明します。

3.4.1　比較できるデータ型の組み合わせ

WHERE 句，またはHAVING 句に指定した探索条件の中で比較できるデータ型の組み合わせを次の表に示します。WHERE 句については，「4.4.6　WHERE 句」を，HAVING 句については，「4.4.8　HAVING 句」を参照してください。

なお，ここで説明する比較の規則は，GROUP BY 句，および集合関数にも適用されます。

表 3-10　比較できるデータ型の組み合わせ

項番	データ型	数データ	文字データ	日付データ	時刻データ	時刻印データ
1	数データ	○	△	×	×	×
2	文字データ	△	○	△	△	△
3	日付データ	×	△	○	×	×
4	時刻データ	×	△	×	○	×
5	時刻印データ	×	△	×	×	○

（凡例）
○：比較できます。
△：条件によっては比較できます。比較できる場合については，「3.4.2　データを比較する際の留意点」の「(2)　文字データの比較」を参照してください。
×：比較できません。

3.4.2　データを比較する際の留意点

ここでは，データを比較する際の留意点について説明します。

(1)　データ型一般の比較

GROUP BY 句を指定した検索では，列すべての結果の値が同一である行が，同じグループとして扱われます。

(2)　文字データの比較

文字データの比較は，Java のString 型データの比較に準じます。

文字データと日付，時刻，または時刻印データを比較する場合，日付，時刻，または時刻印データの形式に沿った文字列定数との比較だけが有効になります。

WHERE 句の数データと文字データの比較では，文字データが明示的に数データの型にキャストされ，かつ，文字データの書式が比較対象の数データの型に従っている場合にだけ比較できます。

文字データのキャストについて

比較に限らず，値式に現れた文字データについても同様にキャストができます。

キャストによって数データと比較できる文字データの書式について次の表に示します。

表 3-11　キャストによって数データと比較できる文字データの書式

数データの型	キャスト指定	文字データの書式	
		規則	例
整数型	(TINYINT) (SMALLINT) (INT[EGER]) (BIGINT)	• 0～9 の数字を指定します。 • ピリオドを含んではいけません。 • 値は数データの範囲内に収まる必要があります。 • 正の値の場合，プラス（+）の符号を指定できません。	正常に比較できる例 • '12' • '-1'（マイナスの符号を付ける） 比較がエラーになる例 • '12.0'（ピリオドを含む） • '1b3'（数字以外を含む） • '+1'（プラスの符号を付ける）
実数型および 10 進数	(REAL) (FLOAT) (DOUBLE) (DEC[IMAL])	• 0～9 の数字を指定します。 • ピリオドを含められます。 • 値は数データの範囲内に収まる必要があります。 • 値にプラス（+）またはマイナス（-）の符号を指定できます。	正常に比較できる例 • '12' • '12.0'（ピリオドを含む） • '-1.0'（マイナスの符号を付ける） • '+1.0'（プラスの符号を付ける） 比較がエラーになる例 • '1b3.0' （数字以外を含む）
注 正負の符号の付加は，java.lang.Number クラスの実装方式に従います。			

キャスト指定の形式については，「4.4.18　値式」を参照してください。

(3)　数データの比較

比較するデータのデータ型が異なる場合，範囲の広い方のデータ型に暗黙的にキャストして比較されます。範囲の広さを次に示します。

```
DECIMAL = NUMERIC > FLOAT = DOUBLE > REAL > BIGINT > INTEGER > SMALLINT > TINYINT
```

(4) 日付，時刻，および時刻印データの比較

日付，時刻，および時刻印データの比較は，Java のjava.sql.Date クラス，java.sql.Time クラス，およびjava.sql.Timestamp クラスの比較に準じます。

3.5　クエリ定義での注意事項および制限値

この節では，クエリを定義する際の注意事項と，クエリ定義での制限値について説明します。

3.5.1　クエリ定義での注意事項

- クエリ名およびストリーム名は，SDP サーバで一意になるように定義してください。登録済みの名前を定義した場合は，クエリグループの登録時にエラーになります。
- あるクエリ定義ファイルで定義されたストリームまたはクエリについて，別のクエリ定義ファイルで定義されたクエリから参照することはできません。

3.5.2　クエリ定義での制限値

クエリ定義での制限値について次の表に示します。

表 3-12　クエリ定義での制限値

項番	項目		値の範囲
1	1 入力ファイルに記述できる CQL コマンド数		1～1,024
2	1CQL コマンドの長さ		1～300,000 文字
3	1 ストリーム中の列数		1～3,000
4	ストリームデータ 1 件の長さ（1 行のサイズ）		1～2,147,483,647 バイト
5	名前の長さ（ストリーム名，リレーション名，クエリ名，列名，相関名）		1～100 文字
6	検索項目数（SELECT 句の選択式の個数）		1～3,000
7	列指定リスト		1～255
8	探索条件内の比較述語および括弧の個数		1～255
9	1CQL クエリ中に指定できるストリームとリレーションの合計数		1～64
10	ROWS ウィンドウの生存する行数		1～100,000
11	時間指定	SECOND	1～2,678,400
12		MILLISECOND	1～86,400,000
13		MINUTE	1～44,640
14		HOUR	1～744
15		DAY	1～31

4 CQL リファレンス

この章では，CQL の文法について説明します。

4.1　CQL 文法説明で使用する記述形式

ここでは，CQL 文法説明で使用する記述形式について説明します。

各 CQL で説明する項目は次のとおりです。ただし，CQL によっては説明しない項目もあります。なお，CQL の文法で使用する記号については，「3.1.4　CQL 文法説明で使用する記号」を参照してください。

形式
　記述形式を説明しています。

説明
　機能を説明しています。

オペランド
　オペランドごとに，指定する項目や，どのような場合に指定するかを説明しています。

引数
　引数ごとに，指定する項目や，どのような場合に指定するかを説明しています。

構文規則
　構文規則を説明しています。

戻り値
　関数の戻り値を説明しています。

注意事項
　注意事項を説明しています。

使用例
　使用例を説明しています。

4.2 CQLの一覧

CQLには，ストリームおよびクエリを定義するために使用する定義系CQLと，定義系CQLのREGISTER QUERY句やREGISTER QUERY_ATTRIBUTE句中に指定する操作系CQLがあります。

定義系CQLの一覧を次の表に示します。

表4-1 定義系CQLの一覧

項番	CQL	説明
1	REGISTER STREAM句	ストリームを定義します。
2	REGISTER QUERY句	クエリを定義します。
3	REGISTER QUERY_ATTRIBUTE句	クエリに時刻解像度を指定します。時刻解像度指定の対象となるクエリは，この句のあとに定義する必要があります。

操作系CQLの一覧を次の表に示します。

表4-2 操作系CQLの一覧

項番	種類	説明
1	問い合わせ	リレーションまたはREGISTER STREAM句で定義したストリームに対して，データの検索を実行します。
2	ストリーム句	出力されるデータをストリームに変換します。
3	リレーション式	1つ以上のリレーションからのデータ検索や検索結果のフィルタリングなどを実行します。
4	SELECT句	検索した結果を出力する項目（選択式）を指定します。検索結果はリレーションとして取得します。
5	FROM句	1つ以上のリレーション（リレーション参照）を指定します。FROM句で取得されるリレーションが，WHERE句またはHAVING句の対象となります。
6	WHERE句	FROM句によって取得されるリレーションに対する探索条件を指定します。
7	GROUP BY句	直前に指定する句によって取得されるリレーション中の列（グループ化列）を指定します。指定したグループ化列でグルーピングが実行されます。
8	HAVING句	FROM句，WHERE句，またはGROUP BY句によって取得されるリレーションに対して，探索条件を指定します。HAVING句の探索条件では，HAVING句中に指定した探索条件によって論理演算を実行し，真の結果だけをリレーションとして取得します。
9	UNION句	複数のSELECT句を結合して，1つのCQL文として実行します。
10	選択リスト	1つまたは複数の選択式を指定します。

項番	種類	説明
11	選択式	検索結果として出力する項目を指定します。
12	列指定リスト	1 つまたは複数の列を指定します。
13	リレーション参照	検索するリレーションを指定します。リレーション参照は，FROM 句で指定します。
14	ウィンドウ指定	ストリームの生存期間を指定します。ウィンドウ指定は，リレーション参照で指定します。
15	時間指定	時間の単位を指定します。
16	探索条件	指定した条件に従って論理演算を実行し，真の結果だけをリレーションとして取得します。 探索条件は，WHERE 句およびHAVING 句で指定します。
17	比較述語	真または偽の論理値を与えるための条件を指定します。
18	値式	値を指定します。
19	定数	定数を指定します。
20	集合関数	複数行から算出される値を求めます。
21	スカラ関数	単一行から算出される値を求めます。
22	ストリーム間演算関数	ストリームデータ間の演算を実行します。
23	外部定義集合関数	複数行から算出される値を求めます。 この関数は，ユーザーが独自に実装できます。
24	外部定義スカラ関数	単一行から算出される値を求めます。 この関数は，ユーザーが独自に実装できます。
25	外部定義ストリーム間演算関数	ストリームデータ間の演算を実行します。 この関数は，ユーザーが独自に実装できます。
26	組み込み集合関数	複数行から算出される値を求めます。 この関数は，Hitachi Streaming Data Platform が CQL の構文として標準提供しています。
27	組み込みスカラ関数	単一行から算出される値を求めます。 この関数は，Hitachi Streaming Data Platform が CQL の構文として標準提供しています。

4.3　定義系 CQL

ここでは定義系 CQL について説明します。

4.3.1　REGISTER STREAM 句（ストリームの定義）

形式

```
REGISTER STREAM::=REGISTER▲STREAM▲ストリーム名
                                  ▲スキーマ指定文字列
    スキーマ指定文字列::='('列名▲型名〔,列名▲型名〕…')'
```

説明

ストリームを定義します。

ストリームは，ストリームの管理テーブルなどを格納する領域（システムカタログ）に登録され，同時にストリームの受付が開始されます。

オペランド

ストリーム名

ストリーム（**入力ストリーム**）の名称を，任意の名称で，ストリームデータ処理システム内で一意になるように指定します。名称の指定については，「3.2.4　名前の指定」を参照してください。

スキーマ指定文字列

ストリーム名で定義されるストリームに含まれる列名と，その型名を指定します。

列名

列名を任意の名称で，ストリームデータ処理システム内で一意になるように指定します。名称の指定については，「3.2.4　名前の指定」を参照してください。

列名に指定できる個数は，1～3,000 個です。

型名

型名については，「3.3　CQL のデータ型」を参照してください。

構文規則

1 つの定義で，スキーマ指定文字列をすべて指定してください。例えば，s1 という名称でストリームを定義する場合は，次の例 1 で示すように，必要なスキーマ指定文字列をすべて指定します。

例 1（正しい指定例）

```
REGISTER STREAM s1(id INT, name VARCHAR(10));
```

次の例 2 で示すように，s1 という名称でストリームを定義する場合に，スキーマ指定文字列を分けて指定すると，同一名称のストリームの二重指定となるためエラーになります。

例 2（誤った指定例）

```
REGISTER STREAM s1 (id INT);
REGISTER STREAM s1 (name VARCHAR(10));
```

注意事項

ありません。

使用例

スキーマとしてINT 型の列id，および 10 個の文字データから構成される列name を持つストリームs1 をシステムカタログに登録します。

```
REGISTER STREAM s1(id INT, name VARCHAR(10));
```

4.3.2　REGISTER QUERY 句（クエリの定義）

形式

```
REGISTER QUERY::=REGISTER▲QUERY▲クエリ名▲問い合わせ
```

説明

クエリを定義します。

クエリは，クエリの管理テーブルなどを格納する領域（クエリリポジトリ）に登録され，同時にクエリ処理が開始されます。

オペランド

クエリ名

クエリの名称を任意の名称で，ストリームデータ処理システム内で一意になるように指定します。名称の指定については，「3.2.4　名前の指定」を参照してください。

ここで指定した名称は，問い合わせによって取得されるストリーム（**出力ストリーム**），またはリレーションの名称となります。

問い合わせ

問い合わせを定義します。定義内容については，「4.4.1　問い合わせ」を参照してください。

構文規則

1 つの定義で，1 つのクエリに対する問い合わせをすべて指定してください。例えば，q1 としてクエリを定義する場合は，次の例 1 で示すように，必要な問い合わせをすべて指定します。

例 1（正しい指定例）

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 10];
```

次の例 2 で示すように，q1 として定義したクエリに対して，問い合わせを分けて指定すると，同一名称のクエリの二重指定となるためエラーになります。

例 2（誤った指定例）

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 10];
REGISTER QUERY q1 SELECT s1.b FROM s1[ROWS 10];
```

注意事項

ありません。

使用例

ストリームs1 の列a のデータを出力するクエリq1 を定義します。

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 10];
```

4.3.3　REGISTER QUERY_ATTRIBUTE 句（時刻解像度の指定）

形式

```
REGISTER QUERY_ATTRIBUTE::=REGISTER▲QUERY_ATTRIBUTE▲クエリ名
                           ▲STREAM_NAME=データ識別子
                           ▲PERIOD=メッシュ間隔
                           ▲TARGETS=集合関数1'('列名1')',
                           …,集合関数N'('列名N')'
```

説明

クエリに時刻解像度を指定します。

オペランド

クエリ名

この句のあとに定義する時刻解像度指定の対象となるクエリ名を，ストリームデータ処理システム内で一意になるように指定します。名称の指定については，「3.2.4　名前の指定」を参照してください。

クエリ名は，ほかのREGISTER QUERY_ATTRIBUTE 句で指定する名称と重複してはいけません。また，時刻解像度指定の対象となるクエリを定義するREGISTER QUERY 句では，必ずRANGE ウィンドウを指定してください。

データ識別子

時刻解像度を指定する列名が定義されている，ストリーム名またはクエリ名を指定します。指定するストリーム名またはクエリ名は，REGISTER QUERY_ATTRIBUTE 句よりも前に定義されている必要があります。

メッシュ間隔

メッシュ間隔をミリ秒単位で任意に指定します。必ず，単位としてms を記述してください。

100 ミリ秒未満の値は設定できません。また，時刻解像度を指定するクエリ名の句に指定されるRANGE ウィンドウの間隔がメッシュ間隔を超える場合は，メッシュ間隔はRANGE ウィンドウの間隔に設定されます。

例えば，メッシュ間隔が1,000 ミリ秒で，RANGE ウィンドウの間隔が500 ミリ秒の場合，メッシュ間隔は500 ミリ秒に設定されます。また，メッシュ間隔がRANGE ウィンドウより小さく，かつ86,400,000 ミリ秒より大きい場合は，86,400,000 ミリ秒に設定されます。

なお，100 ミリ秒未満の値を指定した場合は，100 ミリ秒に設定されます。

集合関数

集合関数を指定します。ここで指定できる集合関数を次の表に示します。

表 4-3　指定できる集合関数

項番	集合関数	意味
1	SUM	合計値の計算に使用します。
2	MAX	最大値の計算に使用します。
3	MIN	最小値の計算に使用します。

指定できる集合関数の個数は，1～256 個です。

列名

STREAM_NAME で指定したデータ識別子が持つ列名を指定します。

列名は，重複してはいけません。

指定できる列名の個数は，1～256 個です。

構文規則

ありません。

注意事項

- REGISTER QUERY_ATTRIBUTE 句では，クエリ名，データ識別子，列名がそれぞれ重複してはいけません。
- STREAM_NAME で指定するストリーム名またはクエリ名は，REGISTER QUERY_ATTRIBUTE 句より前に定義しておくか，またはあらかじめ登録しておいてください。

- TARGETS で指定する集合関数の引数に指定する列名は，データ識別子の列名に含まれている必要があります。
- クエリ名，データ識別子，メッシュ間隔，集合関数，および列名に，特殊文字（セミコロンやピリオドなど）を含めてはいけません。
- クエリ名，データ識別子，および列名として，「REGISTER」，「QUERY_ATTRIBUTE」，「STREAM_NAME」，「PERIOD」，「TARGETS」を指定してはいけません。
- 時刻解像度を指定するクエリの定義（REGISTER QUERY 句）は，次の条件を満たす必要があります。
  - FROM 句に複数のストリームを記述しない。
  - WHERE 句を指定しない。
  - GROUP BY 句を指定する。
  - REGISTER QUERY_ATTRIBUTE 句のあとに定義する。
- REGISTER QUERY_ATTRIBUTE 句でTARGETS 以下に指定する列名とSTREAM_NAME に指定するデータ識別子の持つ列名は，1 対 1 に対応させておく必要があります。また，時刻解像度を指定するクエリの列名とSTREAM_NAME に指定するデータ識別子の持つ列名も，1 対 1 に対応させる必要があります。

  クエリq1 に対して，時刻解像度を指定する場合を例に説明します。下線は，説明で示している個所です。

  （正しい指定例）

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
ISTREAM(
SELECT name, price AS price0, price AS price1
FROM stock[NOW]
);
REGISTER QUERY_ATTRIBUTE q1
STREAM_NAME=q0 PERIOD=300ms TARGETS=SUM(price0),MAX(price1);
REGISTER QUERY q1
ISTREAM(
SELECT q0.name, SUM(q0.price0) AS sum_price, MAX(q0.price1) AS max_price
FROM q0[RANGE 1 MINUTE]
GROUP BY q0.name
);
```

  「REGISTER QUERY q0」で指定した列名（price0, price1），「REGISTER QUERY_ATTRIBUTE q1」で指定した列名（price0, price1），「REGISTER QUERY q1」で指定した列名（price0, price1）が，1 対 1 に対応しています。時刻解像度を指定する場合は，この例で示すように列名を対応させる必要があります。

  （誤った指定例）

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
ISTREAM(
SELECT name, price AS price0, price AS price1, price AS price2, price AS price3
FROM stock[NOW]
);
REGISTER QUERY_ATTRIBUTE q1
STREAM_NAME=q0 PERIOD=300ms TARGETS=SUM(price0),MAX(price1),MIN(price2);
REGISTER QUERY q1
ISTREAM(
```

```
SELECT q0.name, SUM(q0.price0) AS sum_price, MAX(q0.price1) AS max_price
FROM q0[RANGE 1 MINUTE]
GROUP BY q0.name
);
```

「REGISTER QUERY q0」で指定した列名（price0，price1，price2，price3），「REGISTER QUERY_ATTRIBUTE q1」で指定した列名（price0，price1，price2），「REGISTER QUERY q1」で指定した列名（price0，price1）が，1 対 1 に対応していないため，エラーとなります。

- 時刻解像度を指定したクエリでは，タプルをメッシュ間隔内で擬似タプルとして集約するため，集約結果の出力頻度が異なります。例えば，SUM を指定して集計する場合，集約した擬似タプルごとに結果が生成されるために，入力タプルごとに時々刻々と生成される結果の出力頻度とは異なります。

使用例

下線は，この例のポイントとなる個所です。

（例 1）

RANGE ウィンドウ内にあるタプルで，あらかじめ定義されているストリーム名stock の，列名price の合計値を求めます。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY_ATTRIBUTE q1 STREAM NAME=stock PERIOD=300ms TARGETS=SUM(price);
REGISTER QUERY q1
ISTREAM(
SELECT name, SUM (price) AS s1
FROM stock[RANGE 1 MINUTE]
GROUP BY name
);
```

REGISTER QUERY_ATTRIBUTE 句で指定するクエリには，時刻解像度を指定するクエリq1 を指定します。また，REGISTER STREAM 句で定義したストリーム名stock の列名price の合計を求めるため，データ識別子にstock を指定し，TARGETS にSUM(price)を指定します。なお，price の合計値ではなく，最大値や最小値を求める場合には，MAX，MIN を指定します。

（例 2）

RANGE ウィンドウ内にあるタプルで，あらかじめ定義されているストリーム名stock の，列名price の合計値，最大値，最小値を求めます。

```
REGISTER STREAM stock(price INTEGER, name VARCHAR(10));
REGISTER QUERY q0
ISTREAM(
SELECT name, price AS price0, price AS price1, price AS price2
FROM stock[NOW]
);
REGISTER QUERY_ATTRIBUTE q1 STREAM NAME=q0 PERIOD=300ms
TARGETS=SUM(price0),MAX(price1),MIN(price2);
REGISTER QUERY q1
ISTREAM(
SELECT q0.name, SUM(q0.price0) AS sum_price,
MAX(q0.price1) AS max_price, MIN(q0.price2) AS min_price
FROM q0[RANGE 1 MINUTE]
```

```
GROUP BY q0.name
);
```

データ識別子stock，列名price のクエリについて，複数の集合演算を実行する場合に，TARGETS に同一の列名price は指定できません。例えば，「TARGETS=SUM(price),MAX(price),MIN(price)」のようには指定できません。

この場合，列名price を合計値用，最大値用，最小値用にそれぞれ定義する必要があります。この例では，CQL 文によって列名price を置き換えた列名price0，price1，price2 を持つデータ識別子（ここではq0）を定義しています。q0 で定義した列名を使用してq1 でそれぞれ演算を実行することで，複数の文の集合関数を実行できます。

4.4　操作系 CQL

ここでは操作系 CQL について説明します。

4.4.1　問い合わせ

形式

```
問い合わせ::={リレーション式
              |｛ストリーム句’(’リレーション式’)’
              |ストリーム間演算関数｝}
```

説明

リレーションまたはREGISTER STREAM 句で定義したストリームに対して，データの検索を実行します。

オペランド

リレーション式

リレーション式の指定については，「4.4.3　リレーション式」を参照してください。

ストリーム句を指定しないでリレーション式を指定すると，問い合わせ結果はリレーションで取得されます。

ストリーム句

ストリーム句を指定する場合，ストリーム句に続けてリレーション式を括弧（( )）で囲んで記述します。ストリーム句の指定については，「4.4.2　ストリーム句」を参照してください。

ストリーム句を指定すると，問い合わせ結果はストリームで取得されます。

ストリーム間演算関数

ストリーム間演算関数の指定については，「4.4.22　ストリーム間演算関数」を参照してください。

ストリーム間演算関数を指定すると，問い合わせ結果はストリームで取得されます。

構文規則

ありません。

注意事項

ありません。

使用例

リレーション式を指定し，ストリームs1 の列a のデータをリレーションとして生成します。生成されたリレーションはリレーションq1 として出力されます。下線部が問い合わせの部分です。

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 10];
```

4.4.2　ストリーム句

形式

```
ストリーム句::= {ISTREAM | DSTREAM
                | RSTREAM〔'['整数定数〔▲時間指定〕']'〕}
```

説明

出力されるデータをストリームに変換します。

オペランド

ISTREAM

出力リレーション内の増加分だけを出力します。

DSTREAM

出力リレーション内の減少分だけを出力します。

RSTREAM

出力リレーション内の集合すべてを一定間隔で出力します。

整数定数

時間の値そのものを指定します。整数定数の値の範囲については，「4.4.15　時間指定」を参照してください。

指定を省略した場合，指定値には1 秒を仮定します。

時間指定

整数定数で指定した時間の単位を指定します。時間指定の指定については，「4.4.15　時間指定」を参照してください。

指定を省略した場合，整数定数で指定した値を秒単位として認識します。

構文規則

問い合わせに定義するストリーム句の詳細を記述します。ストリーム句のあとにリレーション式を括弧（(　)）で囲んで指定します。

注意事項

ありません。

使用例

リレーションs1 の全データの増加時刻をタイムスタンプに持つストリームを生成します。生成されたストリームは，ストリームq1 として出力されます。下線部がストリーム句の部分です。

```
REGISTER QUERY q1 ISTREAM (SELECT * FROM s1[ROWS 100]);
```

4.4.3　リレーション式

形式

```
リレーション式::=SELECT句▲FROM句〔▲WHERE句〕
                 〔▲GROUP BY句〔▲HAVING句〕〕〔▲UNION句〕
```

説明

1 つ以上のリレーションからのデータ検索や検索結果のフィルタリングなどを実行します。

オペランド

SELECT 句

SELECT 句の指定については，「4.4.4　SELECT 句」を参照してください。

FROM 句

FROM 句の指定については，「4.4.5　FROM 句」を参照してください。

WHERE 句

WHERE 句の指定については，「4.4.6　WHERE 句」を参照してください。

GROUP BY 句

GROUP BY 句の指定については，「4.4.7　GROUP BY 句」を参照してください。

HAVING 句

HAVING 句の指定については，「4.4.8　HAVING 句」を参照してください。

UNION 句

UNION 句の指定については，「4.4.9　UNION 句」を参照してください。

構文規則

問い合わせのリレーション式の詳細を記述します。SELECT 句，FROM 句以降に検索方法やフィルタリング方法を指定します。

注意事項

ありません。

使用例

リレーションs1 の列b の値が20 より小さい列a のデータを出力します。下線部がリレーション式の部分です。

```
REGISTER QUERY q1 SELECT s1.a FROM s1[ROWS 100] WHERE s1.b < 20;
```

4.4.4 SELECT 句

形式

```
SELECT句::=SELECT▲ {選択リスト | *}
```

説明

リレーションに対して検索した結果を出力する項目（選択式）を指定し，選択式をリレーションとして取得します。

オペランド

選択リスト

選択リストの指定については，「4.4.10　選択リスト」を参照してください。

*

リレーションのすべての列を出力します。

リレーション式のFROM 句で指定したリレーションの順序に従い，すべての列を指定することを意味します。各リレーション中の列の順序は，ストリームを定義したときのスキーマ指定文字列，またはクエリを定義したときの選択リストに指定した順序です。

構文規則

ありません。

注意事項

ありません。

使用例

リレーションs1 の全データを出力します。下線部がSELECT 句の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[ROWS 10];
```

4.4.5 FROM 句

形式

```
FROM句::=FROM▲リレーション参照〔,リレーション参照〕…
```

説明

1 つ以上のリレーション（リレーション参照）を指定します。FROM 句で取得されるリレーションが，WHERE 句またはHAVING 句の対象となります。

WHERE 句およびHAVING 句を指定しない場合は，FROM 句で取得されるリレーションがSELECT 句の対象となります。

複数のリレーションを指定した場合，指定したリレーションの順序で各リレーションから 1 行ずつ組み合わせて連結された行が，FROM 句で取得されるリレーションの行になります。行数は，各リレーションの行数の積になります。

オペランド

リレーション参照

リレーション参照の指定については，「4.4.13　リレーション参照」を参照してください。

指定できるリレーション参照の個数は，1～64 個です。

構文規則

ありません。

注意事項

FROM 句で取得されるリレーションは，グループ化列を持たない 1 つのグループになります。

使用例

取得するリレーションs1 を指定します。下線部がFROM 句の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[ROWS 100];
```

4.4.6 WHERE 句

形式

```
WHERE句::＝WHERE▲探索条件
```

説明

FROM 句によって取得されるリレーションに対する探索条件を指定します。

WHERE 句を指定し，かつHAVING 句を指定しない場合は，WHERE 句によって取得されるリレーションがSELECT 句の対象となります。

WHERE 句を省略すると，FROM 句で取得されるリレーションに含まれるすべての行がSELECT 句の対象となります。

オペランド

探索条件

探索条件の指定については，「4.4.16　探索条件」を参照してください。

先行するFROM 句のリレーション参照が 1 つの場合，探索条件には列名だけを指定します。FROM 句のリレーション参照が複数の場合，探索条件に指定する各列名は，データ識別子と列名を指定して，一意にしなければいけません。

構文規則

先行するFROM 句で取得したリレーションをt とした場合，探索条件はt のそれぞれの行に適用されます。WHERE 句で取得されるリレーションは，探索条件の結果が真となる，t の行の集合で構成されるリレーションです。

注意事項

WHERE 句で取得されるリレーションは，グループ化列を持たない 1 つのグループになります。

使用例

リレーションs1 の列b の値が10 より大きい列a，b のデータを出力します。下線部がWHERE 句の部分です。

```
REGISTER QUERY q1 SELECT s1.a,s1.b FROM s1[ROWS 100] WHERE s1.b > 10;
```

4.4.7　GROUP BY 句

形式

```
GROUP BY句::=GROUP▲BY▲列指定リスト
```

説明

直前に指定する句によって取得されるリレーション中の列（グループ化列）を指定します。指定したグループ化列でグルーピングが実行されます。

グルーピングでは，GROUP BY 句で指定した列の値が同一の行をグルーピングして，グループ単位に 1 行にまとめて出力します。

オペランド

列指定リスト

列指定リストの指定については，「4.4.12　列指定リスト」を参照してください。

列指定リストに指定する列指定は，同じ列指定を重複して指定できません。また，先行するFROM 句のリレーション参照が 1 つの場合，列指定リストには列名だけを指定します。FROM 句のリレーション参照が複数の場合，列指定リストに指定する各列名は，データ識別子と列名を指定して，一意にしなければいけません。

構文規則

ありません。

注意事項

リレーション式にGROUP BY 句を指定した場合，SELECT 句中の選択式にはGROUP BY 句で指定された要素（グループ化列の名称）以外の列指定または列名を一次子とする値式は指定できません。

使用例

リレーションs1 の列b，c をグループ化し，列a，b，c のデータを出力します。下線部がGROUP BY 句の部分です。

```
REGISTER QUERY q1 SELECT s1.b,s1.c
                  FROM s1[ROWS 100] GROUP BY s1.b,s1.c;
```

4.4.8 HAVING 句

形式

```
HAVING句::=HAVING▲探索条件
```

説明

FROM 句，WHERE 句，またはGROUP BY 句によって取得されるリレーションに対して，探索条件を指定します。HAVING 句の探索条件では，HAVING 句中に指定した探索条件によって論理演算を実行し，真の結果だけをリレーションとして取得します。

HAVING 句を指定する場合は，HAVING 句によって取得されるリレーションがSELECT 句の対象となります。

オペランド

探索条件

探索条件の指定については，「4.4.16　探索条件」を参照してください。

探索条件中に列指定を指定する場合は，グループ化列を指定するか，または集合関数の引数として指定してください。グループ化列については，「4.4.7　GROUP BY 句」，集合関数については「4.4.20 集合関数」を参照してください。

構文規則

HAVING 句中に指定した探索条件の結果が真となるグループが選択されます。

注意事項

HAVING 句を省略すると，先行するGROUP BY 句，またはFROM 句の結果の全グループが選択されます。

使用例

リレーションs1 の列b，c をグループ化し，列a の平均値が10 より大きい列a，b，c のデータを出力します。下線部がHAVING 句の部分です。

```
REGISTER QUERY q1 SELECT AVG(s1.a) AS a1,s1.b,s1.c FROM s1[ROWS 100]
                  GROUP BY s1.b,s1.c HAVING AVG(s1.a) > 10;
```

4.4.9 UNION 句

形式

```
UNION句::=UNION▲〔ALL▲〕リレーション式
```

説明

複数のSELECT 句を結合して，1 つの CQL 文として実行します。

オペランド

ALL

指定した場合，行の重複を許可します。

ALL を指定しないでUNION を指定した場合，行を重複排除します。

重複排除とは，検索結果に重複した行（選択式で指定した項目で 1 つ以上の行を構成し，その行が同一のもの）が存在する場合，重複した行を排除し，1 つの行と見なして出力することをいいます。

リレーション式

リレーション式の指定については，「4.4.3　リレーション式」を参照してください。

構文規則

次の 2 つの条件を満たすクエリは，構文エラーになります。

- UNION ALL オペランドを使用している
- 複数のSELECT 句で，FROM 句に同一ストリームまたはリレーションを 1 つだけ指定している

構文エラーとなる例を次に示します。下線部がエラーになる個所です。

```
REGISTER QUERY q1 ISTREAM(
    SELECT * FROM s1[RANGE 3 SECOND]
    UNION ALL SELECT * FROM s1[RANGE 5 SECOND]);
```

注意事項

- UNION 句で結合する場合，すべてのSELECT 句の選択リストは，数，型，および文字データの文字数をすべて同じにしてください。
- UNION 句を持つリレーション式で取得されるリレーションの場合，1 番目に指定したリレーション式（UNION 句を除く）の結果として取得されるリレーションの列名が，問い合わせ後の結果として取得されるリレーションの列名になります。

  詳細については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。
- FROM 句で指定するストリームデータが同一の場合，入力したすべての列をそのまま出力するSELECT 句をUNION ALL で結合することはできません。

  誤った例を次に示します。下線部が該当するSELECT 句です。

  ```
  SELECT * FROM s1[RANGE 3 SECOND] UNION ALL SELECT * FROM s1[RANGE 5 SECOND];
  ```

  どちらのSELECT 句も同一のストリームs1 を 1 つだけ入力し，オペランドに*を指定しているので，入力したすべての列をそのまま出力し，エラーとなります。

使用例

リレーションs1 とリレーションs2 の全データを出力します。下線部がUNION 句の部分です。

```
REGISTER QUERY s1 SELECT * FROM s1[ROWS 100] UNION SELECT * FROM s2[ROWS 100];
```

4.4.10　選択リスト

形式

```
選択リスト::=選択式〔,選択リスト〕
```

説明

1 つまたは複数の選択式を指定します。

オペランド

選択式

選択式の指定については，「4.4.11　選択式」を参照してください。

指定できる選択式の個数は，1～3,000 個です。

構文規則

SELECT 句のあとに指定します。

注意事項

選択リストに指定できるのは，FROM 句によって取得されたリレーションのデータだけです。

使用例

リレーションs1 の列a およびb のデータを出力します。下線部が選択リストの部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100];
```

4.4.11　選択式

形式

```
選択式::= { {列指定 | 列名} 〔▲AS▲相関名〕
          | {値式 | 集合関数} ▲AS▲相関名}
```

説明

検索結果として出力する項目を指定します。

オペランド

列指定

列指定については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。

列名

後続のFROM 句のリレーション参照が 1 つである場合，列名を指定できます。

後続のFROM 句のリレーション参照が複数ある場合は，列指定を指定してください。

相関名

次のような場合に区別できる名称を指定します。

- 列名が重複する場合
- 列指定の名称が長く複雑である場合
- 値式が列名と列指定のどちらでもない場合

名称は名前の規則に従います。名前の規則の詳細については，「3.2.4　名前の指定」を参照してください。

相関名には，1 つのSELECT 句中に指定したほかの相関名と同じ名称は指定できません。また，ほかの範囲変数の表識別子と同じ名称も指定できません。範囲変数と表識別子については，「3.2.5　名前の修飾」を参照してください。

値式

値式については，「4.4.18　値式」を参照してください。

i 番目の選択式に対して列名を指定した場合，リレーション式によって取得されるリレーションの *i* 番目の列名は，指定した列名となります。*i* 番目の選択式に対して列名を指定しない場合は，次のようになります。

- 列指定から取得された値式のとき，その列指定で指定した列名が，リレーション式によって取得されるリレーションの *i* 番目の列名になります。
- 列指定以外から取得された値式のとき，相関名の指定が必要になります。

集合関数

集合関数の指定については，「4.4.20　集合関数」を参照してください。

集合関数の引数中には，列指定または列名を含める必要があります。

構文規則

- 選択式に指定した相関名は取得されるリレーションだけで利用できます。選択式の後続の句では利用できません。

  例えば，次のように，q1 から取得された相関名a1 およびa2 は，q2 で利用できます。

（正しい指定例）

```
REGISTER QUERY q1 SELECT s1.c1 AS a1, s2.c1 AS a2 FROM s1[NOW], s2[NOW];
REGISTER QUERY q2 SELECT a1, a2 FROM q1 WHERE a1 > 1 GROUP BY a1, a2
    HAVING a2 > 1;
```

次のように，選択式に指定した相関名a1 およびa2 は，後続の句で利用できません。

（誤った指定例）

```
REGISTER QUERY q1 SELECT s1.c1 AS a1, s2.c1 AS a2 FROM s1[NOW], s2[NOW]
    WHERE a1 > 1 GROUP BY a1, a2 HAVING a2 > 1;
```

- 選択式に相関名を指定する場合は，ほかの選択式で使用した列名と異なる名称を指定します。

（正しい指定例）

```
REGISTER QUERY q1 SELECT s1.c1, s1.c2+1 AS a1 ～ ;
```

次の例では，ほかの選択式に使用しているc1 を使用しているため，エラーになります。

（誤った指定例）

```
REGISTER QUERY q1 SELECT s1.c1, s1.c2+1 AS c1 ～ ;
```

注意事項

選択式に指定できるのは，定義済みのストリームのデータだけです。

使用例

リレーションs1 の列a およびb のデータを出力します。下線部が選択式の部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100];
```

4.4.12　列指定リスト

形式

```
列指定リスト::=列指定〔,列指定リスト〕
```

説明

1 つ，または複数の列を指定します。

オペランド

列指定

列指定については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。

指定できる列指定の個数は，1～255 個です。

構文規則

列指定リストの構文規則については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。

注意事項

列指定リストの注意事項については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。

使用例

リレーションs1 の列b およびc をグループ化し，列a，b，およびc のデータを出力します。下線部が列指定リストの部分です。

```
REGISTER QUERY q1 SELECT SUM(s1.a) AS a1, s1.b, s1.c FROM s1[ROWS 100]
                  GROUP BY s1.b, s1.c;
```

4.4.13　リレーション参照

形式

```
リレーション参照::={リレーション名
                    |ストリーム名'['ウィンドウ指定']'}
                    〔▲AS▲相関名〕
```

説明

検索するリレーションを指定します。リレーション参照は，FROM 句で指定します。

オペランド

リレーション名

検索するリレーションの名称を指定します。

リレーション名に指定できるのは，問い合わせ結果をストリーム句でストリーム化しないで取得した場合のクエリ名です。

ストリーム名

検索するストリームの名称を指定します。

ストリーム名に指定できるのは，問い合わせ結果をストリーム句でストリーム化して取得した場合のクエリ名です。

ウィンドウ指定

ウィンドウ指定については，「4.4.14　ウィンドウ指定」を参照してください。

相関名

リレーション参照の名称が長く複雑である場合，別名として指定します。名称は名前の規則に従います。名前の規則の詳細については，「3.2.4　名前の指定」を参照してください。

相関名には，1 つのSELECT 句中に指定したほかの相関名と同じ名称は指定できません。また，ほかの表識別子と同じ名称も指定できません。表識別子については，「3.2.5　名前の修飾」を参照してください。

構文規則

- リレーション参照は，FROM 句で指定します。
- リレーション参照に指定した相関名は 1 つのクエリ文の中でだけ有効です。複数のクエリ文にわたる利用はできません。

  例えば，次のように，リレーション参照に指定した相関名a1 およびa2 は，1 つのクエリ文の中で利用できます。

  (正しい指定例)

```
REGISTER QUERY q1 SELECT a1.c1, a2.c1 FROM s1[NOW] AS a1, s2[NOW] AS a2
                  WHERE a1.c1 > 1 GROUP BY a1.c1, a2.c1
                  HAVING a2.c1 > 1;
```

  次の例では，q2 でリレーション参照に指定したq1 から相関名a1 およびa2 は取得されないため，利用できません。

  (誤った指定例)

```
REGISTER QUERY q1 SELECT a1.c1, a2.c1 FROM s1[NOW] AS a1, s2[NOW] AS a2;
REGISTER QUERY q2 SELECT a1.c1, a2.c1 FROM q1 WHERE a1.c1 > 1
                   GROUP BY a1.c1, a2.c1 HAVING a2.c1 > 1;
```

- リレーション参照に指定する相関名には，ストリーム名およびクエリ名と異なる名称を指定します。

  次の例では，相関名に指定したs2 はストリーム名として定義されているので指定できません。

  (誤った指定例)

```
REGISTER STREAM s1 (id INT, name VARCHAR(10));
REGISTER STREAM s2 (id INT, name VARCHAR(10));
REGISTER QUERY q1 SELECT * FROM s1[NOW] AS s2;
```

  次の例では，相関名に指定したq1 はクエリ名として定義されているので指定できません。

  (誤った指定例)

```
REGISTER QUERY q1 SELECT * FROM s1[NOW];
REGISTER QUERY q2 SELECT * FROM s2[NOW] AS q1;
```

- リレーション参照に同一名称のストリームを指定する場合は，次のように相関名を指定して一意に識別できるようにします。

  (正しい指定例)

```
REGISTER QUERY q1 ISTREAM(
    SELECT ～ FROM s1[RANGE 5] AS old, s1[NOW] AS new ～);
```

次の例では，ストリームが一意に識別できないため，エラーになります。

（誤った指定例）

```
REGISTER QUERY q1 ISTREAM(
    SELECT ～ FROM s1[RANGE 5], s1[NOW] ～);
```

注意事項

ありません。

使用例

リレーションs1 の列a およびb のデータを出力します。下線部がリレーション参照の部分です。

```
REGISTER QUERY q1 SELECT s1.a, s1.b FROM s1[ROWS 100];
```

4.4.14　ウィンドウ指定

形式

```
ウィンドウ指定::= {ROWS▲整数定数
                  | RANGE▲整数定数〔▲時間指定〕
                  | NOW
                  | PARTITION▲BY▲列指定リスト▲ROWS▲整数定数}
```

説明

ストリームの生存期間を指定します。ウィンドウ指定は，リレーション参照で指定します。

オペランド

ROWS

生存する行数として，1～100,000 の値を指定します。

RANGE

生存する時間を指定します。時刻解像度を指定するクエリでは，必ずRANGE を指定してください。

整数定数

時間の値を指定します。整数定数の値の範囲については，「4.4.15　時間指定」を参照してください。

時間指定

*整数定数*で指定した時間の単位を指定します。時間指定の指定については，「4.4.15　時間指定」を参照してください。

指定を省略した場合，*整数定数*で指定した値は秒単位の値として認識されます。

NOW

生存する時間を現時刻だけにします。

PARTITION BY

列指定リストで指定した列名の値ごとにROWS の処理をします。

列指定リスト

列指定リストの指定については，「4.4.12　列指定リスト」を参照してください。

列指定リストには，列名の指定ができます。

構文規則

ありません。

注意事項

- ROWS に生存する行数の値の範囲を超えた値を指定した場合，その最大値が仮定され，処理が継続されます。
- ROWS を指定した場合に，生存する行数の値が 0 のときは，その最小値が仮定され，処理が継続されます。負または整数定数以外の場合，エラーになります。

使用例

使用例 1

リレーションs1 のストリームの生存期間を 2 秒間に指定します。下線部がウィンドウ指定の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[RANGE 2 SECOND];
```

使用例 2

リレーションs1 のストリームを列a の値ごとに生存する行を 2 に指定します。下線部がウィンドウ指定の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[PARTITION BY s1.a ROWS 2];
```

4.4.15　時間指定

形式

```
時間指定::= {SECOND | MILLISECOND | MINUTE | HOUR | DAY}
```

説明

時間の単位を指定します。

オペランド

SECOND

時間の単位として，秒を指定します。

MILLISECOND

時間の単位として，ミリ秒を指定します。

MINUTE

時間の単位として，分を指定します。

HOUR

時間の単位として，時間を指定します。

DAY

時間の単位として，日を指定します。

構文規則

時間の値そのものは，時間指定とは別に，整数定数で指定します。時間指定と整数定数の関係を次の表に示します。

表 4-4　時間指定と整数定数の関係

項番	時間指定（時間の単位を指定）		整数定数（時間の値を指定）	
	オペランド	意味	最小値	最大値
1	SECOND	秒	1	2678400
2	MILLISECOND	ミリ秒	1	86400000
3	MINUTE	分	1	44640
4	HOUR	時	1	744
5	DAY	日	1	31

注意事項

- 時間指定の値の範囲を超えた値を整数定数に指定した場合，その最大値を仮定し処理を継続します。
- 時間指定の値として，整数定数に0 を指定した場合，最小値である1 を仮定し処理を継続します。負または整数定数以外の値を指定した場合，エラーになります。

使用例

リレーションs1 のストリームの生存期間を 2 秒間に指定します。下線部が時間指定の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[RANGE 2 SECOND];
```

4.4.16　探索条件

形式

HAVING 句の場合

```
探索条件::=比較述語〔▲AND▲探索条件〕
```

WHERE 句の場合

```
探索条件::= { {'('探索条件')'|比較述語}
            |NOT {▲0'('探索条件')'|▲比較述語}
            |探索条件▲OR▲ {'('探索条件')'|比較述語}
            |探索条件▲AND▲ {'('探索条件')'|比較述語} }
```

説明

指定した条件に従って論理演算を実行し，真の結果だけをリレーションとして取得します。

探索条件は，WHERE 句およびHAVING 句で指定します。

オペランド

比較述語

比較述語の指定については，「4.4.17　比較述語」を参照してください。

構文規則

探索条件は，WHERE 句またはHAVING 句に指定します。句によって指定できる論理演算が異なります。指定できる論理演算を次の表に示します。

表 4-5　指定できる論理演算

項番	論理演算	WHERE 句	HAVING 句
1	AND	指定できます。	指定できます。
2	NOT	指定できます。	指定できません。
3	OR	指定できます。	指定できません。

注意事項

- WHERE 句の指定には，次の制限があります。
  - 論理演算子OR は，2 つの項の比較述語がすべて同一リレーションであるときだけ指定できます。
  - 論理演算子NOT は，その右単項の比較述語がすべて同一リレーションであるときだけ利用できます。

  なお，複数リレーションは指定できません。s1, s2 がスキーマ(c1 INT, c2 INT, c3 INT)であるリレーションのときの例を次に示します。

（正しい指定例：同一リレーションのOR を指定）

```
s1.c1 < 1 OR s1.c2 < 1
```

（誤った指定例：複数リレーションのOR を指定）

```
s1.c1 < 1 OR s2.c1 < 1
```

（正しい指定例：同一リレーションのNOT を指定）

```
NOT(s1.c1 < 1) AND NOT(s2.c1 < 1)
```

（誤った指定例：複数リレーションのNOT を指定）

```
NOT((s1.c1 < 1) AND (s2.c1 <1))
```

（正しい指定例：同一リレーションのOR とNOT を指定）

```
(s1.c1 < 1 OR s1.c2 < 1) AND NOT(s2.c1 < 1)
```

（誤った指定例：複数リレーションのOR とNOT を指定）

```
s1.c1 < 1 OR s1.c2 < 1 AND NOT(s2.c1 < 1)…
```

- 論理演算の評価順序は，括弧内，NOT，AND，OR の順です。例えば，次の 2 つの演算は同じ意味になります。

```
s1.c1 < 1 OR (s1.c2 < 1 AND NOT (s1.c3 < 3))
s1.c1 < 1 OR (s1.c2 < 1 AND s1.c3 >= 3)
```

- 探索条件の括弧（( )）は，省略できます。
- 探索条件内に指定できる比較述語および括弧の個数は，1～255 個です。個数の数え方の例を次に示します。

NOT c1>1

：比較述語がc1>1 の 1 つだけなので，個数は1 になります。

NOT(c1>1)

：比較述語がc1>1 の 1 つ，括弧が一組なので，個数は2 になります。

c1>1 AND c2>1

：比較述語がc1>1，c2>1 の 2 つなので個数は2 になります。

(c1>1) AND (c2>1)

：比較述語がc1>1，c2>1 の 2 つ，括弧が二組なので，個数は4 になります。

使用例

リレーションs1 の列a の値が5 より小さく，1 より大きいものを出力します。下線部が探索条件の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[ROWS 100] WHERE s1.a < 5 AND s1.a > 1;
```

4.4.17　比較述語

形式

WHERE 句の場合

```
比較述語::=値式 比較演算子 値式
    比較演算子::= {< | <= | > | >= | = | !=}
```

HAVING 句の場合

```
比較述語::=比較演算項 比較演算子 比較演算項
    比較演算項::= {列指定 | 列名 | スカラ関数 | 集合関数 | 定数}
    比較演算子::= {< | <= | > | >= | = | !=}
```

説明

真，または偽の論理値を与えるための条件を指定します。

オペランド

値式

値式の指定については，「4.4.18　値式」を参照してください。

比較演算子

比較演算項を比較するための比較演算子として，「<」，「<=」，「>」，「>=」，「=」，または「!=」を指定します。

比較演算項

比較対象の項として，列指定，スカラ関数，集合関数，または定数を指定します。

なお，先行するFROM 句のリレーション参照が 1 つであれば，その比較演算項では列名の指定ができます。

列指定

列指定については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。

列名

後続のFROM 句のリレーション参照が 1 つである場合，列名を指定できます。

後続のFROM 句のリレーション参照が複数ある場合は，列指定を指定してください。

スカラ関数

スカラ関数の指定については，「4.4.21　スカラ関数」を参照してください。

集合関数

集合関数の指定については，「4.4.20　集合関数」を参照してください。

定数

定数については，「4.4.19　定数」を参照してください。

構文規則

比較述語を「比較演算項 X 比較演算子 比較演算項 Y」として，比較演算子の意味を次の表に示します。

表 4-6　比較演算子の種類と機能

項番	比較演算子の指定	意味
1	*比較演算項*X=*比較演算項*Y	比較演算項X と比較演算項Y が等しいことを示します。
2	*比較演算項*X!=*比較演算項*Y	比較演算項X と比較演算項Y が等しくないことを示します。
3	*比較演算項*X<*比較演算項*Y	比較演算項X が比較演算項Y より小さいことを示します。
4	*比較演算項*X<=*比較演算項*Y	比較演算項X が比較演算項Y 以下であることを示します。
5	*比較演算項*X>*比較演算項*Y	比較演算項X が比較演算項Y より大きいことを示します。
6	*比較演算項*X>=*比較演算項*Y	比較演算項X が比較演算項Y 以上であることを示します。

- 左右の比較演算項のデータは，比較できるデータ型にしてください。比較できるデータ型については，「3.4　データの比較」を参照してください。
- 数データ同士の比較で，比較するデータの型が異なる場合，範囲の広い方の型で比較します。範囲の広さを次に示します。

```
DECIMAL = NUMERIC > FLOAT = DOUBLE > REAL > BIGINT > INTEGER > SMALLINT > TINYINT
```

注意事項

- 定数と定数の比較はできません。比較した場合，エラーになります。
- 比較演算項または値式のどちらかに，列指定または列名を含める必要があります。
- 比較述語に指定する値式では，異なるデータ識別子の四則演算はできません。例えば，次のような指定はできません。

```
WHERE s1.a + s2.a < 5
```

使用例

リレーションs1 の列a の値が5 より小さいものを出力します。下線部が比較述語の部分です。

```
REGISTER QUERY q1 SELECT * FROM s1[RANGE 10 SECOND] WHERE s1.a < 5;
```

4.4.18　値式

形式

```
値式::=〔符号〕項〔演算子〔符号〕項〕…
    項::=〔キャスト指定〕値式一次子
    演算子::= {+|-|*|/}
    符号::= {+|-}
    キャスト指定::= {'('TINYINT')'|'('SMALLINT')'|'('INT〔EGER〕')'
                    |'('BIGINT')'|'('REAL')'|'('FLOAT')'|'('DOUBLE')'
                    |'('DEC〔IMAL〕')'}
    値式一次子::= {列指定|列名|定数|スカラ関数|'('値式')'}
```

説明

値を指定します。

オペランド

符号

「+」または「-」を指定します。

項

キャスト指定，および値式一次子を指定します。

キャスト指定

キャスト指定に指定するデータ型については，「3.3　CQL のデータ型」を参照してください。

値式一次子

列指定，列名，定数，スカラ関数，または値式を指定します。

＜列指定＞

列指定については，「3.2.5　名前の修飾」の「(2)　列指定」を参照してください。

＜列名＞

後続のFROM 句のリレーション参照が 1 つである場合，列名を指定できます。

後続のFROM 句のリレーション参照が複数ある場合は，列指定を指定してください。

＜定数＞

定数については，「4.4.19　定数」を参照してください。

＜スカラ関数＞

スカラ関数の指定については，「4.4.21　スカラ関数」を参照してください。

演算子

「+」，「-」，「*」，または「/」を指定します。演算子の後ろに符号を指定した項を指定する場合，その項は括弧で囲む必要があります。符号と演算子の例を次に示します。

```
-a+(-b)
a*(-b+1)
a/(-b)+1
```

構文規則

- 値式中に四則演算子を指定することで，四則演算ができます。
- 値式の結果のデータ型は，四則演算または値式一次子の結果と同じデータ型になります。また，データ構造も四則演算または値式一次子の結果と同じになります。
- 四則演算（2 項演算）は数データで実行します。演算中に数データ以外のデータ型（文字データや時刻データなど）を含めることはできません。含めた場合はエラーとなります。

  数データでは，異なるデータ型間で演算できます。データ型と演算結果のデータ型の関係を次の表に示します。

表 4-7　四則演算（2 項演算）の演算項のデータ型と演算結果のデータ型の関係

項番	第 1 演算項データ型	第 2 演算項データ型								
		TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
1	TINYINT	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
2	SMALLINT	SMALLINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
3	INTEGER	INTEGER	INTEGER	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
4	BIGINT	BIGINT	BIGINT	BIGINT	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
5	REAL	REAL	REAL	REAL	REAL	REAL	FLOAT	DOUBLE	DECIMAL	NUMERIC
6	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	FLOAT	DECIMAL	NUMERIC
7	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DOUBLE	DECIMAL	NUMERIC
8	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL	DECIMAL
9	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC	NUMERIC

- DECIMAL 型またはNUMERIC 型を含む演算の場合，DECIMAL 型またはNUMERIC 型以外のデータ型の値は，次のパラメーターの値に従って丸められ，DECIMAL 型（NUMERIC 型）の値に変換されて演算されます。
  - system_config.properties のquery.decimalMaxPrecision

    精度を指定するパラメーターです。
  - system_config.properties のquery.decimalRoundingMode

    丸め方を指定するパラメーターです。

  また，演算結果も同様に丸められます。

注意事項

- 演算途中でオーバーフローが発生した場合，処理は継続されます。このとき，整数であればオーバーフローしたままの値（入りきれない上位バイトをカットし，下位バイトをそのまま使用する）となり，浮動小数点値であれば無限大になります。ただし，DECIMAL 型またはNUMERIC 型の値と，浮動小数点値との演算で，浮動小数点値でオーバーフローが発生した場合は，エラー（クエリグループが閉塞）になります。
- 除算で第 2 演算項の値に0 を指定した場合は，エラーになります。
- 文字データおよび文字列定数に対する演算は，エラーになります。
- 日付/時刻データ（文字列定数を含む）に対するキャストは，エラーになります。
- 文字データの場合，文字データの書式が数データに合うものだけがキャストできます。文字データの書式については，「3.4　データの比較」を参照してください。
- DECIMAL 型にキャストした場合，次のパラメーターの値に従って丸められ，DECIMAL 型に変換されます。
  - system_config.properties のquery.decimalMaxPrecision

    精度を指定するパラメーターです。
  - system_config.properties のquery.decimalRoundingMode

    丸め方を指定するパラメーターです。

使用例

リレーションs1 の列a をDECIMAL 型にキャストした値と，列b をINT 型にキャストした値を出力します。下線部が値式の部分です。

```
REGISTER QUERY q1 SELECT (DECIMAL)s1.a AS xxx, (INT)s1.b AS yyy FROM s1[ROWS 100];
```

4.4.19　定数

形式

```
定数::= {文字列定数 | 数定数}
    文字列定数::= {日付データ | 時刻データ |
                   時刻印データ | 文字列}
    数定数::= {整数定数 | 浮動小数点定数 | 10進定数}
```

説明

定数を指定します。1 つ以上の値を指定できます。

オペランド

文字列定数

日付データ，時刻データ，時刻印データ，または文字列を指定します。データ型はVARCHAR 型です。

日付データ

日付データの指定については，「3.2.6　定数の指定」の「(2)　日付データの規定の文字列表現」を参照してください。

時刻データ

時刻データの指定については，「3.2.6　定数の指定」の「(3)　時刻データの規定の文字列表現」を参照してください。

時刻印データ

時刻印データの指定については，「3.2.6　定数の指定」の「(4)　時刻印データの規定の文字列表現」を参照してください。

文字列

文字列の指定については，「3.2.6　定数の指定」の「(1)　定数の種類と表記方法」を参照してください。

数定数

整数定数，浮動小数点定数または 10 進定数を指定します。

整数定数

整数定数のデータ型は，数字の末尾にL またはl を指定するとBIGINT 型になります。数字だけを指定すると，INTEGER 型になります。

整数定数の指定については，「3.2.6　定数の指定」の「(1)　定数の種類と表記方法」を参照してください。

浮動小数点定数

浮動小数点定数のデータ型はDOUBLE 型です。

浮動小数点定数の指定については，「3.2.6　定数の指定」の「(1)　定数の種類と表記方法」を参照してください。

10 進定数

10 進定数のデータ型はDECIMAL 型またはNUMERIC 型です。

10 進定数の指定については，「3.2.6　定数の指定」の「(1)　定数の種類と表記方法」を参照してください。

構文規則

ありません。

注意事項

- 文字列にエスケープ文字はありません。文字列にタブを入れた場合や途中で改行した場合は，その文字コードが文字列定数に埋め込まれます。
- 値がオーバーフローした浮動小数点定数はエラーになりません。値は無限大になります。
- 値がDECIMAL 型（NUMERIC 型）の上下限値の範囲外となった 10 進定数はエラーになりません。値は CQL で記述した値で表現されます。

使用例

リレーションs1 の列a の値に-5.3 を加算し，列b に4 を乗じた結果を出力します。下線部が定数の部分です。

```
REGISTER QUERY q1 SELECT -5.3+s1.a AS xxx, 4*s1.b AS xxy FROM s1[ROWS 100];
```

4.4.20　集合関数

形式

```
集合関数::= {COUNT'('値式 | *')' | 一般集合関数 | 組み込み集合関数 | 外部定義集合関数}
    一般集合関数::= {MAX | MIN | SUM | AVG} '('値式')'
```

説明

複数行から算出される値を求めます。

オペランド

COUNT

入力行数を求めます。

値式

集合関数の引数を指定します。値式については，「4.4.18　値式」を参照してください。

*

すべての行数を求めます。

一般集合関数

MAX，MIN，SUM，AVG のどれかを指定します。引数中には，列指定を含む値式を指定してください。

- MAX

  最大値を求める集合関数です。

- MIN

最小値を求める集合関数です。

- SUM

  合計値を求める集合関数です。

- AVG

  平均値を求める集合関数です。

組み込み集合関数

組み込み集合関数の指定については，「4.4.26　組み込み集合関数」を参照してください。

外部定義集合関数

外部定義集合関数の指定については，「4.4.23　外部定義集合関数」を参照してください。

構文規則

- 一般集合関数には，引数にALL（全件取得）を制御する修飾子は存在しません。すべてALL で動作します。
- 集合関数は，その集合関数が含まれるリレーション式の選択式中，またはHAVING 句中に指定してください。
- FROM 句，WHERE 句，およびGROUP BY 句のうち，最後に指定された句で取得されるリレーションを集合関数の入力とします。ただし，GROUP BY 句を指定しない場合，FROM 句またはWHERE 句の結果がグループ化列を持たない 1 つのグループとなります。
- COUNT または一般集合関数の引数中に値式を指定する場合，列指定を含む値式を指定してください。
- 組み込み集合関数の引数については，「4.4.26　組み込み集合関数」を参照してください。
- 外部定義集合関数の引数については，「4.4.23　外部定義集合関数」を参照してください。
- リレーション式のSELECT 句の選択式に集合関数および列指定，または列名を指定する場合は，GROUP BY 句を適用します。
- 集合関数の引数の型（列のデータ型）と結果の型の対応は，次の表に示すとおりです。

表 4-8　集合関数の引数の型と結果の型の対応

項番	引数の型	集合関数				
		COUNT(引数)	MAX(引数)	MIN(引数)	SUM(引数)	AVG(引数)
1	INTEGER	INTEGER	INTEGER	INTEGER	INTEGER	INTEGER
2	SMALLINT	INTEGER	SMALLINT	SMALLINT	SMALLINT	SMALLINT
3	TINYINT	INTEGER	TINYINT	TINYINT	TINYINT	TINYINT
4	BIGINT	INTEGER	BIGINT	BIGINT	BIGINT	BIGINT
5	DECIMAL	INTEGER	DECIMAL	DECIMAL	DECIMAL	DECIMAL
6	NUMERIC	INTEGER	NUMERIC	NUMERIC	NUMERIC	NUMERIC
7	REAL	INTEGER	REAL	REAL	REAL	REAL

項番	引数の型	集合関数				
		COUNT(引数)	MAX(引数)	MIN(引数)	SUM(引数)	AVG(引数)
8	FLOAT	INTEGER	FLOAT	FLOAT	FLOAT	FLOAT
9	DOUBLE	INTEGER	DOUBLE	DOUBLE	DOUBLE	DOUBLE
10	CHAR	INTEGER	CHAR	CHAR	－	－
11	VARCHAR	INTEGER	VARCHAR	VARCHAR	－	－
12	DATE	INTEGER	DATE	DATE	－	－
13	TIME	INTEGER	TIME	TIME	－	－
14	TIMESTAMP	INTEGER	TIMESTAMP	TIMESTAMP	－	－

（凡例）

－：使用できません。

この表で，集合関数の引数の型がTINYINT の場合，集合関数AVG には，入力行数が128 以上になる引数を指定しないでください。また，引数の型がSMALLINT の場合，入力行数が32,768 以上になる引数を指定しないでください。

注意事項

演算途中でオーバーフローが発生した場合は処理を継続します。このとき，整数であればオーバーフローしたままの値（入りきれない上位バイトをカットし，下位バイトをそのまま使用する）となり，浮動小数点値であれば無限大になります。ただし，DECIMAL 型またはNUMERIC 型の値と，浮動小数点値との演算で，浮動小数点値でオーバーフローが発生した場合は，エラー（クエリグループが閉塞）になります。

使用例

リレーションs1 の列a の行数を出力します。下線部が集合関数の部分です。

```
REGISTER QUERY q1 SELECT COUNT(s1.a) AS a1 FROM s1[ROWS 100];
```

4.4.21　スカラ関数

形式

```
スカラ関数::={組み込みスカラ関数 | 外部定義スカラ関数}
```

説明

単一行から算出される値を求めます。

オペランド

- *組み込みスカラ関数*
  組み込みスカラ関数の指定については，「4.4.27　組み込みスカラ関数」を参照してください。
- *外部定義スカラ関数*
  外部定義スカラ関数の指定については，「4.4.24　外部定義スカラ関数」を参照してください。

構文規則

- 比較述語の左辺または右辺では，スカラ関数の引数の列指定で，すべて同じデータ識別子の列を指定する必要があります。
- 組み込みスカラ関数の引数については，「4.4.27　組み込みスカラ関数」を参照してください。
- 外部定義スカラ関数の引数については，「4.4.24　外部定義スカラ関数」を参照してください。

注意事項

ありません。

使用例

リレーションs1 の列a の絶対値を出力します。下線部がスカラ関数の部分です。

```
REGISTER QUERY q1 SELECT DABS(s1.a) AS a1 FROM s1[ROWS 100];
```

4.4.22　ストリーム間演算関数

形式

```
ストリーム間演算関数::=外部定義ストリーム間演算関数
```

説明

ストリームデータ間の演算を実行します。

オペランド

外部定義ストリーム間演算関数

外部定義ストリーム間演算関数の指定については，「4.4.25　外部定義ストリーム間演算関数」を参照してください。

構文規則

ありません。

注意事項

ありません。

使用例

「4.4.25　外部定義ストリーム間演算関数」の使用例を参照してください。

4.4.23　外部定義集合関数

形式

外部定義集合関数::=$#*関数グループ名*.*関数名*'('*値式*[,*値式*]...')'

説明

「4.4.20　集合関数」に指定する外部定義集合関数を指定します。

CQL で次の個所に指定できます。

- SELECT 句の選択式
- HAVING 句

オペランド

$#

外部定義集合関数であることを示す記号です。

関数グループ名

外部定義集合関数の実装クラスを指定した外部定義関数定義ファイルの関数グループ名を指定します。ここで指定した値と，外部定義関数定義ファイルのFunctionGroup タグのname 属性が一致する（大文字小文字は区別しない），FunctionGroup タグの情報が使用されます。外部定義関数定義ファイルの詳細は「12.3　外部定義関数の作成例（Java）」を参照してください。

関数名

実装メソッドを識別する関数名を指定します。ここで指定した値と，外部定義関数定義ファイルのfunc タグのname 属性に指定した値が一致する（大文字小文字は区別しない），func タグの情報が使用されます。外部定義関数定義ファイルの詳細は「12.3　外部定義関数の作成例（Java）」を参照してください。

値式

値式については，「4.4.18　値式」を参照してください。

構文規則

- 1 つ以上の引数には列指定を含む値式を指定してください。
- 列指定を指定した引数は，GROUP BY 句，WHERE 句，FROM 句のうち，最後に指定された句の結果として得られる各グループを入力とします。ただし，GROUP BY 句を指定しない場合，WHERE 句またはFROM 句の結果がグループ化列を持たない 1 つのグループになります。
- 比較述語の左辺または右辺では，外部定義集合関数の引数の列指定で，すべて同じデータ識別子の列を指定する必要があります。
- 次のどちらかの場合，SELECT 句の選択式に使用する列指定または列名でGROUP BY 句を指定してください。
  - SELECT 句の選択式に外部定義集合関数と列指定を指定する場合
  - SELECT 句の選択式に外部定義集合関数，列指定および列名を指定する場合

注意事項

ありません。

4.4.24　外部定義スカラ関数

形式

外部定義スカラ関数::=$$*関数グループ名*.*関数名*'('*値式*[,*値式*]...')'

説明

「4.4.21　スカラ関数」に指定する外部定義スカラ関数を指定します。

CQL で次の個所に指定できます。

- SELECT 句の選択式
- WHERE 句
- HAVING 句

オペランド

$$

外部定義スカラ関数であることを示す記号です。

関数グループ名

外部定義スカラ関数の実装クラスを指定した外部定義関数定義ファイルの関数グループ名を指定します。ここで指定した値と，外部定義関数定義ファイルのFunctionGroup タグのname 属性が一致する（大

文字小文字は区別しない），FunctionGroup タグの情報が使用されます。外部定義関数定義ファイルの詳細は「12.3　外部定義関数の作成例（Java）」を参照してください。

関数名

実装メソッドを識別する関数名を指定します。ここで指定した値と，外部定義関数定義ファイルのfuncタグのname 属性に指定した値が一致する（大文字小文字は区別しない），func タグの情報が使用されます。外部定義関数定義ファイルの詳細は「12.3　外部定義関数の作成例（Java）」を参照してください。

値式

値式については，「4.4.18　値式」を参照してください。

構文規則

比較述語の左辺または右辺では，外部定義スカラ関数の引数の列指定で，すべて同じデータ識別子の列を指定する必要があります。

注意事項

ありません。

4.4.25　外部定義ストリーム間演算関数

形式

```
外部定義ストリーム間演算関数::=$*関数グループ名.関数名
['['定数[,定数]...']']
'('ストリーム名[,ストリーム名]...')'
 [ΔASΔ'('相関名[,相関名]...')']
```

説明

「4.4.22　ストリーム間演算関数」に指定する外部定義ストリーム間演算関数を指定します。

オペランド

$*

外部定義ストリーム間演算関数であることを示す記号です。

関数グループ名

外部定義ストリーム間演算関数の実装クラスを定義するグループ（関数グループ）の名称（外部定義関数定義ファイルのFunctionGroup タグのname 属性の指定値）を指定します。

関数名

外部定義ストリーム間演算関数の実装メソッドを識別する関数の名称（外部定義関数定義ファイルのStreamFunction タグのname 属性の指定値）を指定します。

定数

外部定義ストリーム間演算関数に設定する初期化パラメーターを定数で指定します。定数については，「4.4.19　定数」を参照してください。

定数に指定できる個数は，0～16 個です。

ストリーム名

外部定義ストリーム間演算関数の入力ストリーム名を指定します。

ストリーム名に指定できる個数は，1～16 個です。

相関名

ストリーム間演算関数の使用個所ごとに，出力ストリームの列名を区別して指定したい場合に，列名の別名を指定します。なお，外部定義ストリーム間演算関数の定義数（外部定義関数定義ファイルの ReturnInformation タグの数）と合わせる必要があります。

相関名に指定できる個数は，0～3,000 個です。

構文規則

相関名を指定した場合，後続のクエリの列指定または列名では，相関名を指定してください。外部定義関数定義ファイルで指定した列名（ReturnInformation タグのname 属性）は使用できません。

注意事項

- ストリーム名として指定できるのは，登録されたストリーム名（REGISTER STREAM 句に指定）と問い合わせの結果を，ストリーム句または別のストリーム間演算関数でストリーム化して取得したクエリ名です。
- 定数で指定するパラメーターは，実装クラスのコンストラクターの引数で渡されます。なお，定数で指定するパラメーターの型（CQL のデータ型）と，実装クラスのコンストラクターの引数の型（Java のデータ型）のマッピングについては，「3.3.1　CQL のデータ型と Java のデータ型のマッピング」を参照してください。定数で指定するパラメーターは，「3.2.6　定数の指定」の「(1)　定数の種類と表記方法」の定数の表記方法に従って，CQL のデータ型として解釈されます。
- 相関名で指定する名称は名前の規則に従います。名前の規則の詳細については，「3.2.4　名前の指定」を参照してください。相関名には，同一の外部定義ストリーム間演算関数内で一意の名称を指定してください。

使用例

ストリームs1 のデータを外部定義ストリーム間演算関数に渡し，結果をストリームq1 として出力します。下線部が外部定義ストリーム間演算関数の部分です。

```
REGISTER QUERY q1 $*FG1.STREAM_FUNC[10,1.0,'efg'](s1) AS (ca,cb,cc,cd,ce);
```

4.4.26　組み込み集合関数

形式

```
組み込み集合関数::=統計関数
```

説明

「4.4.20　集合関数」に指定する組み込み集合関数を指定します。

オペランド

統計関数

統計関数の指定については，「5.2　統計関数の詳細」を参照してください。

構文規則

組み込み集合関数の構文規則は，集合関数の構文規則と同じです。「4.4.20　集合関数」の構文規則を参照してください。

注意事項

- クエリに記述した組み込み集合関数の形式に不正がある場合，クエリグループの登録に失敗します。
- 組み込み集合関数は，演算方法について特に説明がない場合，差分演算を実行します。このため，組み込み集合関数の計算量は一定で，入力リレーションのタプル数に比例しません。
- このほかの組み込み集合関数の注意事項は，集合関数の注意事項と同じです。「4.4.20　集合関数」の注意事項を参照してください。

使用例

リレーションs1 の列a，b の相関係数を出力します。下線部が組み込み集合関数の部分です。

```
REGISTER QUERY q1 SELECT CORREL(s1.a, s1.b) AS a1 FROM s1[ROWS 100];
```

4.4.27　組み込みスカラ関数

形式

```
組み込みスカラ関数
::={数学関数|文字列関数|時刻関数|変換関数}
```

説明

「4.4.21　スカラ関数」に指定する組み込みスカラ関数を指定します。

オペランド

数学関数

数学関数の指定については，「5.3　数学関数の詳細」を参照してください。

文字列関数

文字列関数の指定については，「5.4　文字列関数の詳細」を参照してください。

時刻関数

時刻関数の指定については，「5.5　時刻関数の詳細」を参照してください。

変換関数

変換関数の指定については，「5.6　変換関数の詳細」を参照してください。

構文規則

- FLOAT 型はDOUBLE 型のシノニムのため，引数の値式の結果がFLOAT 型の場合，引数はDOUBLE 型として扱います。
- CQL で次の個所に指定できます。
  - SELECT 句の選択式
  - WHERE 句
  - HAVING 句

注意事項

- クエリに記述した組み込みスカラ関数の形式に不正がある場合は，クエリグループの登録に失敗します。
- 正規表現を使用する文字列関数で，指定した正規表現の構文に誤りがある場合は，正規表現の解析ができないため，クエリグループが閉塞します。
- 整数値の 0 除算が発生した場合は，クエリグループが閉塞します。
- 数学関数の演算途中でオーバーフローが発生した場合は，浮動小数点値は無限大になります。

使用例

リレーションs1 の列a の正弦値を出力します。下線部が組み込みスカラ関数の部分です。

```
REGISTER QUERY q1 SELECT DSIN(s1.a) AS a1 FROM s1[ROWS 100];
```

5 CQL で指定する組み込み関数

この章では，CQL で指定する組み込み関数の文法について説明します。なお，文法説明で使用する記述形式については，「4.1　CQL 文法説明で使用する記述形式」を参照してください。

5.1　組み込み関数の一覧

組み込み関数には，組み込み集合関数と組み込みスカラ関数があります。それぞれの一覧を次の表に示します。

組み込み集合関数の一覧

関数の分類	関数名
統計関数	CORREL 関数
	COVAR 関数
	COVAR_POP 関数
	STDDEV 関数
	STDDEV_POP 関数
	VAR 関数
	VAR_POP 関数

組み込みスカラ関数の一覧

関数の分類	関数名
数学関数	ABS 関数
	ACOS 関数
	ASIN 関数
	ATAN 関数
	ATAN2 関数
	CEIL 関数
	COS 関数
	COSH 関数
	DISTANCE 関数
	DISTANCE3 関数
	EXP 関数
	FLOOR 関数
	LN 関数
	LOG 関数
	MOD 関数
	NAN 関数

関数の分類	関数名
数学関数	NEGATIVE_INFINITY 関数
	PI 関数
	POSITIVE_INFINITY 関数
	POWER 関数
	ROUND 関数
	SIN 関数
	SINH 関数
	SQRT 関数
	TAN 関数
	TANH 関数
	TODEGREES 関数
	TORADIANS 関数
文字列関数	CONCAT 関数
	LENGTH 関数
	LEQ 関数
	LGE 関数
	LGT 関数
	LLE 関数
	LLT 関数
	REGEXP_FIRSTSEARCH 関数
	REGEXP_REPLACE 関数
	REGEXP_SEARCH 関数
	SPLENGTH 関数
時刻関数	TIMEDIFF 関数
	TIMESTAMPDIFF 関数
変換関数	BIGINT_TOSTRING 関数
	DECIMAL_TOSTRING 関数
	DOUBLE_TOSTRING 関数
	INT_TOSTRING 関数
	NUMBER_TODATE 関数
	NUMBER_TOTIME 関数

関数の分類	関数名
変換関数	REAL_TOSTRING 関数
	TIME_TONUMBER 関数
	TIMESTAMP_TONUMBER 関数

5.2　統計関数の詳細

ここでは，「4.4.26　組み込み集合関数」で指定する統計関数について説明します。

5.2.1　統計関数

形式

```
統計関数::=統計関数名’(’〔値式　〔,値式〕…〕’)’
```

説明

統計関数は，複数行から成る集合の数値を入力し，統計データを分析する数値を求めます。

オペランド

統計関数名

統計関数の名称です。統計関数の一覧を次の表に示します。

表 5-1　統計関数の一覧

関数名	説明
CORREL 関数	ピアソンの相関係数を算出します。
COVAR 関数	標本共分散を算出します。
COVAR_POP 関数	母集団共分散を算出します。
STDDEV 関数	標準偏差を算出します。
STDDEV_POP 関数	母集団標準偏差を算出します。
VAR 関数	分散を算出します。
VAR_POP 関数	母集団分散を算出します。

値式

値式の指定については，「4.4.18　値式」を参照してください。なお，値式の個数は関数によって異なります。統計関数の各関数の説明を参照してください。

使用例

DOUBLE 型の値S1.HEIGHT とS1.WEIGHT から相関係数を算出します。

```
register query FILTER ISTREAM (SELECT CORREL(S1.HEIGHT,S1.WEIGHT)
  FROM S1[ROWS 10] GROUP BY S1.HEIGHT, S1.WEIGHT );
```

5.2.2 CORREL 関数

形式

```
CORREL '('value1,value2')'
```

説明

次に示す計算式に従ってピアソンの相関係数を算出します。

$$Correl(x_i, y_i) = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2 \sum_{i=1}^{n}(y_i-\bar{y})^2}}$$

(x_i, y_i)は2組の数値のデータ列。i=1, 2, . . . , nで, nは列数。

$\bar{x}, \bar{y}$は, $x=\{x_i\}, y=\{y_i\}$ の相加平均。

引数

value1

独立した変数（*value2*）に従属する変数を値式で指定します。

value2

独立した変数を値式で指定します。

戻り値

相関係数を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
CORREL	*value1*	DOUBLE	DOUBLE
		FLOAT	
	value2	DOUBLE	
		FLOAT	

- *value1*，*value2* のどちらかが非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，x_i の合計値が負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，y_i の合計値が負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，非数（NaN）を返します。

5.2.3 COVAR 関数

形式

```
COVAR'('value1,value2')'
```

説明

次に示す計算式に従って標本共分散を算出します。

$$Covar(x_i,y_i)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})\ (y_i-\bar{y})}{n-1}$$

引数

value1

独立した変数（*value2*）に従属する変数を値式で指定します。

value2

独立した変数を値式で指定します。

戻り値

標本共分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
COVAR	*value1*	DOUBLE	DOUBLE
		FLOAT	
	value2	DOUBLE	
		FLOAT	

- *value1*，*value2* のどちらかが非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，x_i の合計値，y_i の合計値のどちらかが正の無限大（Infinity）で，かつ x_iy_i の積算結果の合計値が正の無限大（Infinity）の場合は，非数（NaN）を返します。

- 上記の計算式で，x_i の合計値，y_i の合計値のどちらかが負の無限大（-Infinity）で，かつ x_iy_i の積算結果の合計値が負の無限大（-Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，非数（NaN）を返します。

5.2.4 COVAR_POP 関数

形式

```
COVAR_POP'('value1,value2')'
```

説明

次に示す計算式に従って母集団共分散を算出します。

$$CovarPop(x_i,y_i)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})\ (y_i-\bar{y})}{n}$$

引数

value1

独立した変数（*value2*）に従属する変数を値式で指定します。

value2

独立した変数を値式で指定します。

戻り値

母集団共分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
COVAR_POP	*value1*	DOUBLE	DOUBLE
		FLOAT	
	value2	DOUBLE	
		FLOAT	

- *value1*，*value2* のどちらかが非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。

- 上記の計算式で，x_i の合計値，y_i の合計値のどちらかが正の無限大（Infinity）で，かつ x_iy_i の積算結果の合計値が正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，x_i の合計値，y_i の合計値のどちらかが負の無限大（-Infinity）で，かつ x_iy_i の積算結果の合計値が負の無限大（-Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，正のゼロ（0.0）を返します。

5.2.5 STDDEV 関数

形式

```
STDDEV'('value')'
```

説明

次に示す計算式に従って標準偏差を算出します。

$$Stddev(x_i)=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}}$$

引数

value

演算対象の数値を値式で指定します。

戻り値

標準偏差を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
STDDEV	*value*	DOUBLE	DOUBLE
		FLOAT	

- *value* が非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，x_i の合計値が負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，非数（NaN）を返します。

5.2.6 STDDEV_POP 関数

形式

```
STDDEV_POP'('value')'
```

説明

次に示す計算式に従って母集団標準偏差を算出します。

$$StddevPop(x_i)=\sqrt{\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}}$$

引数

value

演算対象の数値を値式で指定します。

戻り値

母集団標準偏差を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
STDDEV_POP	*value*	DOUBLE	DOUBLE
		FLOAT	

- *value* が非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，x_i の合計値が負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，正のゼロ（0.0）を返します。

5.2.7　VAR 関数

形式

```
VAR'('value')'
```

説明

次に示す計算式に従って分散を算出します。

$$Var(x_i)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n-1}$$

引数

value

演算対象の数値を値式で指定します。

戻り値

分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
VAR	*value*	DOUBLE	DOUBLE
		FLOAT	

- *value* が非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算方式で，x_i の合計値が負の無限大（-Infinity），または正の無限大（Infinity）で，かつ x_i の 2 乗の合計値が正の無限大（Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，非数（NaN）を返します。

5.2.8 VAR_POP 関数

形式

```
VAR_POP'('value')'
```

説明

次に示す計算式に従って母集団分散を算出します。

$$VarPop(x_i)=\frac{\sum_{i=1}^{n}(x_i-\bar{x})^2}{n}$$

引数

value

演算対象の数値を値式で指定します。

戻り値

母集団分散を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

統計関数名	引数	引数のデータ型	戻り値のデータ型
VAR_POP	*value*	DOUBLE	DOUBLE
		FLOAT	

- *value* が非数（NaN），負の無限大（-Infinity），または正の無限大（Infinity）の場合は，非数（NaN）を返します。
- 上記の計算式で，x_i の合計値が負の無限大（-Infinity），または正の無限大（Infinity）で，かつ x_i の 2 乗の合計値が正の無限大（Infinity）の場合は，非数（NaN）を返します。

注意事項

入力リレーションのタプル数が 1 の場合は，正のゼロ（0.0）を返します。

5.3　数学関数の詳細

ここでは，「4.4.27　組み込みスカラ関数」で指定する数学関数について説明します。

5.3.1　数学関数

形式

数学関数::=*プリフィックス文字数学関数名*’(’〔*値式*　〔,*値式*〕…〕’)’

説明

数学関数は，数値を引数とし，演算した値を返します。

オペランド

プリフィックス文字

数学関数の戻り値のデータ型に対応したアルファベット 1 文字です。プリフィックス文字を次に示します。

表 5-2　プリフィックス文字と戻り値のデータ型

プリフィックス文字	戻り値のデータ型
D	倍精度実数型（DOUBLE／FLOAT）
R	単精度実数型（REAL）
L	倍精度整数型（BIGINT）
I	単精度整数型（INTEGER）

数学関数名

数学関数の名称です。

数学関数名の前に，数学関数の戻り値のデータ型に対応したプリフィックス文字を指定してください。なお，プリフィックス文字と数学関数名の間に空白を入れることはできません。

数学関数の一覧を次の表に示します。

表 5-3　数学関数の一覧

関数名	説明
ABS 関数	絶対値を返します。
ACOS 関数	逆余弦を返します。
ASIN 関数	逆正弦を返します。
ATAN 関数	逆正接を返します。

関数名	説明
ATAN2 関数	直交座標（x，y）から逆正接を返します。
CEIL 関数	引数の値以上で，最も近い整数の浮動小数点値を返します。
COS 関数	余弦を返します。
COSH 関数	双曲線余弦を返します。
DISTANCE 関数	2 次元の 2 点間の距離を返します。
DISTANCE3 関数	3 次元の 2 点間の距離を返します。
EXP 関数	オイラー数の累乗値を返します。
FLOOR 関数	引数の値以下で，最も近い整数の浮動小数点値を返します。
LN 関数	自然対数値を返します。
LOG 関数	対数値を返します。
MOD 関数	剰余を返します。
NAN 関数	NaN を返します。
NEGATIVE_INFINITY 関数	負の無限大（-Infinity）を返します。
PI 関数	π の近似値を返します。
POSITIVE_INFINITY 関数	正の無限大（Infinity）を返します。
POWER 関数	累乗値を返します。
ROUND 関数	指定した小数点の桁数に四捨五入した値を返します。
SIN 関数	正弦を返します。
SINH 関数	双曲線正弦を返します。
SQRT 関数	平方根を返します。
TAN 関数	正接を返します。
TANH 関数	双曲線正接を返します。
TODEGREES 関数	度数に変換した値を返します。
TORADIANS 関数	ラジアンに変換した値を返します。

値式

値式については，「4.4.18　値式」を参照してください。なお，値式の個数は関数によって異なります。数学関数の各関数の説明を参照してください。

使用例

DOUBLE 型の値S1.C1 の絶対値が 5.0 と等しいタプルだけを出力します。DOUBLE 型の絶対値を取得するため，数学関数名「ABS」の前にプリフィックス文字「D」を付けて，「DABS」と指定しています。

```
register query FILTER ISTREAM (SELECT * FROM S1[ROWS 10]
  WHERE DABS(S1.C1) = 5.0 );
```

5.3.2 ABS 関数

形式

*プリフィックス文字*ABS '('*value*')'

説明

引数 *value* の絶対値を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ABS	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	Infinity
負の最小値	正の最大値
value<0	*value* の絶対値（符号を逆にした値）
負の最大値	正の最小値
-0.0（負のゼロ）	0.0（正のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	正の最小値
0<*value*	*value*

value の値	戻り値
正の最大値	正の最大値
Infinity	Infinity

注意事項

ありません。

5.3.3 ACOS 関数

形式

```
プリフィックス文字ACOS '('value')'
```

説明

引数 *value* の逆余弦（アークコサイン）を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ACOS	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value<-1.0	NaN
-1.0≦*value*≦1.0	0.0～π の範囲の値

value の値	戻り値
1.0<*value*	NaN
正の最大値	NaN
Infinity	NaN

注意事項

ありません。

5.3.4 ASIN 関数

形式

*プリフィックス文字*ASIN '('*value*')'

説明

引数 *value* の逆正弦（アークサイン）を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ASIN	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value<-1.0	NaN

value の値	戻り値
-1.0≦*value*<-0.0	-π/2～π/2 の範囲の値
負の最大値	-π/2～π/2 の範囲の値
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	-π/2～π/2 の範囲の値
0.0<*value*≦1.0	-π/2～π/2 の範囲の値
1.0<*value*	NaN
正の最大値	NaN
Infinity	NaN

注意事項

ありません。

5.3.5 ATAN 関数

形式

```
プリフィックス文字ATAN '('value')'
```

説明

引数 *value* の逆正接（アークタンジェント）を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ATAN	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-π/2
負の最小値	計算した逆正接の値（-π/2～π/2）
value<-0.0	計算した逆正接の値（-π/2～π/2）
負の最大値	計算した逆正接の値（-π/2～π/2）
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	計算した逆正接の値（-π/2～π/2）
0.0<*value*	計算した逆正接の値（-π/2～π/2）
正の最大値	計算した逆正接の値（-π/2～π/2）
Infinity	π/2

注意事項

ありません。

5.3.6 ATAN2 関数

形式

```
プリフィックス文字ATAN2 '('valueX,valueY')'
```

説明

直交座標（*valueX*，*valueY*）から逆正接（アークタンジェント）を返します。

直交座標（*valueX*，*valueY*）を極座標（r，θ）に変換し，θ 成分を返します。

引数

valueX

直交座標 *X* を値式で指定します。

valueY

直交座標 *Y* を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ATAN2	*valueX*	DOUBLE	DOUBLE
			FLOAT	
		valueY	DOUBLE	
			FLOAT	

- *valueX*，*valueY* の条件値と戻り値を次の表に示します。

***valueX* の値**	***valueY* の値**	**戻り値**
NaN	任意の値	NaN
-Infinity	NaN	NaN
	-Infinity	-3π/4
	負の最小値	-π
	valueY<-0.0	-π
	負の最大値	-π
	-0.0（負のゼロ）	-π
	0.0（正のゼロ）	π
	正の最小値	π
	0.0<*valueY*	π
	正の最大値	π
	Infinity	3π/4
負の最小値	NaN	NaN
	-Infinity	-π/2
	負の最小値	計算した θ 成分（-3π/4 の近似値）
	valueY<-0.0	計算した θ 成分（-π の近似値）
	負の最大値	計算した θ 成分（-π の近似値）
	-0.0（負のゼロ）	-π
	0.0（正のゼロ）	π
	正の最小値	計算した θ 成分（π の近似値）
	0.0<*valueY*	計算した θ 成分（π の近似値）
	正の最大値	計算した θ 成分（3π/4 の近似値）

valueX の値	*valueY* の値	戻り値
負の最小値	Infinity	π/2
valueX<-0.0	NaN	NaN
	-Infinity	-π/2
	負の最小値	計算した θ 成分（-π/2 の近似値）
	valueY<-0.0	計算した θ 成分
	負の最大値	計算した θ 成分（-π の近似値）
	-0.0（負のゼロ）	-π
	0.0（正のゼロ）	π
	正の最小値	計算した θ 成分（π の近似値）
	0.0<*valueY*	計算した θ 成分
	正の最大値	計算した θ 成分（π/2 の近似値）
	Infinity	π/2
負の最大値	NaN	NaN
	-Infinity	-π/2
	負の最小値	計算した θ 成分（-π/2 の近似値）
	valueY<-0.0	計算した θ 成分（-π/2 の近似値）
	負の最大値	計算した θ 成分
	-0.0（負のゼロ）	-π
	0.0（正のゼロ）	π
	正の最小値	計算した θ 成分
	0.0<*valueY*	計算した θ 成分（π/2 の近似値）
	正の最大値	計算した θ 成分（π/2 の近似値）
	Infinity	π/2
-0.0 （負のゼロ）	NaN	NaN
	-Infinity	-π/2
	負の最小値	-π/2
	valueY<-0.0	-π/2
	負の最大値	-π/2
	-0.0（負のゼロ）	-π
	0.0（正のゼロ）	π
	正の最小値	π/2

valueX の値	*valueY* の値	戻り値
-0.0 （負のゼロ）	0.0<*valueY*	π/2
	正の最大値	π/2
	Infinity	π/2
0.0 （正のゼロ）	NaN	NaN
	-Infinity	-π/2
	負の最小値	-π/2
	valueY<-0.0	-π/2
	負の最大値	-π/2
	-0.0（負のゼロ）	-0.0（負のゼロ）
	0.0（正のゼロ）	0.0（正のゼロ）
	正の最小値	π/2
	0.0<*valueY*	π/2
	正の最大値	π/2
	Infinity	π/2
正の最小値	NaN	NaN
	-Infinity	-π/2
	負の最小値	計算した θ 成分（-π/2 の近似値）
	valueY<-0.0	計算した θ 成分（-π/2 の近似値）
	負の最大値	計算した θ 成分
	-0.0（負のゼロ）	-0.0（負のゼロ）
	0.0（正のゼロ）	0.0（正のゼロ）
	正の最小値	計算した θ 成分
	0.0<*valueY*	計算した θ 成分（π/2 の近似値）
	正の最大値	計算した θ 成分（π/2 の近似値）
	Infinity	π/2
0.0<*valueX*	NaN	NaN
	-Infinity	-π/2
	負の最小値	計算した θ 成分（-π/2 の近似値）
	valueY<-0.0	計算した θ 成分
	負の最大値	計算した θ 成分
	-0.0（負のゼロ）	-0.0（負のゼロ）

valueX の値	*valueY* の値	戻り値
0.0<*valueX*	0.0（正のゼロ）	0.0（正のゼロ）
	正の最小値	計算した θ 成分
	0.0<*valueY*	計算した θ 成分
	正の最大値	計算した θ 成分（π/2 の近似値）
	Infinity	π/2
正の最大値	NaN	NaN
	-Infinity	-π/2
	負の最小値	計算した θ 成分（-π/4 の近似値）
	valueY<-0.0	計算した θ 成分
	負の最大値	計算した θ 成分（-0.0 の近似値）
	-0.0（負のゼロ）	-0.0（負のゼロ）
	0.0（正のゼロ）	0.0（正のゼロ）
	正の最小値	計算した θ 成分（0.0 の近似値）
	0.0<*valueY*	計算した θ 成分
	正の最大値	計算した θ 成分（π/4 の近似値）
	Infinity	π/2
Infinity	NaN	NaN
	-Infinity	-π/4
	負の最小値	-0.0（負のゼロ）
	valueY<-0.0	-0.0（負のゼロ）
	負の最大値	-0.0（負のゼロ）
	-0.0（負のゼロ）	-0.0（負のゼロ）
	0.0（正のゼロ）	0.0（正のゼロ）
	正の最小値	0.0（正のゼロ）
	0.0<*valueY*	0.0（正のゼロ）
	正の最大値	0.0（正のゼロ）
	Infinity	π/4

注意事項

ありません。

5.3.7　CEIL 関数

形式

*プリフィックス文字*CEIL ’(’*value*’)’

説明

引数 *value* の値以上で，最も近い整数の浮動小数点値を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	CEIL	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* が整数の場合は，*value* と同じ値を返します。
- *value* の条件値と戻り値を次の表に示します。

***value* の値**	**戻り値**
NaN	NaN
-Infinity	-Infinity
負の最小値	負の最小値
value≦-1.0	*value* の値以上で最も近い整数の浮動小数点値
-1.0<*value*<-0.0	-0.0（負のゼロ）
負の最大値	-0.0（負のゼロ）
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	1.0
0.0<*value*	*value* の値以上で最も近い整数の浮動小数点値
正の最大値	正の最大値
Infinity	Infinity

注意事項

ありません。

5.3.8　COS 関数

形式

*プリフィックス文字*COS'('*value*')'

説明

引数 *value* の余弦（コサイン）を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	COS	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	計算した余弦の値
value<-0.0	計算した余弦の値
負の最大値	1.0（計算した余弦の値）
-0.0（負のゼロ）	1.0
0.0（正のゼロ）	1.0
正の最小値	1.0（計算した余弦の値）
0.0<*value*	計算した余弦の値

value の値	戻り値
正の最大値	計算した余弦の値
Infinity	NaN

注意事項

ありません。

5.3.9 COSH 関数

形式

```
プリフィックス文字COSH '('value')'
```

説明

引数 *value* の双曲線余弦を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	COSH	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	Infinity
負の最小値	Infinity
value<-0.0	計算した双曲線余弦の値
負の最大値	1.0（計算した双曲線余弦の値）

value の値	戻り値
-0.0（負のゼロ）	1.0
0.0（正のゼロ）	1.0
正の最小値	1.0（計算した双曲線余弦の値）
0.0<*value*	計算した双曲線余弦の値
正の最大値	Infinity
Infinity	Infinity

注意事項

ありません。

5.3.10　DISTANCE 関数

形式

```
プリフィックス文字DISTANCE '('x1,y1,x2,y2')'
```

説明

2 次元の 2 点（*x1*，*y1*），（*x2*，*y2*）間の距離を返します。

引数

x1

点（*x1*，*y1*）の x 軸上の位置を値式で指定します。

y1

点（*x1*，*y1*）の y 軸上の位置を値式で指定します。

x2

点（*x2*，*y2*）の x 軸上の位置を値式で指定します。

y2

点（*x2*，*y2*）の y 軸上の位置を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	DISTANCE	*x1*	DOUBLE	DOUBLE
			FLOAT	
		y1	DOUBLE	
			FLOAT	
		x2	DOUBLE	
			FLOAT	
		y2	DOUBLE	
			FLOAT	

- *x1*，*y1*，*x2*，*y2* のどれかが非数（NaN）の場合は，非数（NaN）を返します。
- *x1*，*y1*，*x2*，*y2* のどれかが正の無限大（Infinity），または負の無限大（-Infinity）で，かつそれ以外の引数が非数（NaN）でない場合は，正の無限大（Infinity）を返します。

注意事項

ありません。

5.3.11 DISTANCE3 関数

形式

*プリフィックス文字*DISTANCE3 '('*x1*,*y1*,*z1*,*x2*,*y2*,*z2*')'

説明

3 次元の 2 点（*x1*，*y1*，*z1*），（*x2*，*y2*，*z2*）間の距離を返します。

引数

x1

点（*x1*，*y1*，*z1*）の x 軸上の位置を値式で指定します。

y1

点（*x1*，*y1*，*z1*）の y 軸上の位置を値式で指定します。

z1

点（*x1*，*y1*，*z1*）の z 軸上の位置を値式で指定します。

x2

点（*x2*，*y2*，*z2*）の x 軸上の位置を値式で指定します。

y2

点（*x2*，*y2*，*z2*）の y 軸上の位置を値式で指定します。

z2

点（*x2*，*y2*，*z2*）の z 軸上の位置を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	DISTANCE3	*x1*	DOUBLE	DOUBLE
			FLOAT	
		y1	DOUBLE	
			FLOAT	
		z1	DOUBLE	
			FLOAT	
		x2	DOUBLE	
			FLOAT	
		y2	DOUBLE	
			FLOAT	
		z2	DOUBLE	
			FLOAT	

- *x1*，*y1*，*z1*，*x2*，*y2*，*z2* のどれかが非数（NaN）の場合は，非数（NaN）を返します。
- *x1*，*y1*，*z1*，*x2*，*y2*，*z2* のどれかが正の無限大（Infinity），または負の無限大（-Infinity）で，かつそれ以外の引数が非数（NaN）でない場合は，正の無限大（Infinity）を返します。

注意事項

ありません。

5.3.12　EXP 関数

形式

*プリフィックス文字*EXP'('*value*')'

説明

引数 *value* を指数としオイラー数 e を累乗した値 e^{value} を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	EXP	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	0.0
負の最小値	0.0（オイラー数 e を累乗した値 e^{value}）
value<-0.0	オイラー数 e を累乗した値 e^{value}
負の最大値	1.0（オイラー数 e を累乗した値 e^{value}）
-0.0（負のゼロ）	1.0
0.0（正のゼロ）	1.0
正の最小値	1.0（オイラー数 e を累乗した値 e^{value}）
0.0<*value*	オイラー数 e を累乗した値 e^{value}
正の最大値	Infinity
Infinity	Infinity

注意事項

ありません。

5.3.13　FLOOR 関数

形式

*プリフィックス文字*FLOOR'('*value*')'

説明

引数 *value* の値以下で，最も近い整数の浮動小数点値を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	FLOOR	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity
負の最小値	負の最小値
value<-0.0	*value* の値以下で最も近い整数の浮動小数点値
負の最大値	-1.0
-0.0（負のゼロ）	-0.0
0.0（正のゼロ）	0.0
正の最小値	0.0
0.0<*value*	*value* の値以下で最も近い整数の浮動小数点値

value の値	戻り値
正の最大値	正の最大値
Infinity	Infinity

注意事項

ありません。

5.3.14　LN 関数

形式

*プリフィックス文字*LN '('*value*')'

説明

引数 *value* の自然対数値を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	LN	*value*	DOUBLE	DOUBLE
			FLOAT	

- *value* の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value<-0.0	NaN
負の最大値	NaN

value の値	戻り値
-0.0（負のゼロ）	-Infinity
0.0（正のゼロ）	-Infinity
正の最小値	*value* の自然対数値
0.0<*value*	*value* の自然対数値
正の最大値	*value* の自然対数値
Infinity	Infinity

注意事項

ありません。

5.3.15　LOG 関数

形式

*プリフィックス文字*LOG '('*base*,*value*')'

説明

引数 *base* を底とした引数 *value* の対数値を返します。戻り値は，$\log_{base}value$ の値となります。

引数

base

底を値式で指定します。

value

真数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	LOG	*base*	DOUBLE	DOUBLE
			FLOAT	
		value	DOUBLE	
			FLOAT	

base，*value* の条件値と戻り値を次の表に示します。

***base* の値**	***value* の値**	**戻り値**
NaN	任意の値	NaN
-Infinity	任意の値	NaN
負の最小値	任意の値	NaN
base<0.0	任意の値	NaN
負の最大値	任意の値	NaN
0.0	任意の値	NaN
正の最小値	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<0.0	NaN
	負の最大値	NaN
	0.0	Infinity
	正の最小値	1.0
	0.0<*value*<1.0	$\log_{base}$*value* の値
	value=1.0	-0.0（負のゼロ）
	1.0<*value*	$\log_{base}$*value* の値
	正の最大値	$\log_{base}$*value* の値
	Infinity	-Infinity
0.0<*base*<1.0	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<0.0	NaN
	負の最大値	NaN
	0.0	Infinity
	正の最小値	$\log_{base}$*value* の値
	0.0<*value*<1.0	$\log_{base}$*value* の値
	value=1.0	-0.0（負のゼロ）
	1.0<*value*	$\log_{base}$*value* の値
	正の最大値	$\log_{base}$*value* の値

base の値	*value* の値	戻り値
0.0<*base*<1.0	Infinity	-Infinity
base=1.0	任意の値	NaN
1.0<*base*	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<0.0	NaN
	負の最大値	NaN
	0.0	-Infinity
	正の最小値	$\log_{base}$*value* の値
	0.0<*value*<1.0	$\log_{base}$*value* の値
	value=1.0	0.0
	1.0<*value*	$\log_{base}$*value* の値
	正の最大値	$\log_{base}$*value* の値
	Infinity	Infinity
正の最大値	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<0.0	NaN
	負の最大値	NaN
	0.0	-Infinity
	正の最小値	$\log_{base}$*value* の値
	0.0<*value*<1.0	$\log_{base}$*value* の値
	value=1.0	0.0
	1.0<*value*	$\log_{base}$*value* の値
	正の最大値	1.0
	Infinity	Infinity
Infinity	任意の値	NaN

注意事項

ありません。

5.3.16　MOD 関数

形式

```
プリフィックス文字MOD '('value1,value2')'
```

説明

引数 *value1* を引数 *value2* で割ったあとの剰余を返します。

引数

value1

被除数を値式で指定します。

value2

除数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
I	MOD	*value1*	INTEGER	INTEGER
		value2	INTEGER	

注意事項

value2 が 0 と等しい場合，整数値の 0 除算が発生するため，クエリグループが閉塞します。

5.3.17　NAN 関数

形式

```
プリフィックス文字NAN '()'
```

説明

NaN を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	NAN	DOUBLE

注意事項

ありません。

5.3.18　NEGATIVE_INFINITY 関数

形式

```
プリフィックス文字NEGATIVE_INFINITY '()'
```

説明

負の無限大（-Infinity）を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	NEGATIVE_INFINITY	DOUBLE

注意事項

ありません。

5.3.19　PI 関数

形式

```
プリフィックス文字PI '()'
```

説明

π の近似値を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	PI	DOUBLE

注意事項

ありません。

5.3.20　POSITIVE_INFINITY 関数

形式

*プリフィックス文字*POSITIVE_INFINITY '()'

説明

正の無限大（Infinity）を返します。

引数

ありません。

戻り値

戻り値のデータ型を次の表に示します。

プリフィックス文字	数学関数名	戻り値のデータ型
D	POSITIVE_INFINITY	DOUBLE

注意事項

ありません。

5.3.21　POWER 関数

形式

```
プリフィックス文字POWER '('base,value')'
```

説明

引数 *base* を引数 *value* で累乗した値を返します。戻り値は，$base^{value}$ の値となります。

引数

base

基数を値式で指定します。

value

指数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	DPOWER	*base*	DOUBLE	DOUBLE
			FLOAT	
		value	DOUBLE	
			FLOAT	

- 演算結果がDOUBLE 型，FLOAT 型のデータの範囲外となる場合は，正の無限大（Infinity），または負の無限大（-Infinity）を返します。
- *base*，*value* の条件値と戻り値を次の表に示します。

***base* の値**	***value* の値**	**戻り値**
NaN	NaN	NaN
	-Infinity	NaN
	負の最小値	NaN
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	NaN
	value<-0.0 で奇数	NaN
	負の最大値	NaN

base の値	*value* の値	戻り値
NaN	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	NaN
	1.0	NaN（*base* と同じ値）
	0.0<*value* で 1 以外の奇数	NaN
	0.0<*value* で偶数	NaN
	0.0<*value* で小数	NaN
	正の最大値	NaN
	Infinity	NaN
-Infinity	NaN	NaN
	-Infinity	0.0（正のゼロ）
	負の最小値	0.0（正のゼロ）
	value<-0.0 で小数	0.0（正のゼロ）
	value<-0.0 で偶数	0.0（正のゼロ）
	value<-0.0 で奇数	-0.0（負のゼロ）
	負の最大値	0.0（正のゼロ）
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	Infinity
	1.0	-Infinity（*base* と同じ値）
	0.0<*value* で 1 以外の奇数	-Infinity
	0.0<*value* で偶数	Infinity
	0.0<*value* で小数	Infinity
	正の最大値	Infinity
	Infinity	Infinity
負の最小値	NaN	NaN
	-Infinity	0.0（正のゼロ）
	負の最小値	0.0（正のゼロ）
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）

base の値	*value* の値	戻り値
負の最小値	*value*<-0.0 で 1 以外の奇数	-0.0（負のゼロ）（*base* の絶対値を *value* で累乗した負の値）
	value=-1.0	*base* の絶対値を *value* で累乗した負の値
	負の最大値	NaN
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	NaN
	1.0	負の最小値（*base* と同じ値）
	0.0<*value* で 1 以外の奇数	-Infinity（*base* の絶対値を *value* で累乗した負の値）
	0.0<*value* で偶数	Infinity（*base* の絶対値を *value* で累乗した値）
	0.0<*value* で小数	NaN
	正の最大値	Infinity（*base* の絶対値を *value* で累乗した値）
	Infinity	Infinity
-1.0<*base*<-0.0	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で奇数	*base* の絶対値を *value* で累乗した負の値
	負の最大値	NaN
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	NaN
	1.0	*base* と同じ値
	0.0<*value* で 1 以外の奇数	*base* の絶対値を *value* で累乗した負の値
	0.0<*value* で偶数	*base* の絶対値を *value* で累乗した値
	0.0<*value* で小数	NaN

base の値	*value* の値	戻り値
-1.0<*base*<-0.0	正の最大値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	Infinity	0.0（正のゼロ）
base<-1.0	NaN	NaN
	-Infinity	0.0（正のゼロ）
	負の最小値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で奇数	*base* の絶対値を *value* で累乗した負の値
	負の最大値	NaN
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	NaN
	1.0	*base* と同じ値
	0.0<*value* で 1 以外の奇数	*base* の絶対値を *value* で累乗した負の値
	0.0<*value* で偶数	*base* の絶対値を *value* で累乗した値
	0.0<*value* で小数	NaN
	正の最大値	Infinity（*base* の絶対値を *value* で累乗した値）
	Infinity	Infinity
base=-1.0	NaN	NaN
	-Infinity	NaN
	負の最小値	1.0
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	1.0
	value<-0.0 で奇数	-1.0
	負の最大値	NaN
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	NaN

base の値	*value* の値	戻り値
base=-1.0	1.0	-1.0
	0.0<*value* で 1 以外の奇数	-1.0
	0.0<*value* で偶数	1.0
	0.0<*value* で小数	NaN
	正の最大値	1.0
	Infinity	NaN
負の最大値	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で小数	NaN
	value<-0.0 で偶数	Infinity（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で奇数	-Infinity（*base* の絶対値を *value* で累乗した負の値）
	負の最大値	NaN
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	NaN
	1.0	*base* と同じ値
	0.0<*value* で 1 以外の奇数	-0.0（負のゼロ）（*base* の絶対値を *value* で累乗した負の値）
	0.0<*value* で偶数	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	0.0<*value* で小数	NaN
	正の最大値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	Infinity	0.0（正のゼロ）
-0.0（負のゼロ）	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity
	value<-0.0 で小数	Infinity
	value<-0.0 で偶数	Infinity

base の値	*value* の値	戻り値
-0.0（負のゼロ）	*value*<-0.0 で奇数	-Infinity
	負の最大値	Infinity
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	0.0（正のゼロ）
	1.0	-0.0（*base* と同じ値）
	0.0<*value* で 1 以外の奇数	-0.0（負のゼロ）
	0.0<*value* で偶数	0.0（正のゼロ）
	0.0<*value* で小数	0.0（正のゼロ）
	正の最大値	0.0（正のゼロ）
	Infinity	0.0（正のゼロ）
0.0（正のゼロ）	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity
	value<-0.0 で小数	Infinity
	value<-0.0 で偶数	Infinity
	value<-0.0 で奇数	Infinity
	負の最大値	Infinity
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	0.0（正のゼロ）
	1.0	0.0（*base* と同じ値）
	0.0<*value* で 1 以外の奇数	0.0（正のゼロ）
	0.0<*value* で偶数	0.0（正のゼロ）
	0.0<*value* で小数	0.0（正のゼロ）
	正の最大値	0.0（正のゼロ）
	Infinity	0.0（正のゼロ）
正の最小値	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity（*base* の絶対値を *value* で累乗した値）

base の値	*value* の値	戻り値
正の最小値	*value*<-0.0 で小数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で偶数	Infinity（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で奇数	Infinity（*base* の絶対値を *value* で累乗した値）
	負の最大値	1.0（*base* の絶対値を *value* で累乗した値）
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	1.0（*base* の絶対値を *value* で累乗した値）
	1.0	正の最小値（*base* と同じ値）
	0.0<*value* で 1 以外の奇数	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	0.0<*value* で偶数	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	0.0<*value* で小数	*base* の絶対値を *value* で累乗した値
	正の最大値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	Infinity	0.0（正のゼロ）
1.0	NaN	NaN
	-Infinity	NaN
	負の最小値	1.0
	value<-0.0 で小数	1.0
	value<-0.0 で偶数	1.0
	value<-0.0 で奇数	1.0
	負の最大値	1.0
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	1.0
	1.0	1.0
	0.0<*value* で 1 以外の奇数	1.0
	0.0<*value* で偶数	1.0

base の値	*value* の値	戻り値
1.0	0.0<*value* で小数	1.0
	正の最大値	1.0
	Infinity	NaN
1.0<*base*	NaN	NaN
	-Infinity	0.0（正のゼロ）
	負の最小値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で小数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で偶数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で奇数	*base* の絶対値を *value* で累乗した値
	負の最大値	1.0（*base* の絶対値を *value* で累乗した値）
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	1.0（*base* の絶対値を *value* で累乗した値）
	1.0	*base* と同じ値
	0.0<*value* で 1 以外の奇数	*base* の絶対値を *value* で累乗した値
	0.0<*value* で偶数	*base* の絶対値を *value* で累乗した値
	0.0<*value* で小数	*base* の絶対値を *value* で累乗した値
	正の最大値	Infinity（*base* の絶対値を *value* で累乗した値）
	Infinity	Infinity
0.0<*base*<1.0	NaN	NaN
	-Infinity	Infinity
	負の最小値	Infinity（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で小数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で偶数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で奇数	*base* の絶対値を *value* で累乗した値
	負の最大値	1.0（*base* の絶対値を *value* で累乗した値）
	-0.0（負のゼロ）	1.0

base の値	*value* の値	戻り値
0.0<*base*<1.0	0.0（正のゼロ）	1.0
	正の最小値	1.0（*base* の絶対値を *value* で累乗した値）
	1.0	*base* と同じ値
	0.0<*value* で 1 以外の奇数	*base* の絶対値を *value* で累乗した値
	0.0<*value* で偶数	*base* の絶対値を *value* で累乗した値
	0.0<*value* で小数	*base* の絶対値を *value* で累乗した値
	正の最大値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	Infinity	0.0（正のゼロ）
正の最大値	NaN	NaN
	-Infinity	0.0（正のゼロ）
	負の最小値	0.0（正のゼロ）（*base* の絶対値を *value* で累乗した値）
	value<-0.0 で小数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で偶数	*base* の絶対値を *value* で累乗した値
	value<-0.0 で奇数	*base* の絶対値を *value* で累乗した値
	負の最大値	1.0（*base* の絶対値を *value* で累乗した値）
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	1.0（*base* の絶対値を *value* で累乗した値）
	1.0	*base* と同じ値
	0.0<*value* で 1 以外の奇数	Infinity（*base* の絶対値を *value* で累乗した値）
	0.0<*value* で偶数	Infinity（*base* の絶対値を *value* で累乗した値）
	0.0<*value* で小数	*base* の絶対値を *value* で累乗した値
	正の最大値	Infinity（*base* の絶対値を *value* で累乗した値）
	Infinity	Infinity
Infinity	NaN	NaN
	-Infinity	0.0（正のゼロ）

base の値	*value* の値	戻り値
Infinity	負の最小値	0.0（正のゼロ）
	value<-0.0 で小数	0.0（正のゼロ）
	value<-0.0 で偶数	0.0（正のゼロ）
	value<-0.0 で奇数	0.0（正のゼロ）
	負の最大値	0.0（正のゼロ）
	-0.0（負のゼロ）	1.0
	0.0（正のゼロ）	1.0
	正の最小値	Infinity
	1.0	Infinity
	0.0<*value* で 1 以外の奇数	Infinity
	0.0<*value* で偶数	Infinity
	0.0<*value* で小数	Infinity
	正の最大値	Infinity
	Infinity	Infinity

注意事項

ありません。

5.3.22　ROUND 関数

形式

```
プリフィックス文字ROUND '('value,scale')'
```

説明

引数 *value* の値を，引数 *scale* で指定した小数点桁数に丸めた値（*scale* ＋ 1 の小数点位置で四捨五入した値）にして返します。

引数

value

丸め対象の数値を値式で指定します。

scale

小数点以下の桁数を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	ROUND	*value*	DOUBLE	DOUBLE
			FLOAT	
		scale	INTEGER	

- DOUBLE 型，FLOAT 型の有効桁数は，16～17 桁です。このため，*value*，*scale* の指定が有効桁数に収まらない場合，有効桁数に丸めた数値を返します。

  DROUND('4235678.285714372187327312 73',10)の場合，戻り値として，4235678.285714372 を返します。

  DROUND('1.285714456745621337293721 37',20)の場合，戻り値として，1.2857144567456213 を返します。
- *value* が非数（NaN），正の無限大（Infinity），負の無限大（-Infinity）の場合，処理が継続できないため，クエリグループが閉塞します。

注意事項

ありません。

5.3.23　SIN 関数

形式

```
プリフィックス文字SIN '('value')'
```

説明

引数 *value* の正弦（サイン）を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SIN	*value*	DOUBLE	DOUBLE
			FLOAT	

value の条件値と戻り値を次の表に示します。

***value* の値**	**戻り値**
NaN	NaN
-Infinity	NaN
負の最小値	計算した正弦の値
value<-0.0	計算した正弦の値
負の最大値	計算した正弦の値
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	計算した正弦の値
0.0<*value*	計算した正弦の値
正の最大値	計算した正弦の値
Infinity	NaN

注意事項

ありません。

5.3.24　SINH 関数

形式

*プリフィックス文字*SINH '('*value*')'

説明

引数 *value* の双曲線正弦を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SINH	*value*	DOUBLE	DOUBLE
			FLOAT	

value の条件値と戻り値を次の表に示します。

***value* の値**	**戻り値**
NaN	NaN
-Infinity	-Infinity
負の最小値	-Infinity（計算した双曲線正弦の値）
value<-0.0	計算した双曲線正弦の値
負の最大値	計算した双曲線正弦の値
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	計算した双曲線正弦の値
0.0<*value*	計算した双曲線正弦の値
正の最大値	Infinity（計算した双曲線正弦の値）
Infinity	Infinity

注意事項

ありません。

5.3.25　SQRT 関数

形式

*プリフィックス文字*SQRT '('*value*')'

説明

引数 *value* の正の平方根を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	SQRT	*value*	DOUBLE	DOUBLE
			FLOAT	

value の条件値と戻り値を次の表に示します。

***value* の値**	**戻り値**
NaN	NaN
-Infinity	NaN
負の最小値	NaN
value<-0.0	NaN
負の最大値	NaN
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	計算した正の平方根
0.0<*value*	計算した正の平方根
正の最大値	計算した正の平方根
Infinity	Infinity

注意事項

ありません。

5.3.26　TAN 関数

形式

```
プリフィックス文字TAN '('value')'
```

説明

引数 *value* の正接（タンジェント）を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TAN	*value*	DOUBLE	DOUBLE
			FLOAT	

value の条件値と戻り値を次の表に示します。

***value* の値**	**戻り値**
NaN	NaN
-Infinity	NaN
負の最小値	計算した正接の値
value<-0.0	計算した正接の値
負の最大値	計算した正接の値
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	計算した正接の値
0.0<*value*	計算した正接の値
正の最大値	計算した正接の値
Infinity	NaN

注意事項

ありません。

5.3.27　TANH 関数

形式

*プリフィックス文字*TANH '('*value*')'

説明

引数 *value* の双曲線正接を返します。

引数

value

値式を指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TANH	*value*	DOUBLE	DOUBLE
			FLOAT	

value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-1.0
負の最小値	-1.0（計算した双曲線正接の値）
value<-0.0	計算した双曲線正接の値
負の最大値	計算した双曲線正接の値
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	計算した双曲線正接の値
0.0<*value*	計算した双曲線正接の値
正の最大値	1.0（計算した双曲線正接の値）
Infinity	1.0

注意事項

ありません。

5.3.28　TODEGREES 関数

形式

*プリフィックス文字*TODEGREES'('*value*')'

説明

ラジアンで計測した角度を度数に変換した値を返します。

なお，ラジアンから度数への変換は，正確ではありません。例えば，TORADIANS(90.0)は，浮動小数で変換するため，正確な π/2 になりません。そのため，COS(TORADIANS(90.0))は，正確に 0.0 になりません。

引数

value

ラジアンで計測した角度を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TODEGREES	*value*	DOUBLE	DOUBLE
			FLOAT	

度数への変換結果がDOUBLE 型，FLOAT 型のデータの範囲外の場合は，正の無限大（Infinity），または負の無限大（-Infinity）を返します。

value の条件値と戻り値を次の表に示します。

***value* の値**	**戻り値**
NaN	NaN
-Infinity	-Infinity
負の最小値	-Infinity（度数に変換した値）
value<-0.0	度数に変換した値
負の最大値	度数に変換した値

value の値	戻り値
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	度数に変換した値
0.0<*value*	度数に変換した値
正の最大値	Infinity（度数に変換した値）
Infinity	Infinity

注意事項

ありません。

5.3.29　TORADIANS 関数

形式

*プリフィックス文字*TORADIANS'('*value*')'

説明

度数で計測した角度をラジアンに変換した値を返します。

なお，度数からラジアンへの変換は正確ではありません。

引数

value

　度数で計測した角度を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

プリフィックス文字	数学関数名	引数	引数のデータ型	戻り値のデータ型
D	TORADIANS	*value*	DOUBLE	DOUBLE
			FLOAT	

value の条件値と戻り値を次の表に示します。

value の値	戻り値
NaN	NaN
-Infinity	-Infinity
負の最小値	ラジアンに変換した値
value<-0.0	ラジアンに変換した値
負の最大値	-0.0（負のゼロ） （ラジアンに変換した値）
-0.0（負のゼロ）	-0.0（負のゼロ）
0.0（正のゼロ）	0.0（正のゼロ）
正の最小値	0.0（正のゼロ） （ラジアンに変換した値）
0.0<*value*	ラジアンに変換した値
正の最大値	ラジアンに変換した値
Infinity	Infinity

注意事項

ありません。

5.4　文字列関数の詳細

ここでは，「4.4.27　組み込みスカラ関数」で指定する文字列関数について説明します。

5.4.1　文字列関数

形式

文字列関数::=*文字列関数名*’(’*値式*　〔,*値式*〕…’)’

説明

文字列関数は，文字列を引数とし，検索または操作した結果を返します。

オペランド

文字列関数名

文字列関数の名称です。文字列関数の一覧を次の表に示します。

表 5-4　文字列関数の一覧

関数名	説明
CONCAT 関数	2 つの引数の文字列を結合した文字列を返します。
LENGTH 関数	引数の文字列の文字数を返します。文字列にサロゲートペアが含まれている場合，サロゲートペア 1 文字を 2 文字として数えて結果を返します。
LEQ 関数	辞書式順序で，引数の文字列が等しいかどうかを比較し，比較結果を返します。
LGE 関数	辞書式順序で，引数の文字列が大きいかどうか，または等しいかどうかを比較し，比較結果を返します。
LGT 関数	辞書式順序で，引数の文字列が大きいかどうかを比較し，比較結果を返します。
LLE 関数	辞書式順序で，引数の文字列が小さいかどうか，または等しいかどうかを比較し，比較結果を返します。
LLT 関数	辞書式順序で，引数の文字列が小さいかどうかを比較し，比較結果を返します。
REGEXP_FIRSTSEARCH 関数	引数の文字列を正規表現パターンで検索し，最初に一致した文字列を返します。
REGEXP_REPLACE 関数	引数の文字列を正規表現パターンで検索し，最初に一致した文字列を置換文字列に置き換えて返します。
REGEXP_SEARCH 関数	引数の文字列を正規表現パターンで検索し，パターンに一致する部分があるかどうかを返します。
SPLENGTH 関数	引数の文字列の文字数を返します。文字列にサロゲートペアが含まれている場合，サロゲートペア 1 文字を 1 文字として数えて結果を返します。

値式

値式については，「4.4.18　値式」を参照してください。なお，値式の個数は関数によって異なります。文字列関数の各関数の説明を参照してください。

使用例

日付データ*YYYY-MM-DD* の固定形式であるCHAR 型のS1.DATE1 から年の文字列を抽出します。正規表現'^¥¥d¥¥d¥¥d¥¥d'は，先頭の数値文字列 4 文字を表現しています。

```
register query FILTER ISTREAM (SELECT REGEXP_FIRSTSEARCH(S1.DATE1,'^¥¥d¥¥d¥¥d¥¥d')
  FROM S1[ROWS 10] );
```

S1.DATE1 が2013-01-23 の場合，2013 を返します。

注意事項

- 「3.3.1　CQL のデータ型と Java のデータ型のマッピング」に示すように，CHAR 型，VARCHAR 型は，Java のjava.lang.String クラスに対応しています。
- 引数にサロゲートペアが含まれる場合の文字列関数の動作について，次の表に示します。

表 5-5　サロゲートペア入力時の動作

文字列関数名	文字列関数の動作
CONCAT	○：正常に動作します。
LENGTH	×：正常に動作しません。サロゲートペア 1 文字を 2 文字として数えて結果を返します。
LEQ	○：正常に動作します。
LGE	○：正常に動作します。
LGT	○：正常に動作します。
LLE	○：正常に動作します。
LLT	○：正常に動作します。
REGEXP_FIRSTSEARCH	○：正常に動作します。
REGEXP_REPLACE	○：正常に動作します。
REGEXP_SEARCH	○：正常に動作します。
SPLENGTH	○：正常に動作します。サロゲートペア 1 文字を 1 文字として数えて結果を返します。

5.4.2 CONCAT 関数

形式

```
CONCAT '('string1,string2')'
```

説明

第 1 引数*string1* と第 2 引数*string2* の文字列を結合した結果を返します。

引数

string1

文字列を値式で指定します。

string2

文字列を値式で指定します。

戻り値

string1 の文字列の最後に*string2* の文字列を連結した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
CONCAT	*string1*	CHAR	VARCHAR
		VARCHAR	
	string2	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。
  string1 が'Abc'，*string2* が'Def'の場合，'AbcDef'を返します。
- 結合した結果の文字列がVARCHAR 型の最大文字数 32,767 文字を超えた場合は，超えた分は切り捨てられます。

注意事項

ありません。

5.4.3 LENGTH 関数

形式

```
LENGTH '('string')'
```

説明

引数*string* の文字数を返します。*string* 内にサロゲートペアが含まれている場合，サロゲートペア 1 文字を 2 文字として数えて結果を返します。

引数

string

文字列を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LENGTH	*string*	CHAR	INTEGER
		VARCHAR	

- 「は＋゛（濁点）」→「ば゛」，「は＋゜（半濁点）」→「ぱ゜」のように 1 字を表す結合文字については，2 文字として結果を返します。
  結合文字：「ば゛すけっと」＝ 6 文字
  非結合文字：「ばすけっと」＝ 5 文字
- 文字列にサロゲートペアが含まれている場合，サロゲートペア 1 文字を，2 文字として数えて結果を返します。
  「○○である」の「○○」がサロゲートペアの場合，戻り値は，7 となります。

注意事項

ありません。

5.4.4 LEQ 関数

形式

```
LEQ '('string1,string2')'
```

説明

辞書式順序で，引数*string1* と*string2* が等しいかどうかを比較し，比較結果を返します。

引数

string1

文字列を値式で指定します。

string2

文字列を値式で指定します。

戻り値

- *string1* ＝*string2* の場合，1 を返します。
- *string1* ≠*string2* の場合，0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LEQ	*string1*	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書式順序の定義について説明します。

次の場合に，2 つの文字列が異なると判定します。

1. 両方の文字列に対して有効なインデックスに位置する文字が異なる場合
2. 文字列の長さが異なる場合
3. 1.と 2.の両方に該当する場合

なお，2 つの文字列が異なる場合で，異なる文字を指す最も小さいインデックスをk とします。位置k にある文字のCHAR 型の値を比較し「より小さい」値と判定される文字を持つ文字列が，辞書式順序でもう一方の文字列よりも前になります。

また，有効なインデックス位置での文字がすべて同じ場合，辞書式順序では短い方の文字列が前になります。

5.4.5　LGE 関数

形式

```
LGE '('string1,string2')'
```

説明

辞書式順序で，引数*string1* が*string2* より大きいかどうか，または引数*string1* と*string2* が等しいかどうかを比較し，比較結果を返します。

引数

string1

文字列を値式で指定します。

string2

文字列を値式で指定します。

戻り値

- *string1* ≧*string2* の場合，1 を返します。
- *string1*<*string2* の場合，0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LGE	*string1*	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書式順序の定義については，「5.4.4　LEQ 関数」の注意事項を参照してください。

5.4.6　LGT 関数

形式

```
LGT '('string1,string2')'
```

説明

辞書式順序で，引数*string1* が*string2* より大きいかどうかを比較し，比較結果を返します。

引数

string1

文字列を値式で指定します。

string2

文字列を値式で指定します。

戻り値

- *string1*>*string2* の場合，1 を返します。
- *string1* ≦*string2* の場合，0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LGT	*string1*	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書式順序の定義については，「5.4.4　LEQ 関数」の注意事項を参照してください。

5.4.7　LLE 関数

形式

```
LLE '('string1,string2')'
```

説明

辞書式順序で，引数*string1* が*string2* より小さいかどうか，または引数*string1* と*string2* が等しいかどうかを比較し，比較結果を返します。

引数

string1
　文字列を値式で指定します。

string2
　文字列を値式で指定します。

戻り値

string1 ≦*string2* の場合，1 を返します。

string1string2 の場合，0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LLE	*string1*	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書式順序の定義については，「5.4.4　LEQ 関数」の注意事項を参照してください。

5.4.8　LLT 関数

形式

```
LLT '('string1,string2')'
```

説明

辞書式順序で，引数*string1* が*string2* より小さいかどうかを比較し，比較結果を返します。

引数

string1
　文字列を値式で指定します。

string2

文字列を値式で指定します。

戻り値

- *string1*<*string2* の場合，1 を返します。
- *string1*≧*string2* の場合，0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
LLT	*string1*	CHAR	INTEGER
		VARCHAR	
	string2	CHAR	
		VARCHAR	

注意事項

辞書式順序の定義については，「5.4.4　LEQ 関数」の注意事項を参照してください。

5.4.9　REGEXP_FIRSTSEARCH 関数

形式

```
REGEXP_FIRSTSEARCH '('string,regexp')'
```

説明

引数*string* に指定された文字列を，引数*regexp* に指定した正規表現の文字列で検索し，最初に一致した文字列を返します。

引数

string

文字列を値式で指定します。

regexp

正規表現を定数で指定します。

戻り値

引数*string* の文字列を，引数*regexp* の正規表現の文字列で検索し，最初に一致した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_FIRSTSEARCH	*string*	CHAR	VARCHAR
		VARCHAR	
	regexp	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。

  REGEXP_FIRSTSEARCH('abcdefabc', '[d-z]*')の場合，'def'を返します。
- 正規表現で検索し，一致した文字列がない場合，空文字""を返します。

注意事項

- CQL 上では，正規表現を定数で指定してください。定数で指定しない場合，クエリの登録時にエラーとなります。

  (正しい指定例)

  文字列input.Str1 を正規表現'.A*'の定数で検索する場合

```
SELECT
 REGEXP_FIRSTSEARCH( input.Str1, '.A*' ) AS S1
 FROM input[ROWS 100];
```

  (誤った指定例)

  文字列input.Str1 を文字列input.Str2 の列名を指定して検索する場合

```
SELECT
 REGEXP_FIRSTSEARCH( input.Str1, input.Str2 ) AS S1
 FROM input[ROWS 100];FROM input[ROWS 100];
```

- 正規表現の構文に誤りがある場合，正規表現の解析ができないため，クエリグループが閉塞します。
- 正規表現の解析には，java.util.regex.Pattern クラスを使用します。このため，正規表現は，java.util.regex.Pattern クラスがサポートする正規表現の範囲で記述してください。

5.4.10 REGEXP_REPLACE 関数

形式

```
REGEXP_REPLACE '('string,regexp,replacement')'
```

説明

引数*string* に指定された文字列を，引数*regexp* に指定された正規表現の文字列で検索します。最初に一致した文字列を，引数*replacement* に指定された置換文字列に置き換えて返します。

引数

string

文字列を値式で指定します。

regexp

正規表現を定数で指定します。

replacement

置換文字列を値式で指定します。

戻り値

引数*string* の文字列を，引数*regexp* の正規表現の文字列で検索し，最初に一致した文字列を，引数*replacement* の置換文字列に置き換えて返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_REPLACE	*string*	CHAR	VARCHAR
		VARCHAR	
	regexp	CHAR	
		VARCHAR	
	replacement	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。
  REGEXP_REPLACE('abcdefabcdef', '[d-z]*', 'xyz')の場合，'abcxyzabcdef'を返します。
- 正規表現で検索し，一致した文字列がない場合，文字列string をそのまま返します。

注意事項

「5.4.9　REGEXP_FIRSTSEARCH 関数」の注意事項を参照してください。

5.4.11　REGEXP_SEARCH 関数

形式

```
REGEXP_SEARCH '('string,regexp')'
```

説明

引数*string* に指定された文字列を，引数*regexp* に指定された正規表現の文字列で検索し，パターンに一致する部分があるかどうかを返します。

引数

string

文字列を値式で指定します。

regexp

正規表現を定数で指定します。

戻り値

- 一致する文字列がある場合，1 を返します。
- 一致する文字列がない場合，0 を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
REGEXP_SEARCH	*string*	CHAR	INTEGER
		VARCHAR	
	regexp	CHAR	
		VARCHAR	

- 戻り値の例を次に示します。
  REGEXP_SEARCH('abcdefabc', '[d-z]*')の場合，1 を返します。

注意事項

「5.4.9　REGEXP_FIRSTSEARCH 関数」の注意事項を参照してください。

5.4.12　SPLENGTH 関数

形式

```
SPLENGTH ’(’string’)’
```

説明

引数*string* の文字数を返します。*string* 内にサロゲートペアが含まれている場合，サロゲートペア 1 文字を 1 文字として数えて結果を返します。

引数

string

文字列を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

文字列関数名	引数	引数のデータ型	戻り値のデータ型
SPLENGTH	*string*	CHAR	INTEGER
		VARCHAR	

- 「は＋゛（濁点）」→「ば゛」，「は＋゜（半濁点）」→「ぱ゜」のように 1 字を表す結合文字については，2 文字として結果を返します。
  結合文字：「ば゛すけっと」＝ 6 文字
  非結合文字：「ばすけっと」＝ 5 文字
- 文字列にサロゲートペアが含まれている場合，サロゲートペア 1 文字を，1 文字として数えて結果を返します。
  「○○である」の「○○」がサロゲートペアの場合，戻り値は，5 となります。

注意事項

ありません。

5.5　時刻関数の詳細

ここでは，「4.4.27　組み込みスカラ関数」で指定する時刻関数について説明します。

5.5.1　時刻関数

形式

```
時刻関数::=時刻関数名’(’値式 ,値式’)’
```

説明

時刻関数は，日付時刻型の入力値に対して演算を実行し，演算結果を数値で返します。

オペランド

時刻関数名

時刻関数の名称です。時刻関数の一覧を次に示します。

表 5-6　時刻関数の一覧

関数名	説明
TIMEDIFF 関数	TIME 型の時刻の差分を計算し，結果をミリ秒の数値で返します。
TIMESTAMPDIFF 関数	TIMESTAMP 型の時刻の差分を計算し，結果をミリ秒の数値で返します。

値式

値式については，「4.4.18　値式」を参照してください。なお，値式の個数は関数によって異なります。時刻関数の各関数の説明を参照してください。

注意事項

- タイムゾーンは，Streaming Data Platform が動作している JavaVM に従います。
- 時刻関数は，夏時間に対応していません。夏時間を実施するタイムゾーンで使用した場合でも，日付データ，時刻データ，時刻印データは，標準時間に従って扱います。

使用例

TIME 型の時刻データ S1.DATETIME が 01:00:00 以前のタプルだけを出力します。

```
register query FILTER ISTREAM (SELECT * FROM S1[ROWS 10]
  WHERE TIMEDIFF('01:00:00',S1.DATETIME) < 0 );
```

5.5.2 TIMEDIFF 関数

形式

```
TIMEDIFF'('time1,time2')'
```

説明

引数*time1* と*time2* の差分を計算し，結果をミリ秒の数値で返します。

引数

time1

時刻データを値式で指定します。

time2

時刻データを値式で指定します。

戻り値

引数*time1*，*time2* を 1970 年 1 月 1 日 00:00:00 GMT からのミリ秒に変換し，「*time2*－*time1* ＝*差分*」で計算した値を返します。単位はミリ秒です。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

時刻関数名	引数	引数のデータ型	戻り値のデータ型
TIMEDIFF	*time1*	TIME	BIGINT
	time2	TIME	

TIME 型は，java.sql.Time 型に対応するため，引数の時刻データは 1970 年 1 月 1 日 00:00:00 GMT が基準となります。引数に日付の繰り上がりがある時刻データを指定した場合，結果の差分は繰り上がった日付を含んだミリ秒の数値を返します。

例えば，*time1* に「09:00:00」，*time2* に「33:00:00」を指定した場合，*time1* は「1970-01-01 09:00:00」，*time2* は「1970-01-02 09:00:00」となり，*time2* の日付が繰り上がります。また，戻り値は，86400000 となります。下線部が，日付の部分です。

注意事項

入力値によっては，想定しない値が返る場合があります。

例えば，TIME1='00:00:00'（時刻印データで表すと1970-01-01 00:00:00.000），および BIGINT1=253402236000000 をTIME 型に変換した値を入力した場合，TIMEDIFF(TIME1,NUMBER_TOTIME(BIGINT1))は，戻り値253402268400000 のミリ秒数値を返します。戻り値253402268400000 を時刻印データで表すと，'10000-01-01 00:00:00.000'となります。

5.5.3　TIMESTAMPDIFF 関数

形式

```
TIMESTAMPDIFF'('timestamp1,timestamp2')'
```

説明

引数 *timestamp1* と *timestamp2* の差分を計算し，結果をミリ秒の数値で返します。

引数

timestamp1

時刻印データを値式で指定します。

timestamp2

時刻印データを値式で指定します。

戻り値

引数 *timestamp1*，*timestamp2* を 1970 年 1 月 1 日 00:00:00 GMT からのミリ秒に変換し，「*timestamp2－timestamp1＝差分*」で計算した値を返します。単位はミリ秒です。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

時刻関数名	引数	引数のデータ型	戻り値のデータ型
TIMESTAMPDIFF	*timestamp1*	TIMESTAMP	BIGINT
	timestamp2	TIMESTAMP	

注意事項

「5.5.2　TIMEDIFF 関数」の注意事項を参照してください。

5.6　変換関数の詳細

ここでは，「4.4.27　組み込みスカラ関数」で指定する変換関数について説明します。

5.6.1　変換関数

形式

変換関数::=*変換関数名*’(’*値式*’)’

説明

変換関数は，引数のデータ型を別のデータ型に変換する機能を提供します。

オペランド

変換関数名

変換関数の名称です。変換関数の一覧を次に示します。

表 5-7　変換関数の一覧

関数名	説明
BIGINT_TOSTRING 関数	BIGINT 型の数値を文字列型に変換します。
DATE_TONUMBER 関数	DATE 型の日付データをミリ秒の数値に変換します。
DECIMAL_TOSTRING 関数	DECIMAL 型またはNUMERIC 型の数値を文字列型に変換します。
DOUBLE_TOSTRING 関数	DOUBLE 型またはFLOAT 型の数値を文字列型に変換します。
INT_TOSTRING 関数	INTEGER 型の数値を文字列型に変換します。
NUMBER_TODATE 関数	ミリ秒の数値をDATE 型の日付データに変換します。
NUMBER_TOTIME 関数	ミリ秒の数値をTIME 型の時刻データに変換します。
NUMBER_TOTIMESTAMP 関数	ミリ秒の数値をTIMESTAMP 型の時刻印データに変換します。
REAL_TOSTRING 関数	REAL 型の数値を文字列型に変換します。
STRING_TOBIGINT 関数	文字列型をBIGINT 型の数値に変換します。
STRING_TODECIMAL 関数	文字列型をDECIMAL 型の数値に変換します。
STRING_TODOUBLE 関数	文字列型をDOUBLE 型の数値に変換します。
STRING_TOINT 関数	文字列型をINTEGER 型の数値に変換します。
STRING_TOREAL 関数	文字列型をREAL 型の数値に変換します。
TIME_TONUMBER 関数	TIME 型の時刻データをミリ秒の数値に変換します。

関数名	説明
TIMESTAMP_TONUMBER 関数	TIMESTAMP 型の時刻印データをミリ秒の数値に変換します。

値式

値式については，「4.4.18　値式」を参照してください。なお，値式の個数は関数によって異なります。変換関数の各関数の説明を参照してください。

注意事項

- 変換関数の引数・戻り値での日付データ，時刻データ，時刻印データは，Streaming Data Platform が動作している JavaVM のタイムゾーンに従います。また，変換関数の引数・戻り値でのミリ秒の数値は，1970 年 1 月 1 日 00:00:00 GMT を基準とした値です。
- 時刻を扱う変換関数は，夏時間に対応していません。夏時間を実施するタイムゾーンで使用した場合でも，日付データ，時刻データ，時刻印データは，標準時間に従って扱います。
- 変換関数で数値に変換できる文字データの書式を次に示します。

データ型	文字データの書式	
	規則	例
INTEGER BIGINT	• 0～9 の数字を指定します。 • ピリオドを含んではいけません。 • 変換先の範囲内に値を収める必要があります。 • 正の値の場合，プラス(+)の符号を指定できません。	-123 45 6789
DOUBLE REAL DECIMAL	• 0～9 の数字を指定します。 • ピリオドを含められます。 • 変換先の範囲内に値を収める必要があります。 • 値にプラス(+)またはマイナス(-)の符号を指定できます。	-12.3 456 +0.789

使用例

INTEGER 型のS1.CODE を文字列型に変換し出力します。

```
register query FILTER ISTREAM ( SELECT INT_TOSTRING(S1.CODE) FROM S1[ROWS 10] );
```

S1.CODE が34512 の場合，34512 を返します。

5.6.2　BIGINT_TOSTRING 関数

形式

```
BIGINT_TOSTRING '('value')'
```

説明

BIGINT 型の数値 *value* を文字列型に変換します。

引数

value

BIGINT 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し，変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
BIGINT_TOSTRING	*value*	BIGINT	VARCHAR

注意事項

ありません。

5.6.3 DATE_TONUMBER 関数

形式

```
DATE_TONUMBER '('date')'
```

説明

DATE_TONUMBER 関数は，DATE 型の *date* をミリ秒の数値に変換します。このとき，時刻は00:00:00 として変換されます。

引数

date

DATE 型の数値を値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
DATE_TONUMBER	*date*	DATE	BIGINT

注意事項

詳細については，「5.5.2　TIMEDIFF 関数」の注意事項を参照してください。

5.6.4　DECIMAL_TOSTRING 関数

形式

```
DECIMAL_TOSTRING '('value')'
```

説明

DECIMAL 型またはNUMERIC 型の数値 *value* を文字列型に変換します。

引数

value

DECIMAL 型またはNUMERIC 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し，変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
DECIMAL_TOSTRING	*value*	DECIMAL	VARCHAR
		NUMERIC	

- 数値が 10^{-6} 未満の場合，指数表現を使用した文字形式に変換します。
  （例）
  入力数値「0.00000100121」 → 変換後文字列「0.00000100121」
  入力数値「0.000000100121」 → 変換後文字列「1.00121E-7」

注意事項

DECIMAL 型（NUMERIC 型）以外のデータ型からDECIMAL 型（NUMERIC 型）へのキャスト（暗黙的なキャストを含む）では，結果の桁数がquery.decimalMaxPrecision パラメーターで指定した桁数を超えた場合，指

定した桁数に丸められます。引数は丸められた数値となるため，DECIMAL_TOSTRING 関数も丸められた数値を変換した文字列を返します。

（例）

query.decimalMaxPrecision=5 の場合は，次のようになります。

- DECIMAL_TOSTRING((DECIMAL)DBL1)
  入力数値（DOUBLE 型の DBL1）「23.45666」 → 変換後文字列「23.457」
- DECIMAL_TOSTRING(DEC1)
  入力数値（DECIMAL 型の DEC1）「23.45666」 → 変換後文字列「23.45666」

5.6.5 DOUBLE_TOSTRING 関数

形式

```
DOUBLE_TOSTRING '('value')'
```

説明

DOUBLE 型またはFLOAT 型の数値 *value* を文字列型に変換します。

引数

value

DOUBLE 型またはFLOAT 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し，変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
DOUBLE_TOSTRING	*value*	DOUBLE	VARCHAR
		FLOAT	

- 数値が 10^{-3} 未満の場合，または 10^{7} 以上の場合，指数表現を使用した文字形式に変換します。
  （例）
  入力数値「2768237.5693」 → 変換後文字列「2768237.5693」
  入力数値「27682371.5693」 → 変換後文字列「2.76823715693E7」
  入力数値「0.005693442」 → 変換後文字列「0.005693442」

入力数値「0.0005693442」 → 変換後文字列「5.693442E-4」

注意事項

浮動小数点型は，IEEE 754 に準拠しているため，正確に変換できない場合があります。

5.6.6 INT_TOSTRING 関数

形式

```
INT_TOSTRING '('value')'
```

説明

INTEGER 型の数値 *value* を文字列型に変換します。

引数

value

INTEGER 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し，変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
INT_TOSTRING	*value*	INTEGER	VARCHAR

注意事項

ありません。

5.6.7 NUMBER_TODATE 関数

形式

```
NUMBER_TODATE '('value')'
```

説明

ミリ秒の数値 *value* をDATE 型の日付データに変換します。

引数

value

ミリ秒の数値（BIGINT 型）を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
NUMBER_TODATE	*value*	BIGINT	DATE

注意事項

「5.6.8　NUMBER_TOTIME 関数」の注意事項を参照してください。

5.6.8　NUMBER_TOTIME 関数

形式

```
NUMBER_TOTIME '('value')'
```

説明

ミリ秒の数値 *value* をTIME 型の時刻データに変換します。

引数

value

ミリ秒の数値（BIGINT 型）を値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
NUMBER_TOTIME	*value*	BIGINT	TIME

注意事項

入力値によっては，想定しない値が返る場合があります。

例えば，BIGINT1=9223372036854775807 を入力した場合，NUMBER_TOTIME(BIGINT1)は，戻り値16:12:55 を返します。戻り値16:12:55 は，時刻印データで表すと292278994-08-17 16:12:55.807 となります。

5.6.9　NUMBER_TOTIMESTAMP 関数

形式

```
NUMBER_TOTIMESTAMP '('value')'
```

説明

NUMBER_TOTIMESTAMP 関数は，ミリ秒の数値 *value* をTIMESTAMP 型の時刻印データに変換します。

引数

value

ミリ秒の数値（BIGINT 型）を値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
NUMBER_TOTIMESTAMP	*value*	BIGINT	TIMESTAMP

注意事項

詳細については，「5.6.8　NUMBER_TOTIME 関数」の注意事項を参照してください。

5.6.10　REAL_TOSTRING 関数

形式

```
REAL_TOSTRING '('value')'
```

説明

REAL 型の数値 *value* を文字列型に変換します。

引数

value

REAL 型の数値を値式で指定します。

戻り値

数値を文字列型に変換し，変換した文字列を返します。

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
REAL_TOSTRING	*value*	REAL	VARCHAR

- 数値が 10^{-3} 未満の場合，または 10^{7} 以上の場合，指数表現を使用した文字形式に変換します。
  （例）
  入力数値「2768237.5693」 → 変換後文字列「2768237.5」
  入力数値「27682371.5693」 → 変換後文字列「2.7682372E7」
  入力数値「0.005693442」 → 変換後文字列「0.005693442」
  入力数値「0.0005693442」 → 変換後文字列「5.693442E-4」

注意事項

浮動小数点型は，IEEE754 に準拠しているため，正確に変換できない場合があります。

5.6.11 STRING_TOBIGINT 関数

形式

```
STRING_TOBIGINT '('value')'
```

説明

STRING_TOBIGINT 関数は，文字データ *value* を，BIGINT データ型の数値データに変換します。

引数

value

文字データを値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
STRING_TOBIGINT	*value*	CHAR	BIGINT
		VARCHAR	

注意事項

引数に誤りがあるために引数をBIGINT データ型に変換できない場合，または値に変換される引数がBIGINT データ型の有効範囲外の場合，クエリグループはブロックされます。

5.6.12 STRING_TODECIMAL 関数

形式

```
STRING_TODECIMAL '('value')'
```

説明

STRING_TODECIMAL 関数は，文字データ *value* を，DECIMAL データ型の数値データに変換します。

引数

value

文字データを値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
STRING_TODECIMAL	*value*	CHAR	DECIMAL
		VARCHAR	

注意事項

- 引数に誤りがあるために引数をDECIMAL データ型に変換できない場合，または値に変換される引数がDECIMAL データ型の有効範囲外の場合，クエリグループはブロックされます。

- 変換された値の桁数がquery.decimalMaxPrecision引数で指定した桁数を超えた場合，戻り値は指定した桁数に切り上げられます。この値は，query.decimalRoundingMode引数で指定された方法に基づいて切り上げられます。

5.6.13 STRING_TODOUBLE関数

形式

```
STRING_TODOUBLE '('value')'
```

説明

STRING_TODOUBLE関数は，文字データ*value*を，DOUBLEデータ型の数値データに変換します。

引数

value

文字データを値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
STRING_TODOUBLE	*value*	CHAR	DOUBLE
		VARCHAR	

注意事項

- 引数に誤りがあるために引数をDOUBLEデータ型に変換できない場合，または値に変換される引数がDOUBLEデータ型の有効範囲外の場合，クエリグループはブロックされます。
- 浮動小数点データ型は，IEEE 754規格に準拠しているため，正確に変換されない場合があります。

5.6.14 STRING_TOINT関数

形式

```
STRING_TOINT '('value')'
```

説明

STRING_TOINT 関数は，文字データ *value* を，INT データ型の数値データに変換します。

引数

value

文字データを値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
STRING_TOINT	*value*	CHAR	INT
		VARCHAR	

注意事項

引数に誤りがあるために引数をINT データ型に変換できない場合，または値に変換される引数がINT データ型の有効範囲外の場合，クエリグループはブロックされます。

5.6.15 STRING_TOREAL 関数

形式

```
STRING_TOREAL '('value')'
```

説明

STRING_TOREAL 関数は，文字データ *value* を，REAL データ型の数値データに変換します。

引数

value

文字データを値式で指定します。

戻り値

次の表に，引数のデータ型と返される値の関係を示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
STRING_TOREAL	*value*	CHAR	REAL

変換関数名	引数	引数のデータ型	戻り値のデータ型
STRING_TOREAL	*value*	VARCHAR	REAL

注意事項

- 引数に誤りがあるために引数をREAL データ型に変換できない場合，または値に変換される引数がREAL データ型の有効範囲外の場合，クエリグループはブロックされます。
- 浮動小数点データ型は，IEEE 754 規格に準拠しているため，正確に変換されない場合があります。

5.6.16　TIME_TONUMBER 関数

形式

```
TIME_TONUMBER '('time')'
```

説明

TIME 型の時刻データ *time* をミリ秒の数値に変換します。

引数

time

TIME 型の時刻データを値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
TIME_TONUMBER	*time*	TIME	BIGINT

注意事項

「5.5.2　TIMEDIFF 関数」の注意事項を参照してください。

5.6.17　TIMESTAMP_TONUMBER 関数

形式

```
TIMESTAMP_TONUMBER '('timestamp')'
```

説明

TIMESTAMP 型の時刻印データ *timestamp* をミリ秒の数値に変換します。

引数

timestamp

TIMESTAMP 型の時刻印データを値式で指定します。

戻り値

引数のデータ型と戻り値のデータ型の関係を次の表に示します。

変換関数名	引数	引数のデータ型	戻り値のデータ型
TIMESTAMP_TONUMBER	*timestamp*	TIMESTAMP	BIGINT

注意事項

「5.5.2　TIMEDIFF 関数」の注意事項を参照してください。

6 CQL デバッグツールの操作

この章では，クエリのデバッグ，テスト結果の作成，および CQL のチューニングで使用する開発支援ツールについて説明します。

6.1　概要

CQL デバッグツールは開発支援ツールであり，単純なデバッグ，テスト結果の作成，およびクエリのチューニングをするために使用できます。ツールを使用することで，初期開発コストの削減を実現できます。

クエリ開発フェーズでは，クエリをテストするために用意されているテストデータを入力できます。システム稼働フェーズでは，Hitachi Streaming Data Platform からテストデータとして収集されたタプルログを入力することによって，運用システムで発生した問題を再現できます。

CQL デバッグツールは，クエリ検証機能（hsdpcqldebug コマンド）でクエリを検証します。

6.1.1　CQL デバッグツールを構成するコマンドとファイル

Hitachi Streaming Data Platform software development kit は，クエリ検証機能を提供します。この機能は入力データとクエリグループを分析し，検証結果を出力します。この機能は，CQL 構文，文法（コンパイルエラーのチェック）および CQL ランタイムエラー（ランタイムエラーのチェック）を検証します。この機能は，hsdpcqldebug コマンドを実行することで使用できます。

CQL デバッグツールを構成するコマンドとファイルを次の表に示します。

表 6-1　CQL デバッグツールを構成するコマンドとファイル

項番	インタフェース	説明	参考
1	hsdpcqldebug コマンド	クエリの検証に使用します。	「16.3　hsdpcqldebug」
2	入力データファイル	クエリの検証でクエリに入力するデータは，入力データファイルで指定します。	「6.1.1　CQL デバッグツールを構成するコマンドとファイル」の「(1)　入力データファイル」
3	出力結果ファイル	クエリの検証結果は，出力結果ファイルに出力されます。	「6.1.1　CQL デバッグツールを構成するコマンドとファイル」の「(2)　出力結果ファイル」

(1)　入力データファイル

説明

クエリを検証するために使用する入力データは，入力データファイルで指定します。各入力データファイルは，入力ストリームに対応します。このファイルは，CQL デバッグツールを使用するユーザーが事前に準備します。

hsdpcqldebug コマンドの引数に指定したクエリグループ用プロパティファイルのquerygroup.encoding パラメーターに指定したエンコードで入力データファイルが読み込まれます。このため，入力データファイルのエンコードとquerygroup.encoding パラメーターには，同じ値を指定してください。入力データファ

イルのエンコードとquerygroup.encodingパラメーターに指定した値が異なる場合，デコードできない文字が文字化けします。

ファイル名

入力ストリーム名※.csv

注※

ファイル名の*入力ストリーム名*は，大文字で指定します。クエリ定義ファイルでは，入力ストリーム名に同じ名前（大文字と小文字は区別されません）を指定する必要があります。

ファイル格納場所

入力データファイルは，任意のディレクトリに格納できます。

形式

項目	説明
ファイル形式	入力データファイルのファイル形式はCSV形式です。各入力データファイルには，1行につき1タプルのデータが含まれています。各行のサイズは，32KB未満にしてください。
コメント	シャープ記号（#）で始まる行は，コメント行と見なされます。コメントは，各行の行頭だけに指定できます。
スキーマ定義 （任意）	入力データのスキーマ定義は，最初の行に定義する必要があります。スキーマ定義は，（行頭の）##と（コンマで区切られた）Javaデータ型で構成されます。タプルに設定するシステム時刻は，先頭行の最初の要素として__systemtime__を記述し，続けてスキーマ，コンマで区切られたJavaデータ型を列挙することで指定できます。 スキーマ定義が指定されていない場合は，クエリ定義で定義されているスキーマが使用されます。クエリ定義で定義されているスキーマが使用されている場合，タプルに設定されているシステム時刻はレコードに指定できません。指示文EXTERNAL C BEGINおよびEXTERNAL C ENDがクエリ定義ファイルで指定されている場合，次のJavaデータ型は指定できません。 • BigDecimal • Date • Time 入力データに設定されているスキーマ定義では，データ型の名前は大文字と小文字を区別しません。使用できるJavaデータ型は，次のとおりです。 • Integer • Double • String • Date • Time • Timestamp • BigDecimal • Byte • Short • Long

項目	説明
スキーマ定義 （任意）	• Float
空白行	文字のない行は処理されません。
特別な値の処理	スペースとタブ スペースとタブは，そのまま処理する必要があります。 改行 CSV ファイルでは，改行は区切り文字と見なされます。改行を列内で改行として処理したい場合は，その列を（改行を含めて）二重引用符で囲む必要があります。 ,（コンマ） レコードでは，コンマは区切り文字と見なされます。コンマをレコード内でコンマとして処理したい場合は，その列を（コンマを含めて）二重引用符で囲む必要があります。 #（シャープ記号） 行の先頭にシャープ記号（#）を置くと，その行はコメント行と見なされます。 シャープ記号（#）を列内で#として処理したい場合は，その列を（#を含めて）二重引用符で囲む必要があります。 ”（二重引用符） 二重引用符で囲まれた範囲は 1 つの列と見なされます。 二重引用符をレコード内で二重引用符として処理したい場合は，その列を（二重引用符を含めて）二重引用符で囲む必要があります。
システム時刻	次のシナリオでは，タプルに設定する必要があるシステム時刻を各レコードの最初の値として指定する必要があります。 • 対象のクエリグループのタイムスタンプモードが，サーバモードである場合 • 対象のクエリグループが，カスケーディングアダプターを使用している場合 システム時刻は，次の形式で指定する必要があります。 *yyyy-MM-ddhh:mm:ss*[*.nn…n*] • *yyyy*：0001～9999（年） • *MM*：01～12（月） • *dd*：01～その月の最終日（日） • *hh*：00～23（時間） • *mm*：00～59（分） • *ss*：00～59（秒） • *nn...n*：1～9 桁の小数秒（*n*：0～9） 対象のクエリグループのタイムスタンプモードがデータソースモードであり，カスケーディングアダプターが使用されていない場合も，各レコードの最初の値としてシステム時刻を指定できます（ただし，この値は無視されます）。各レコードの最初の値としてシステム時刻を指定する場合は，先頭行にスキーマ定義を記述します。

例

入力ストリーム名がIN1 とIN2 の入力データファイルの例は，次のとおりです。

入力データファイル：IN1.csv

```
## Integer,Timestamp,String,Double
001,2013-09-09 10:00:00.000,"Washington, D.C.",20.6
002,2013-09-09 10:00:00.513,New York,18.2
003,2013-09-09 10:00:01.048, Miami,29.2
004,2013-09-09 10:00:01.628,Houston,27.6
#005,2013-09-09 10:00:02.193,Chicago,17.4
006,2013-09-09 10:00:02.871,Oklahoma City,23.1
```

入力データファイル：IN2.csv

```
## __systemtime__,Timestamp,String,Long
2013-09-10 10:00:00.0,2013-09-10 10:00:00.0,"""Los Angeles, California""",77
2013-09-10 10:00:01.0,2013-09-10 10:00:01.0,"""Honolulu, Hawaii""",61
2013-09-10 10:00:02.0,2013-09-10 10:00:02.0,"""Columbus, Ohio""",55
```

(2) 出力結果ファイル

説明

hsdpcqldebug コマンドを実行すると，クエリの実行結果が出力結果ファイルに出力されます。各出力ストリームに対して，出力結果ファイルが生成されます。この出力結果ファイルは，入力データファイルとしても使用できます。分割された分析シナリオをテストする場合は，出力ストリームと同じスキーマ定義を持つ入力ストリームの入力データファイルとして出力結果ファイルを使用します。

hsdpcqldebug コマンドの引数に指定したクエリグループ用プロパティファイルのquerygroup.encoding パラメーターに指定したエンコードの設定で結果ファイルが出力されます。

ファイル名

出力ストリーム名※.csv

注※

*出力ストリーム名*のファイル名は，大文字で指定します。クエリ定義ファイルでは，出力ストリーム名に同じ名前を指定する必要があります。

ファイル格納場所

出力結果ファイルは，任意のディレクトリに格納できます。

形式

項目	説明
ファイル形式	クエリ結果が出力される出力結果ファイルは CSV 形式です。各行はタプルを表します。
スキーマ定義	スキーマ定義は先頭行に出力されます。スキーマ定義には，次の要素があります。 • 先頭の## • __systemtime__

項目	説明
スキーマ定義	• 各列に対応する Java データ型 __systemtime__と各 Java データ型はコンマで区切られます。
システム時刻	システム時刻は，各行の最初の値として出力されます。 *yyyy-MM-ddhh:mm:ss.nnnnnnnnn* • *yyyy*：0001～9999（年） • *MM*：01～12（月） • *dd*：01～その月の最終日（日） • *hh*：00～23（時間） • *mm*：00～59（分） • *ss*：00～59（秒） • *nnnnnnnnn*：秒数以降の 9 桁の 10 進数
String 型の列	String 型の列は，出力時には二重引用符で囲まれます。列内の二重引用符は，2 つの二重引用符に変換されます。

例

この例では，「6.1.1　CQL デバッグツールを構成するコマンドとファイル」の「(1)　入力データファイル」で例として使用したファイルを入力データファイルとして使用しています。出力ストリーム名がOUT1とOUT2 の出力データファイルの例は，次のとおりです。

出力結果ファイル：OUT1.csv

入力データファイルとしてIN1.csv を指定し，4 番目の列の値が 24 未満のタプルを選択して出力するクエリを指定してhsdpcqldebug コマンドを実行すると，次の出力結果ファイルが生成されます。

```
## __systemtime__,Integer,Timestamp,String,Double
2013-09-09 10:00:00.000000000,1,2013-09-09 10:00:00.0,"Washington, D.C.",20.6
2013-09-09 10:00:00.513000000,2,2013-09-09 10:00:00.513,"New York",18.2
2013-09-09 10:00:02.871000000,6,2013-09-09 10:00:02.871,"Oklahoma City",23.1
```

出力結果ファイル：OUT2.csv

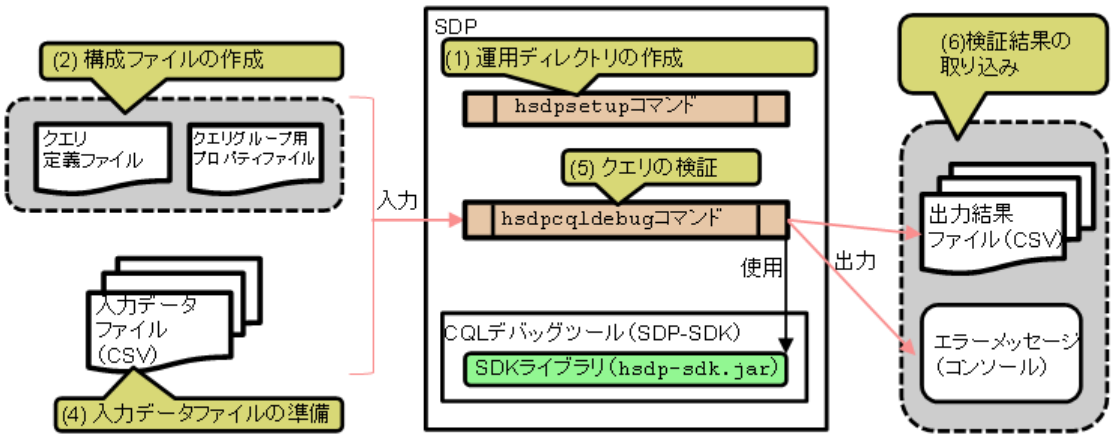
入力データファイルとしてIN2.csv を指定し，3 番目の列の値を 10 で割った結果を出力するクエリを指定してhsdpcqldebug コマンドを実行すると，次の出力結果ファイルが生成されます。

```
## __systemtime__,Timestamp,String,Long
2013-09-10 10:00:00.000000000,2013-09-10 10:00:00.0,"""Los Angeles, California""",7
2013-09-10 10:00:01.000000000,2013-09-10 10:00:01.0,"""Honolulu, Hawaii""",6
2013-09-10 10:00:02.000000000,2013-09-10 10:00:02.0,"""Columbus, Ohio""",5
```

6.1.2　CQL デバッグツールの使用例（手順の流れ）

CQL デバッグツールを使用した手順の流れの例を説明します。

CQL デバッグツールを使用した手順の流れ



説明

CQL デバッグツールを使用する手順（流れ）は次のとおりです。

1. 運用ディレクトリの作成

   開発フェーズで使用する運用ディレクトリを，hsdpsetup コマンドで作成します。

   運用ディレクトリの作成の詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

2. 構成ファイル（クエリ定義ファイルとクエリグループ用プロパティファイル）の作成

   開発者は，クエリ定義ファイルとクエリグループ用プロパティファイルを作成する必要があります。

   クエリ定義ファイルとクエリグループ用プロパティファイルの作成の詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

3. 外部定義関数および外部定義関数定義ファイルの作成（任意）

   開発者がクエリで外部定義関数を使用する場合は，外部定義関数および外部定義関数定義ファイルを作成する必要があります。

   > **メモ**
   >
   > 外部定義関数および外部定義関数定義ファイルを作成する手順は，図では説明していません。

   外部定義関数定義ファイルの作成の詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

4. 入力データファイルの準備

   開発者は，クエリの入力データとして入力データファイルを準備する必要があります。準備した入力データファイルは，任意の場所に格納できます。

   入力データファイルの準備の詳細については，「6.1.1　CQL デバッグツールを構成するコマンドとファイル」の「(1)　入力データファイル」を参照してください。

5. クエリの検証

hsdpcqldebug コマンドを使用して，作成した入力データを入力し，作成したクエリを実行して実行可能かどうかを確認します。hsdpcqldebug コマンドを使用してクエリを実行した場合，エラーが発生しなければ，結果が出力結果ファイルに出力されます。hsdpcqldebug コマンドを使用してクエリを実行した場合，エラーが発生すると，エラーメッセージがコンソールに出力されます。hsdpcqldebug コマンドの詳細については，「16.3　hsdpcqldebug」を参照してください。

クエリの検証例

```
$ hsdpcqldebug テストCQLグループ -i ./入力 -o ./出力
```

6. 検証結果の取り込み

エラーメッセージまたは予期しない結果が出力された場合は，クエリ定義ファイルまたは入力データファイルを修正し，hsdpcqldebug コマンドを再度実行してください。

7 カスタムアダプターの作成

この章では，カスタムアダプターの作成方法について説明します。

カスタムアダプターは，標準提供アダプターで扱えない形式のデータを扱う場合などに，必要に応じて作成してください。

なお，カスタムアダプターの作成で使用する API の詳細については，「8.　データ送受信用 API」を参照してください。

7.1　作成するカスタムアダプターの種類

この節では，作成するカスタムアダプターの種類について説明します。

カスタムアダプターとは，入出力データと SDP サーバの間でデータをやり取りする機能を持つアプリケーションです。Hitachi Streaming Data Platform が提供する API を使用して作成します。

カスタムアダプターは，処理内容によって，データ送信 AP とデータ受信 AP の 2 種類に分類できます。

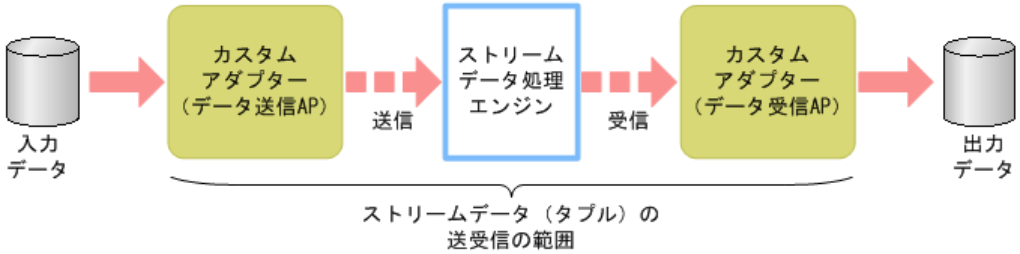
7.1.1　データ送信 AP とデータ受信 AP

カスタムアダプターは，処理内容ごとに，次の 2 種類に分類できます。

- **データ送信 AP**
  入力データを受け付け，SDP サーバに対してストリームデータを送信します。
- **データ受信 AP**
  SDP サーバで分析した結果（クエリ結果）のストリームデータを受信します。

カスタムアダプターの位置づけについて，次の図に示します。

図 7-1　カスタムアダプターの位置づけ



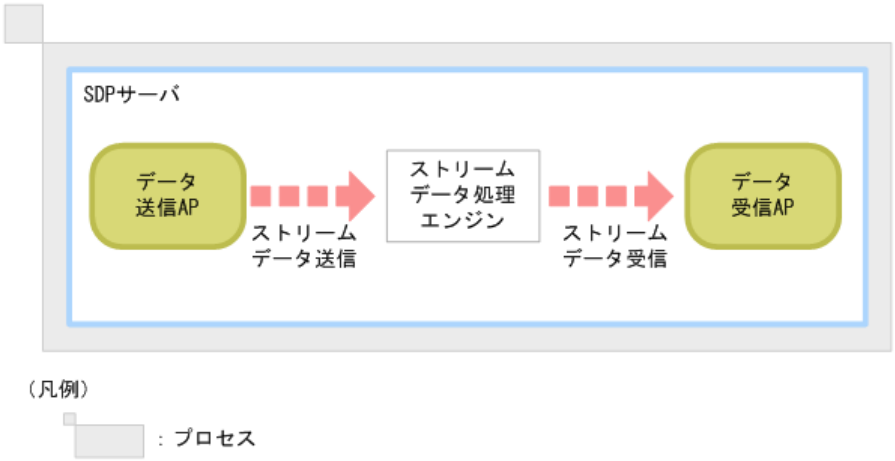
データ送信 AP では，任意の形式のデータを受け付け，ストリームデータとしてストリームデータ処理エンジンに送信します。データ受信 AP では，ストリームデータ処理エンジンで処理されたストリームデータを受信して，任意の形式のデータとして出力します。

7.1.2　インプロセス連携カスタムアダプター

SDP サーバとのデータの送受信を SDP サーバのプロセス内で実行します。カスタムアダプターは，SDP サーバの一部として動作します。

インプロセス連携カスタムアダプターを使用する場合のストリームデータの流れを次の図に示します。

図 7-2　インプロセス連携カスタムアダプター



7.2 インプロセス連携カスタムアダプターの作成

この節では，インプロセス連携カスタムアダプターの作成方法について説明します。

インプロセス連携カスタムアダプターの実装では，StreamInprocessUP インタフェースを実装したクラスを作成する必要があります。

インプロセス連携カスタムアダプターは，SDP サーバで生成したSDPConnector 型オブジェクトを，StreamInprocessUP インタフェースの実装クラスのexecute メソッドのパラメーターとして受け取ります。execute メソッドには，カスタムアダプターのmain メソッドに相当する処理を記述します。

ストリームデータの送受信には，StreamInput インタフェースおよびStreamOutput インタフェースを使用します。StreamInput 型オブジェクトおよびStreamOutput 型オブジェクトは，execute メソッドのパラメーターとして渡されたSDPConnector 型オブジェクトのメソッドによって生成します。

ストリームデータの送受信時に使用するインタフェースについて，概要を示します。

ストリームデータ送信時

ストリームデータの送信には，StreamInput **インタフェース**を使用します。

カスタムアダプターでは，execute メソッドのパラメーターとして渡されたSDPConnector 型オブジェクトを使用して，StreamInput 型オブジェクトを生成します。生成したStreamInput 型オブジェクトのput メソッドを使用して，データを送信します。

ストリームデータ受信時

ストリームデータの受信には，StreamOutput **インタフェース**を使用します。

カスタムアダプターでは，execute メソッドのパラメーターとして渡されたSDPConnector 型オブジェクトを使用して，StreamOutput 型オブジェクトを生成します。生成したStreamOutput 型オブジェクトを使用して，次のどちらかの方式で SDP サーバからクエリ結果のストリームデータ（タプル）を取得します。

- **ポーリング方式**

  SDP サーバから能動的にタプルを取得する，レイテンシ重視の方式です。get メソッドまたはgetAll メソッドを使用して，タプルを受信します。

- **コールバック方式**

  SDP サーバ上でタプルが生成されたときに，タプルを取得するためのメソッドが SDP サーバによって呼び出される方式です。この方式を使用する場合，SDP サーバに呼び出されるメソッドは，事前に登録しておく必要があります。ポーリング方式と比較してレイテンシは増えますが，結果が生成された場合だけ処理するので CPU 効率を重視した方式です。

なお，インプロセス連携カスタムアダプターを使用する場合は，user_app.*＜インプロセス連携AP名＞*.properties に次の記述が必要になります。

```
user_app.classname=カスタムアダプターのメインクラス名
user_app.classpath_dir=カスタムアダプターのメインクラスのパス名
```

user_app.＜インプロセス連携AP名＞.properties の指定方法については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

7.2.1　ストリームデータの送信（インプロセス連携カスタムアダプター）

ここでは，インプロセス連携カスタムアダプターでストリームデータを送信する処理の基本的な流れについて，実装例を基に説明します。

実装例

```
public class Inpro_SendSample implements StreamInprocessUP {
  // StreamInprocessUPの実装
  public void execute(SDPConnector con) {
    // 1. 送信対象の入力ストリームに接続
    StreamInput in = con.openStreamOutput(”GROUP”,”STREAM1”);

    // 2. タプルを送信
    Object[] data=new Object[]{new Integer(1)};
    StreamTuple tuple=new StreamTuple(data);
    in.put(tuple);

    // 3. データ送信完了を通知
    in.putEnd();

    // 4. 入力ストリームとの接続を切断
    in.close();

    // 5. SDPサーバとの接続を切断
    con.close();
  }
}
```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

1. execute メソッドのパラメーターとして渡されたSDPConnector 型オブジェクト（con）を使用して，クエリグループ名が”GROUP”，ストリーム名が”STREAM1”の入力ストリームへ接続し，StreamInput 型オブジェクト（in）を取得します。

   ```
   StreamInput in = con.openStreamInput(”GROUP”,”STREAM1”);
   ```

2. StreamInput 型オブジェクト（in）を使用して，タプルを送信します。

   ```
   in.put(tuple);
   ```

3. StreamInput 型オブジェクト（in）を使用して，データ送信完了を通知します。

   ```
   in.putEnd();
   ```

4. StreamInput 型オブジェクト（in）を使用して，入力ストリームとの接続を切断します。

   ```
   in.close();
   ```

5. SDPConnector 型オブジェクトを使用して，SDP サーバとの接続を切断します。

```
con.close();
```

7.2.2　クエリ結果データの受信（インプロセス連携カスタムアダプター）

ここでは，インプロセス連携カスタムアダプターでのクエリ結果データの受信処理の基本的な流れについて，実装例を基に説明します。

インプロセス連携アダプターでのクエリ結果データの受信方式には，ポーリング方式とコールバック方式の 2 種類があります。それぞれの方式について説明します。

(1)　ポーリング方式

SDP サーバから能動的にストリームクエリ結果データを受信する方式です。

ポーリング方式によるクエリ結果データの受信処理について，実装例を次に示します。

実装例

```
public class Inpro_ReceiveSample1 implements StreamInprocessUP {
  // StreamInprocessUPの実装
  public void execute(SDPConnector con) {
    try {
      // 1. 出力ストリームに接続
      StreamOutput o = con.openStreamOutput("GROUP","QUERY1");

      // 2. タプルを受信
      try {
        while(true) {
          ArrayList tupleList = o.getAll();
        }
      } catch (SDPClientEndOfStreamException e) {
        System.out.println("データ受信完了");
      }

      // 3. 出力ストリームとの接続を切断
      o.close();

      // 4. SDPサーバとの接続を切断
      con.close();
    } catch (SDPClientException e) {
      System.err.println(e.getMessage());
    }
  }
}
```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

1. execute メソッドのパラメーターで渡されたSDPConnector 型オブジェクト（con）を使用して，クエリグループ名が"GROUP"，クエリ名が"QUERY1"の出力ストリームに接続し，StreamOutput 型オブジェクト（o）を取得します。

```
StreamOutput o = con.openStreamOutput(”GROUP”,”QUERY1”);
```

2. StreamOutput 型オブジェクト（o）を使用して，クエリ結果データ（タプル）を取得します。

```
ArrayList tupleList = o.getAll();
```

3. クエリ結果データの受信完了後，StreamOutput 型オブジェクト（o）を使用して，出力ストリームとの接続を切断します。

```
o.close();
```

4. SDPConnector 型オブジェクト（con）を使用して，SDP サーバとの接続を切断します。

```
con.close();
```

(2) コールバック方式

クエリ結果データ受信用のメソッドが SDP サーバによって呼び出される方式です。

コールバック方式によるクエリ結果データの受信処理について，実装例を次に示します。

コールバック方式によってクエリ結果データを受信する場合，次の 2 つのメソッドを実装する必要があります。

- StreamInprocessUP インタフェースを実装したクラスのexecute メソッド
- StreamEventListener インタフェースを実装したコールバッククラスのonEvent メソッド

それぞれの実装例について説明します。

実装例（execute メソッドの実装）

```
public class Inpro_ReceiveSample2 implements StreamInprocessUP {
  // StreamInprocessUPの実装
  public void execute(SDPConnector con) {  // AP起動時に実行される処理
    // 1. 出力ストリームに接続
    StreamOutput o = con.openStreamOutput(”GROUP”,”QUERY2”);

    // 2. コールバック用リスナーオブジェクトの生成
    CallBack notifiable = new CallBack();

    // 3. コールバック用リスナーオブジェクトの登録
    o.registerForNotification(notifiable);
  }
  public void stop() {  // AP停止時に実行される処理
    // 4. コールバック用リスナーオブジェクトの解除
    o.unregisterForNotification(notifiable);

    // 5. SDPサーバとの接続を切断
    con.close();
```

```
  }
}
```

実装例（onEvent メソッドの実装）

```
public class CallBack implements StreamEventListener {
  // StreamEventListenerの実装
  public void onEvent(StreamTuple tuple) {
    // 6. タプル受信時の処理
    System.out.println(”タプル受信:” + tuple);
  }
}
```

実装内容の説明

それぞれの処理の意味について説明します。番号は実装例中のコメントの番号に対応しています。

1. execute メソッドのパラメーターに指定されたSDPConnector 型オブジェクト（con）を使用して，クエリグループ名が"GROUP"，クエリ名が"QUERY2"の出力ストリームに接続し，StreamOutput 型オブジェクト（o）を取得します。

   ```
   StreamOutput o = con.openStreamOutput(”GROUP”,”QUERY2”);
   ```

2. StreamEventListener インタフェースの実装クラス（CallBack）のオブジェクト（notifiable）を生成します。

   ```
   CallBack notifiable = new CallBack();
   ```

3. StreamOutput 型オブジェクト（o）を使用して，手順 2.で作成したコールバック用リスナーオブジェクトを登録します。

   ```
   o.registerForNotification(notifiable);
   ```

4. カスタムアダプターを停止するときに，StreamOutput 型オブジェクト（o）を使用して，コールバック用リスナーオブジェクト（notifiable）を解除します。

   ```
   o.unregisterForNotification(notifiable);
   ```

5. カスタムアダプターを停止するときに，SDPConnector 型オブジェクト（con）を使用して，SDP サーバとの接続を切断します。

   ```
   con.close();
   ```

6. 出力ストリームにクエリ結果データのタプルが到着したときに，StreamEventListener インタフェースのonEvent メソッドに記述した内容が実行されます。

   ```
   public void onEvent(StreamTuple tuple) {
   // 6. タプル受信時の処理
     }
   ```

7.2.3 注意事項

インプロセス連携カスタムアダプターの実装では，System.exit メソッドを使用しないでください。インプロセス連携カスタムアダプターは SDP サーバと同じプロセスで実行されるため，インプロセス連携カスタムアダプター内でSystem.exit メソッドが呼び出された場合，SDP サーバも終了してしまいます。

7.3　カスタムアダプターで実装する処理

ここでは，カスタムアダプターで実装する次の処理について説明します。「7.2　インプロセス連携カスタムアダプターの作成」で示した基本的な処理と組み合わせて，必要に応じて実装してください。

- データ受信 AP の終了契機の把握（データ処理終了の通知）
- データソースモードでの時刻情報の設定
- キューあふれの事前防止
- ハートビートの設定
- 一時停止/再開

なお，この節で説明する機能の一部は，定義ファイルでのパラメーターの指定が必要になります。各定義ファイルの詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

また，処理で使用する API の詳細については，「8.　データ送受信用 API」を参照してください。

7.3.1　データ受信 AP の終了契機の把握（データ処理終了の通知）

データ受信 AP は，Streaming Data Platform のコマンドでは終了できないため，データ受信 AP の中で終了処理を実装しておく必要があります。

ここでは，データ受信 AP を終了するための契機をデータ受信 AP で把握するための実装について説明します。ここで説明する方法は，ストリームデータ処理エンジンであらかじめ送信ストリームデータの終了契機がわかる，次のような業務に使用できます。

- ファイルのデータをストリームデータとして処理する場合
- データ送信元からのデータの配信が定刻で終わることがわかっている場合

> **メモ**
>
> データ送信 AP を終了させる場合，データ処理終了の通知は不要です。データ送信 AP については，入力データの送信がなくなり，ストリームデータ処理エンジンへのデータ送信が完了した時点で終了させることができます。

(1)　データ受信 AP の終了契機の把握方法

データ受信 AP は，ストリームデータ処理エンジンでクエリが実行されている間，結果データの受信を繰り返し実行します。データ受信 AP を終了するためには，次の状態をデータ受信 AP で把握する必要があります。

- ストリームデータ処理エンジンでのクエリの実行がすべて終了していること

- 出力ストリームにクエリ結果データが残っていないこと

データ受信 AP 側では，これらの状態をデータ受信 API（get メソッドまたはgetAll メソッド）呼び出し時に発生する例外SDPClientEndOfStreamException または例外SDPClientQueryGroupStopException をキャッチすることで把握できます。これらの例外をキャッチしたことを契機として，データ受信 AP を終了させてください。

なお，これらの例外を発生させるためには，データ送信 AP でのメソッドの呼び出し，またはコマンドの実行によって，出力ストリームに対してデータ処理終了を通知する必要があります。

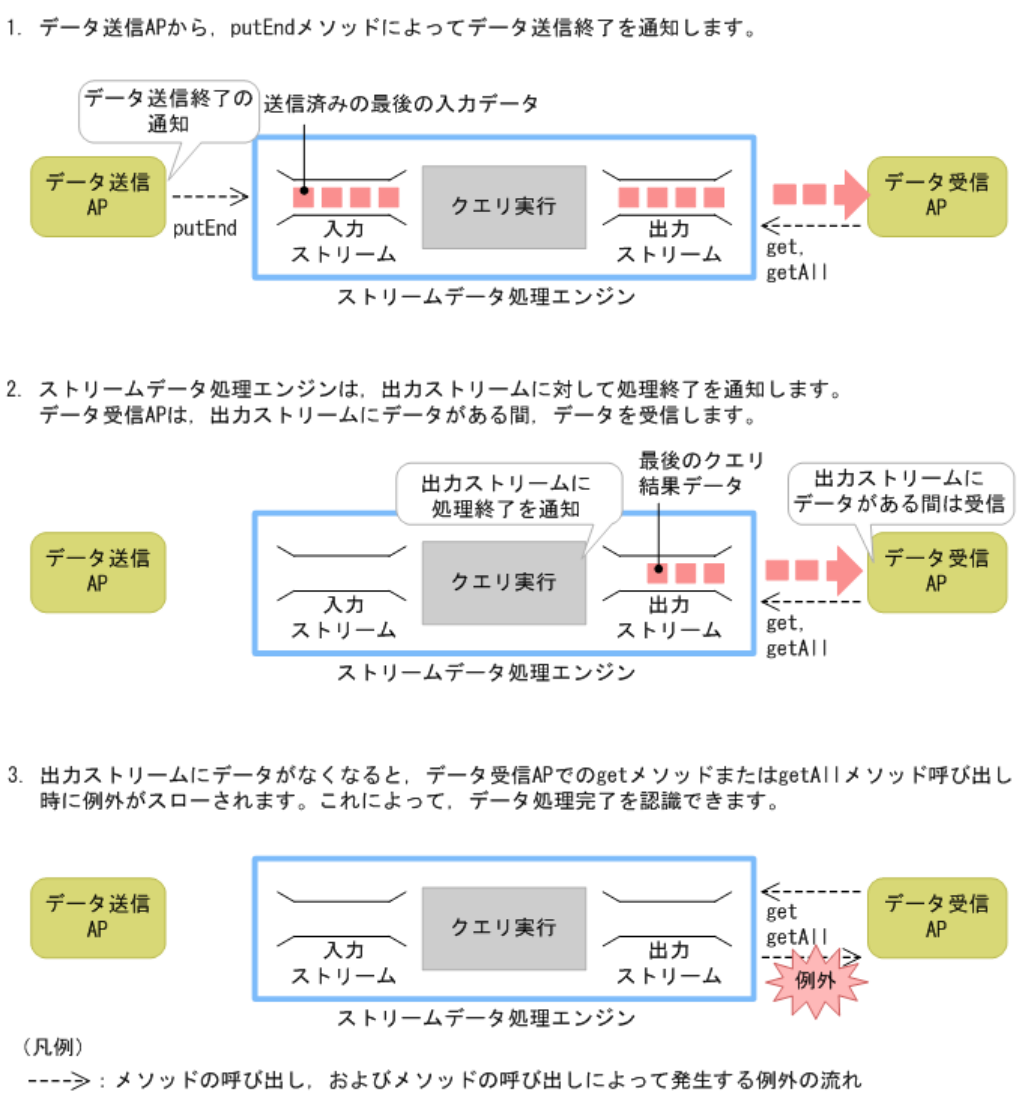
それぞれの方法での通知方法について，「7.3.1　データ受信 AP の終了契機の把握（データ処理終了の通知）」の「(2)　データ送信 AP のメソッド呼び出し（putEnd メソッド）によるデータ送信終了の通知」および「(3)　hsdpcqlstop コマンドの実行によるクエリグループ停止の通知」で説明します。

(2)　データ送信 AP のメソッド呼び出し（putEnd メソッド）によるデータ送信終了の通知

データ送信 AP からのデータ送信が終了したときに，putEnd メソッドを呼び出します。出力ストリームにクエリ結果データがなくなると，データ受信 AP に対して例外SDPClientEndOfStreamException がスローされます。

putEnd メソッドの呼び出しによってデータ送信終了を通知する流れを次の図に示します。

図 7-3　データ送信 AP によるデータ送信終了通知の流れ



図で示した流れの詳細を示します。なお，項番は，図中の番号と対応しています。

1. データ送信 AP では，ストリームデータ処理エンジンに対するストリームデータの送信がすべて終了した際，putEnd メソッドによって該当する入力ストリームに対して，データ送信終了を通知します。
2. ストリームデータ処理エンジンは，クエリグループ内のすべての入力ストリームに対して送信終了が通知されると，入力ストリーム内のすべてのタプルに対してクエリを実行して，クエリ結果データを出力ストリームに出力したあとで，すべての出力ストリームに対して処理終了を通知します。

   なお，処理終了が通知されたあとでも，データ受信 AP は，出力ストリーム内にクエリ結果データがある間はデータを受信し続けます。データの受信には，get メソッドまたはgetAll メソッドを使用します。
3. 出力ストリーム内のすべてのクエリ結果データの受信が完了して，出力ストリームにデータがなくなると，データ受信 AP でのget メソッドまたはgetAll メソッド呼び出しに対して例外 SDPClientEndOfStreamException がスローされます。

   データ受信 AP では，この例外を契機として，データ受信 AP を停止します。

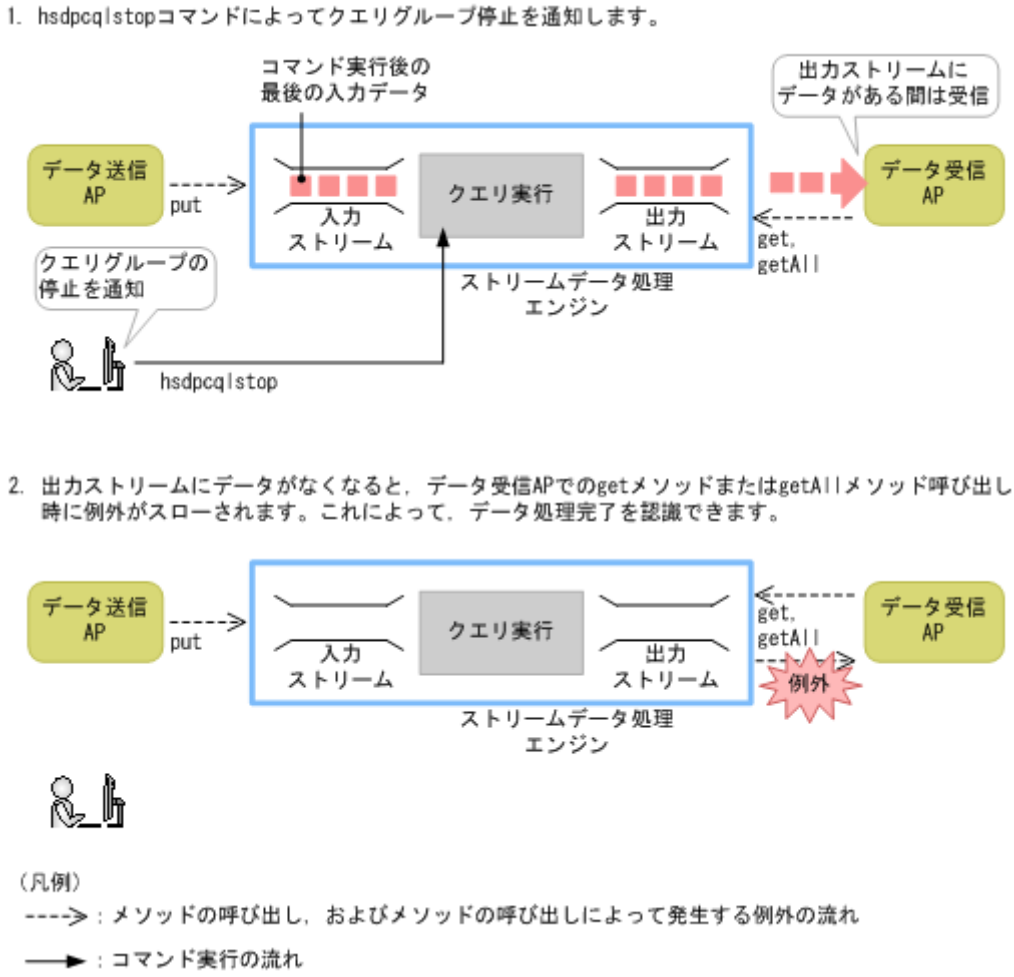
なお，クエリグループ内の入力ストリームの中にputEnd メソッドを呼び出していない入力ストリームがある場合，データ受信 AP からのget メソッドまたはgetAll メソッド呼び出し時に例外SDPClientEndOfStreamException が発生することはありません。

(3)　hsdpcqlstop コマンドの実行によるクエリグループ停止の通知

hsdpcqlstop コマンドを実行して，クエリグループを停止します。出力ストリームにクエリ結果データがなくなると，データ受信 AP に対して例外SDPClientQueryGroupStopException がスローされます。

hsdpcqlstop コマンドによるクエリグループ停止の流れを次の図に示します。

図 7-4　hsdpcqlstop コマンドによるクエリグループ停止の流れ



図で示した流れの詳細を示します。なお，項番は，図中の番号と対応しています。

1. hsdpcqlstop コマンドによってクエリグループ停止を通知すると，ストリームデータ処理エンジンでは，入力ストリームデータの受け付けを停止します。その後，すでに入力ストリームに格納されているデータの処理を実行してから，処理を終了します。

   なお，データ受信 AP では，コマンド実行後も出力ストリームにクエリ結果データがある間は，get メソッドまたはgetAll メソッドでデータを受信します。

2. 出力ストリームにデータがなくなると，データ受信 AP でのget メソッドまたはgetAll メソッドの呼び出し時に例外SDPClientQueryGroupStopException がスローされます。データ受信 AP では，この例外を契機として，データ受信 AP を停止します。

7.3.2　データソースモードでの時刻情報の設定

ストリームデータのタプルの時刻情報をデータソースモードで管理する場合，データ送信 AP で時刻情報を設定する必要があります。また，送信するタプルの時刻情報が昇順になるように，データ送信 AP で制御する必要があります。

ここでは，タプルに時刻情報を設定するためにデータ送信 AP で実装する内容について説明します。また，データ送信後のストリームデータ処理エンジンでの処理についても説明します。

なお，ここで説明する処理は，サーバモードの場合は不要な処理です。

(1)　タプルへの時刻情報の設定

データソースモードでは，送信するタプルのカラムデータに時刻情報を設定する必要があります。

時刻情報を設定するためには，クエリの定義と，データ送信 AP での実装が必要です。それぞれの例を次に示します。

クエリの定義

クエリ定義ファイルに記述するストリーム定義のスキーマ指定文字列に，時刻情報をTIMESTAMP 型で定義します。定義例を次に示します。

```
REGISTER STREAM s1(t1 TIMESTAMP, id INT, name VARCHAR(10));
```

データ送信 AP での実装

データ送信 AP で，タプルのカラムデータに格納する時刻情報として，java.sql.Timestamp クラスの Java オブジェクトを作成します。実装例を次に示します。

```
    :
    :
//タプルオブジェクトを生成します
Object[] data = new Object[]{
  new Timestamp(System.currentTimeMillis()),
  new Integer(100),
  new String("data1")
};

//オブジェクトをタプルに設定します
StreamTuple tuple1 = new StreamTuple(data);
    :
    :
```

注

実際にデータ送信 AP を作成する場合，new Timestamp には，ログファイルなどに含まれるデータソースの時刻情報を抽出して指定してください。

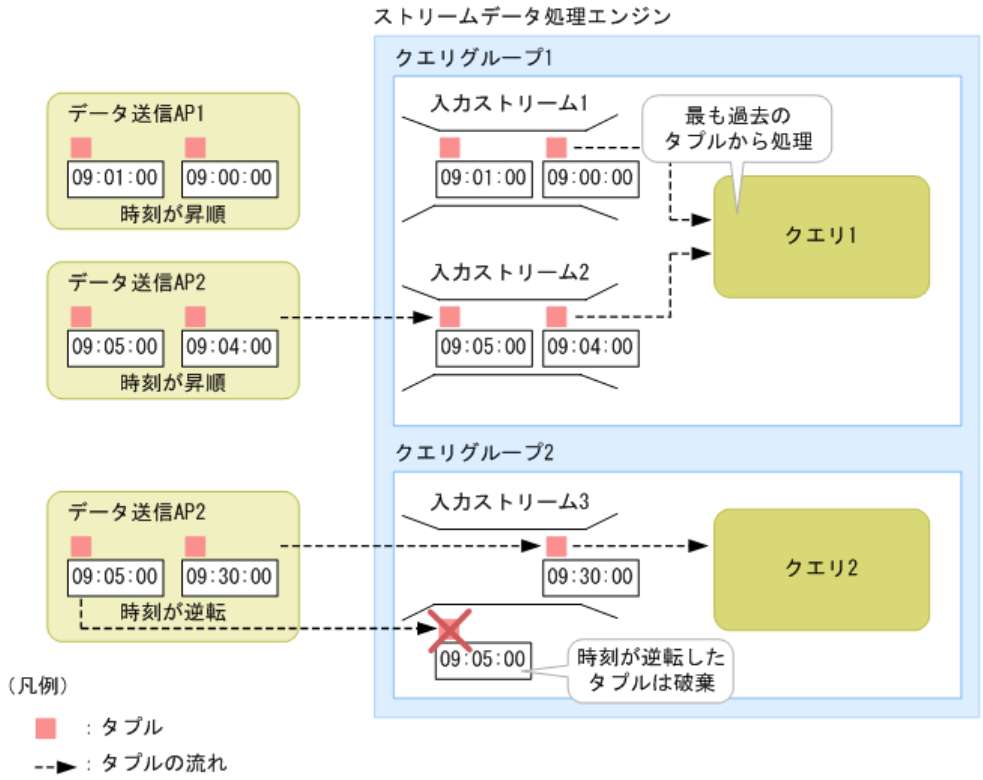
(2)　タプル送信時の時刻情報の制御

データソースモードの場合，送信するタプルの時刻情報が昇順になるように，データ送信 AP で制御する必要があります。

ストリームデータ処理エンジンでは，入力ストリームごとに，時刻情報の昇順で送信されたタプルに対して処理を実行します。時刻が逆転したタプルが送信された場合，そのタプルは SDP サーバで破棄されますので注意してください。

次の図に，複数の入力ストリームに対して複数のデータ送信 AP からタプルを送信する場合の処理順序の例を示します。この図では，入力ストリーム 1 と入力ストリーム 2 は同じクエリグループに含まれています。また，図中のタプル下に示した数値（「09:00:00」など）は，そのタプルに設定された時刻情報です。

図 7-5　複数の入力ストリームに対して複数のデータ送信 AP からタプルを送信する場合の処理順序の例



1 つのクエリグループ内に複数の入力ストリームがある場合，ストリームデータ処理エンジンでは，最も過去の時刻情報が設定されたタプルから処理します。図の例の場合は，**入力ストリーム 1** のタプルを処理したあとで，**入力ストリーム 2** のタプルを処理します。なお，クエリの実行には，putEnd メソッドで処理の終了を通知済みのストリーム以外，すべてのストリームにタプルがあることが必要です。1 つでも空の

ストリームがある場合，最も過去のタプルがどのストリームから送信されるかが判断できないため，ストリームデータ処理エンジンは処理を実行しません。

また，時刻情報が逆転したタプルが到着した場合，ストリームデータ処理エンジンでは，そのタプルを破棄します。図の場合は，**入力ストリーム 3** に到着したタプルの時刻情報の順序が逆転していたため，あとから到着したタプルは破棄します。

> **メモ**
>
> データソースモードでデータを送信する場合，複数のデータ送信 AP から送信されたデータについて，誤差によってストリームデータ処理エンジンに到着する順序が逆転して，処理がタイムスタンプ順の昇順にならないことがあります。この場合に，順序が逆転したファイルを破棄しないで処理するためには，タイムスタンプ調整機能を使用して時刻調整を実行する必要があります。タイムスタンプ調整機能による時刻調整については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』の，タプルのタイムスタンプの調整の検討についての説明を参照してください。

(3) タプル入力完了後の処理

データ送信 AP からのタプルの入力が完了したら，ストリームデータ処理エンジンに対してデータ送信の完了を通知してください。

データソースモードの場合，ストリームデータ処理エンジンでは，タプルの時刻情報を基に，時系列に処理を実行します。データ送信 AP からのタプルの入力が完了して，入力ストリームにタプルがなくなると，ストリームデータ処理エンジンの時刻は停止します。ウィンドウ演算などのクエリの処理は動作時刻が進んだときに実行されるため，入力ストリームにタプルがない場合，ストリームデータがストリーム処理エンジン内で滞留してしまうことがあります。

このような滞留を防ぐためには，データ送信 AP 側で，データ送信の完了を明示的に通知する必要があります。データ送信の完了が通知されると，ストリームデータ処理エンジンでは，滞留しているすべてのタプルの処理を実行して，結果を出力ストリームに出力します。

データ送信完了の通知方法については，「7.3.1　データ受信 AP の終了契機の把握（データ処理終了の通知）」を参照してください。

7.3.3 キューあふれの事前防止

ストリームデータ処理エンジンとカスタムアダプター間でタプルをやり取りする際，通路として入力ストリームのキュー（**入力ストリームキュー**）と出力ストリームのキュー（**出力ストリームキュー**）を使用します。これらのキューは，FIFO 方式でタプルを制御します。

入力ストリームキューおよび出力ストリームキューで管理できるタプルの最大値は，次の定義ファイルで指定できます。

- system_config.properties
- クエリグループ用プロパティファイル
- ストリーム用プロパティファイル

指定した値を超えた数のタプルが送信されると，キューあふれが発生します。

ここでは，入力ストリームキューまたは出力ストリームキューでキューあふれが発生した場合の動作，およびキューあふれを事前に防止するための実装方法について説明します。

(1) キューあふれが発生した場合の動作

キューあふれが発生した場合の動作は，入力ストリームキューがあふれた場合と出力ストリームキューがあふれた場合で異なります。

入力ストリームキューでキューあふれが発生した場合の動作について，次の表に示します。

表 7-1 入力ストリームキューでキューあふれが発生した場合の動作

項番	タイムスタンプモード	タイムスタンプ調整機能	動作
1	サーバモード	－	• データ送信 AP では，put メソッドの呼び出しに対して例外 SDPClientFreeInputQueueSizeLackException がスローされます。 • ストリームデータ処理エンジンでは，データ送信 AP から送信されたタプル数と入力ストリームキューの空きサイズを比較して入力ストリームキューがあふれる場合，送信されたタプルを一切入力ストリームキューに登録しません。ただし，クエリグループは閉塞しません。
2	データソースモード	有効	• データ送信 AP では，put メソッドの呼び出しに対して例外 SDPClientException がスローされます。 • ストリームデータ処理エンジンでは，クエリグループを閉塞します。また，クエリグループの閉塞に伴って，入力ストリームキューに登録されていたすべての入力タプルを破棄します。ただし，その閉塞処理によって出力ストリームキューに登録されたタプルは破棄しません。
3		無効	• データ送信 AP では，put メソッドの呼び出しに対して例外 SDPClientFreeInputQueueSizeLackException がスローされます。 • ストリームデータ処理エンジンでは，データ送信 AP から送信されたタプル数と入力ストリームキューの空きサイズを比較して入力ストリームキューがあふれる場合，送信されたタプルを一切入力ストリームキューに登録しません。ただし，クエリグループは閉塞しません。

（凡例）

－：該当しません。

出力ストリームキューのキューあふれが発生した場合の動作について，次の表に示します。動作は，定義ファイル（system_config.properties またはクエリグループ用プロパティファイル）の querygroup.sleepOnOverStoreRetryCount パラメーターの指定によって異なります。

表 7-2　出力ストリームキューのキューあふれが発生した場合の動作

項番	定義ファイルの指定	動作
1	querygroup.sleepOnOverStoreRetryCount=1以上を指定した場合	クエリグループが登録する出力タプル数と出力ストリームキューの空きサイズを比較して，出力ストリームキューがあふれる場合，クエリグループの実行を一時的に停止します（クエリを実行するスレッドをスリープさせます）。 querygroup.sleepOnOverStoreRetryCount に指定した値の回数分，空きサイズをチェックします（空きができた場合，チェック回数は 0 にリセットされます）。 指定した回数を超えて，出力ストリームキューに出力タプルを登録した場合に出力ストリームキューがあふれたときは，クエリグループを閉塞します。また，クエリグループの閉塞に伴って，入力ストリームキューに登録されていたすべてのタプルを破棄します。ただし，その閉塞処理の際に出力ストリームキューに登録されたタプルについては，破棄しません。
2	querygroup.sleepOnOverStoreRetryCount＝0 を指定した場合	クエリグループを閉塞します。また，クエリグループの閉塞に伴って入力ストリームキューに登録されたすべてのタプルを破棄します。ただし，その閉塞処理の際に出力ストリームキューに登録されたタプルについては，破棄しません。
3	querygroup.sleepOnOverStoreRetryCount=-1 を指定した場合	クエリグループが登録する出力タプル数と出力ストリームキューの空きサイズを比較して，出力ストリームキューがあふれる場合，クエリグループの実行を一時的に停止します（クエリを実行するスレッドをスリープさせます）。この処理を，出力ストリームキューの空きが出るまで無限に繰り返します。

(2)　入力ストリームキューのキューあふれの事前防止

ここでは，データ送信 AP の実装によって，入力ストリームキューのキューあふれを事前に防止する方法について説明します。

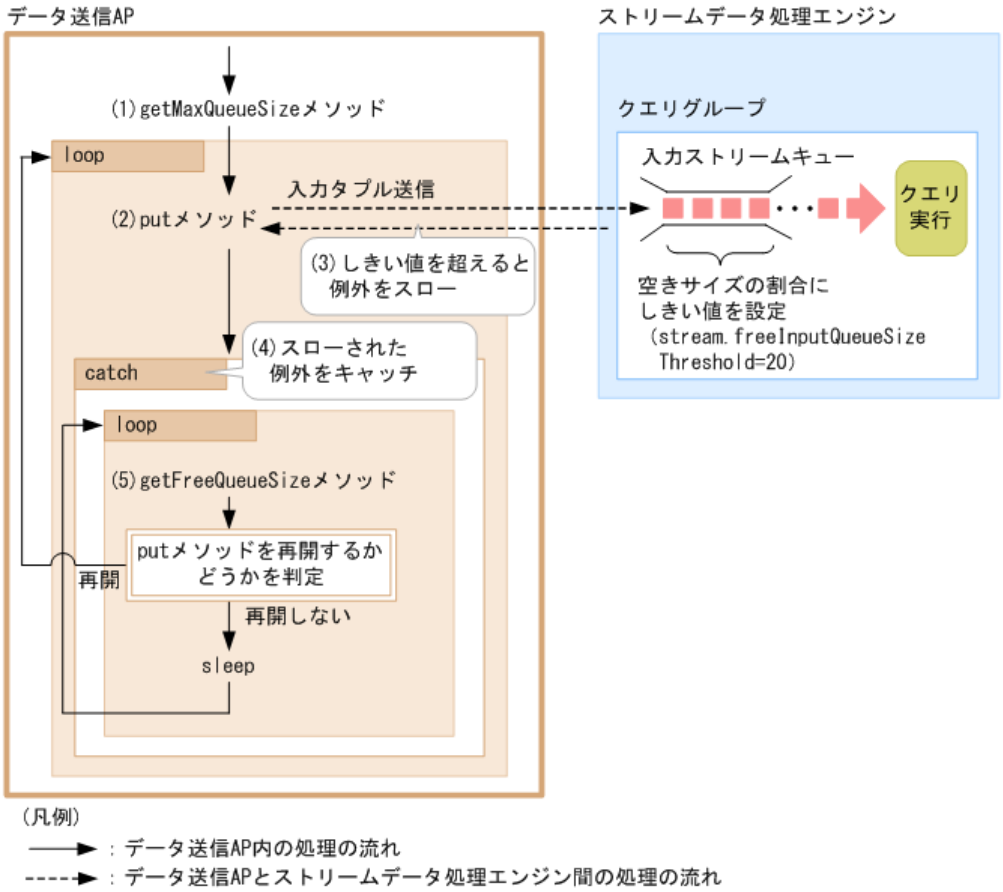
メモ

ここで説明する処理を有効にするためには，SDP サーバの定義ファイル（system_config.properties，クエリグループ用プロパティファイル，またはストリーム用プロパティファイル）でstream.freeInputQueueSizeThreshold パラメーターを指定しておく必要があります。

なお，以降の例は，stream.freeInputQueueSizeThreshold=20 と指定した場合の例です。

キューあふれを事前に防止するデータ送信 AP の実装方法について，概要を次の図に示します。

図 7-6　入力ストリームキューのキューあふれを事前に防止するデータ送信 AP の実装方法



図で示した処理の内容について説明します。説明の番号は図中の番号と対応しています。

1. データ送信 AP でgetMaxQueueSize メソッドを呼び出し，入力ストリームのキューの最大サイズを取得します。この値は，手順 5.で入力ストリームキューの空きサイズが全体の何割残っているかを算出するときに使用します。
2. データ送信 AP では，put メソッドによって，ストリームデータ処理エンジンに入力タプルを送信します。送信されたタプルは，ストリームデータ処理エンジンのクエリグループで管理されている入力ストリームキューに格納されます。
3. 入力ストリームキューの空きサイズが，定義ファイルに指定したしきい値（stream.freeInputQueueSizeThreshold の指定値）の割合以下になると，ストリームデータ処理エンジンからデータ送信 AP に例外SDPClientFreeInputQueueSizeThresholdOverException が通知されます。
4. データ送信 AP では，ストリームデータ処理エンジンからの通知によってput メソッドに対してスローされた例外SDPClientFreeInputQueueSizeThresholdOverException をキャッチします。

   なお，このとき，定義ファイルにstream.freeInputQueueSizeThresholdOutputMessage=true を指定していた場合は，メッセージログにメッセージ KFSP42032-W が出力されます。
5. データ送信 AP で，getFreeQueueSize メソッドを呼び出して，入力ストリームキューの空きサイズを取得します。この値を基に，put メソッドの呼び出しを再開できるかどうかを判定します。

   再開できると判定した場合，put メソッドの呼び出しを再開します。

再開できないと判定した場合，一定時間スリープしたあとで，再度getFreeQueueSize メソッドを呼び出して，put メソッドの呼び出しを再開できるか判定します。

この判定処理によって，入力ストリームキューの空きサイズが確保できるまで，put メソッドによるタプルの送信は抑止されます。

> **重要**
>
> stream.freeInputQueueSizeThreshold を指定しても入力ストリームキューがあふれてしまった場合，ストリームデータ処理エンジンは，データ送信 AP に対して，例外 SDPClientFreeInputQueueSizeLackException をスローします。この場合，例外 SDPClientFreeInputQueueSizeThresholdOverException はスローされません。

入力ストリームキューのキューあふれを事前に防止する実装例を次に示します。なお，実装例中の処理 1 と処理 2 は，判定方法に応じてどちらかの処理を実装してください。

実装例

```
:
try {
  streamInput = connector.openStreamInput(TARGET_QUERYGROUP, TARGET_STREAM);

  //キューの最大サイズを取得する。
  int maxQueueSize = streamInput.getMaxQueueSize();

  for (int i=0; i<1000; i++){
    Object[] data = new Object[] {
      new Integer(i),
      new String("data:"+i)
      };
    StreamTuple tuple = new StreamTuple(data);

    try {
      streamInput.put(tuple);
    } catch (SDPClientFreeInputQueueSizeThresholdOverException sce) {
      //キューの空きサイズが20%以下になった。
      // (stream.freeInputQueueSizeThreshold=20指定時) 。
      while (true) {
        int freeQueueSize = streamInput.getFreeQueueSize();


        //処理1：キューの空きサイズを確認してputメソッドを再開する場合。
        //キューの空きサイズが500以上になったら，putメソッドを再開する。
        if (freeQueueSize >= 500) {
          break;
        } else {
          Thread.sleep(100);
        }
        //処理1はここまで。

        //処理2：キューの空きサイズと最大サイズの割合を確認して
        //putメソッドを再開する場合。
        double freePerMax = ((double)freeQueueSize / (double)maxQueueSize) * 100;
        //キューの空きサイズが50%以上となったら，putメソッドを再開する。
```

```
        if (freePerMax >= 50) {
          break;
        } else {
          Thread.sleep(100);
        }
        //処理2はここまで。

      }
    } catch (SDPClientFreeInputQueueSizeLackException sce2) {
      //キュー空きサイズがない。
      //(putメソッドで送信したタプルはストリームデータ処理エンジンに
      //受け付けられなかった)。
      //このため，タプルを再送する前にスリープする。
      Thread.sleep(100);
      try {
        //タプルを再送する。
        streamInput.put(tuple);
      } catch (SDPClientFreeInputQueueSizeThresholdOverException sce3) {
        //タプルの再送が成功したので，ここでのしきい値例外は無視する。
      }
    }
  }
  streamInput.putEnd();
} catch (SDPClientFreeInputQueueSizeLackException sce4) {
  //キューの空きサイズがない。
  //(putメソッドで送信したタプルはストリームデータ処理エンジンに
  //受け付けられなかった)。
  : (省略)
  //送信を停止する。
  streamInput.putEnd();
  : (省略)
} catch (...) {
  //その他の例外をcatchしたときの処理を実装する。
  : (省略)
```

(3) 出力ストリームキューのキューあふれの事前防止

出力ストリームキューのキューあふれは，定義ファイルのquerygroup.sleepOnOverStoreRetryCount パラメーターおよびquerygroup.sleepOnOverStore パラメーターの指定で防止します。これらのパラメーターの指定によって，出力ストリームキューがあふれそうな場合に，クエリグループの実行が一時停止（スリープ）されます。

また，「7.3.3 キューあふれの事前防止」の「(2) 入力ストリームキューのキューあふれの事前防止」で示した実装をしておくことで，クエリグループの実行が一時停止した場合に，入力ストリームキューに対する入力タプルの送信も抑止できるため，キューをあふれさせることなく，システムの処理を正常に戻すことができます。

出力ストリームキューのキューあふれが事前に防止される処理の概要について，次の図に示します。データ送信 AP の実装は，「7.3.3 キューあふれの事前防止」の「(2) 入力ストリームキューのキューあふれの事前防止」と同じです。

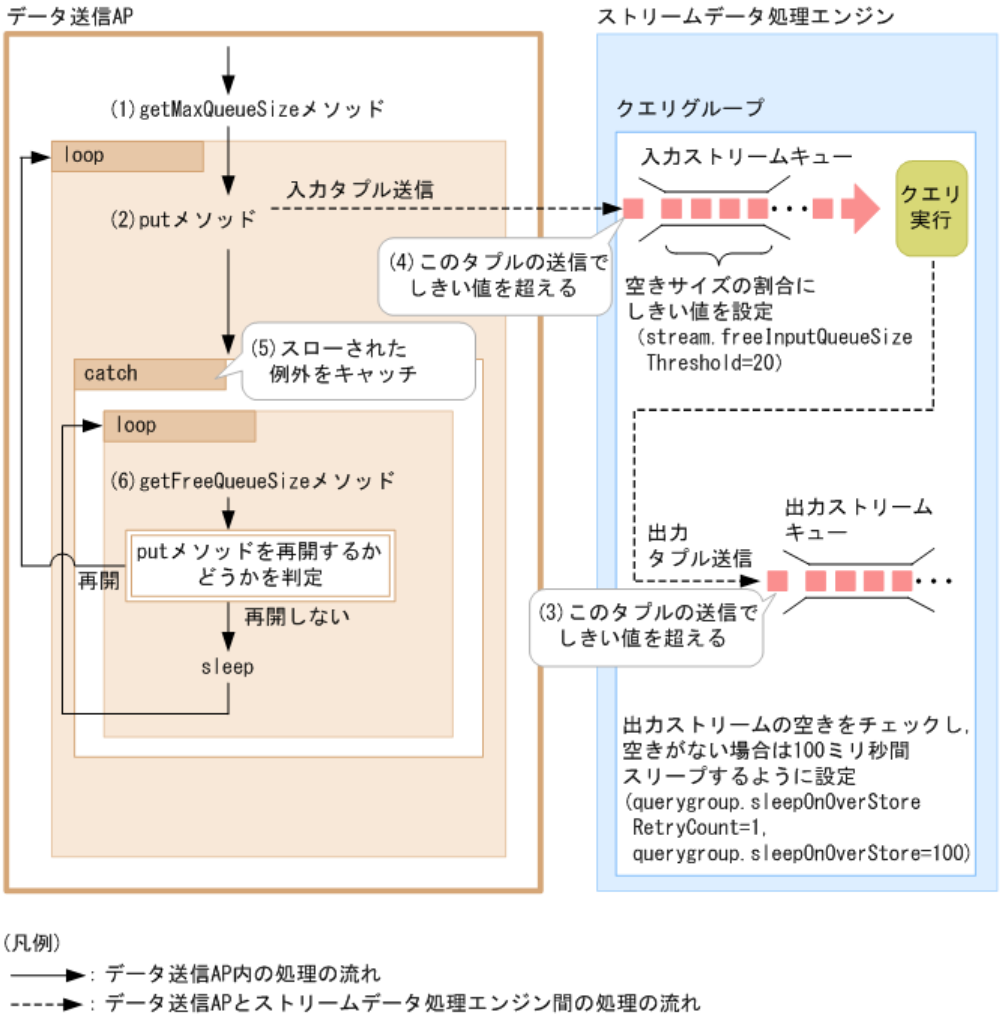
メモ

ここで説明する処理を有効にするためには，SDP サーバの定義ファイルで次のパラメーターを指定しておく必要があります。

- stream.freeInputQueueSizeThreshold
- querygroup.sleepOnOverStoreRetryCount
- querygroup.sleepOnOverStore

なお，この例は，それぞれ，stream.freeInputQueueSizeThreshold=20，querygroup.sleepOnOverStoreRetryCount=1，querygroup.sleepOnOverStore=100 と指定した場合の例です。

図 7-7　出力ストリームキューのキューあふれが事前に防止される処理の概要



図で示した処理の内容について説明します。説明の番号は図中の番号と対応しています。

1. データ送信 AP でgetMaxQueueSize メソッドを呼び出し，入力ストリームのキューの最大サイズを取得します。この値は，入力ストリームキューの空きサイズが全体の何割残っているかを算出するときに使用します。

2. データ送信 AP では，put メソッドによって，ストリームデータ処理エンジンにタプルを送信します。送信されたタプルは，ストリームデータ処理エンジンのクエリグループで管理されている入力ストリームキューに格納されます。
3. ストリームデータ処理エンジンでは，クエリ実行によって送信された出力タプルによって出力ストリームキューがあふれることを検知すると，クエリグループの実行を一時停止（スリープ）します。スリープするかどうかの設定，およびスリープさせる時間については，querygroup.sleepOnOverStoreRetryCount パラメーターおよびquerygroup.sleepOnOverStore パラメーターの指定値に従います。

   クエリグループがスリープすると，ストリームデータ処理エンジンは，メッセージログにメッセージ KFSP42033-W を出力します。

   なお，このとき，スリープ中のクエリグループのステータスは「**実行中**」となっています。データ送信 AP による入力ストリームキューに対するタプルの送信，およびデータ受信 AP による出力ストリームキューからのタプルの取得は許可されています。
4. クエリグループの実行がスリープしている間，データ送信 AP がput メソッドを呼び出し続けると，入力ストリームキューに処理されていないタプルが蓄積されていきます。入力ストリームキューの空きサイズが，定義ファイルに指定したしきい値（stream.freeInputQueueSizeThreshold パラメーターの指定値）の割合以下になると，ストリームデータ処理エンジンからデータ送信 AP に例外 SDPClientFreeInputQueueSizeThresholdOverException が通知されます。
5. データ送信 AP では，ストリームデータ処理エンジンからの通知によってput メソッドに対してスローされた例外SDPClientFreeInputQueueSizeThresholdOverException をキャッチします。
6. データ送信 AP で，getFreeQueueSize メソッドを呼び出して，入力ストリームキューの空きサイズを取得します。この値を基に，put メソッドの呼び出しを再開できるかどうかを判定します。

   再開できると判定した場合，put メソッドの呼び出しを再開します。

   再開できないと判定した場合，一定時間スリープしたあとで，再度getFreeQueueSize メソッドを呼び出して，put メソッドの呼び出しを再開できるか判定します。

   この処理によって，入力ストリームキューの空きサイズが確保できるまで，put メソッドによる入力タプルの送信は抑止されます。出力ストリームキューの空きサイズが確保され，クエリグループの実行が再開されて入力ストリームキューのタプルが処理されると，再度，入力タプルの送信ができるようになり，システムが正常な状態に戻ります。

7.3.4 ハートビートの設定

ハートビートは，入力ストリームのクロックを，ハートビートタプルで指定した時刻に設定する機能です。カスタムアダプターがハートビートタプル（指定されたタイムスタンプ付き）を入力ストリームに配置するために必要な API は，Hitachi Streaming Data Platform によって提供されます。

ハートビートタプルが入力ストリームによって受信されると，入力ストリームのクロックはタイムスタンプに設定された時刻まで継続され，クエリ処理（入力ストリームを含む）を続行できます。

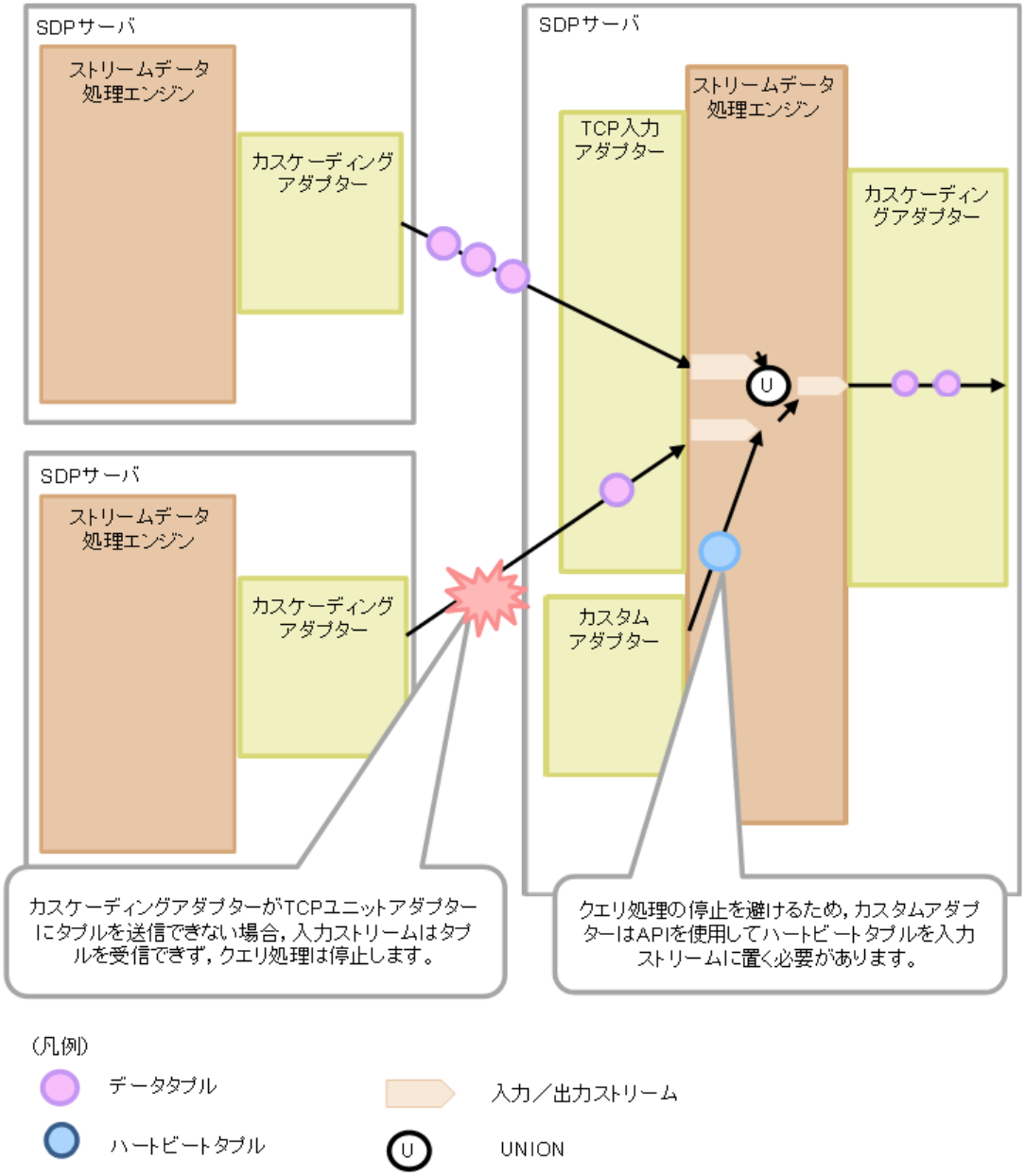
> **メモ**
>
> 入力ストリームがサーバモードで実行されている場合，API はハートビートタプルを入力ストリームに配置できません。

また，入力ストリームの現在のクロックよりも古いタイムスタンプを持つ古いハートビートタプルが入力ストリームによって受信されると，タプルは入力ストリームによって破棄されます。次の条件がすべて満たされている場合は，ハートビートを使用する必要があります。

- データソースモードが選択されている。
- 1 つのクエリグループに複数の入力ストリームが登録されている。
- 入力ストリームを結合するクエリ処理（UNION など）が，クエリグループによって実行されている。
- 処理が定期的に実行され，タプルが特定の間隔で入力ストリームに配置される。

これらの条件をすべて満たす場合に，タプルを受信していない入力ストリームが 1 つでもあるときは，その受信していない入力ストリームがタプルを受信するまでクエリ処理は停止します。この問題を回避するには，カスタムアダプターは特定の時間間隔で入力ストリームにハートビートタプルを送信する必要があります。

図 7-8　ハートビート機能のユースケース



実装例

```
public class Inpro_SendSample implements StreamInprocessUP {
public void execute(SDPConnector con) {
    try {
        // Connect to the input stream "STREAM1" on the query group "GROUP".
        StreamInput in = con.openStreamInput("GROUP","STREAM1");

        // Repeat putting a heartbeat tuple in the input stream until an exception occurs
.
        while (1) {
            // Put a heartbeat tuple in the input stream with current timestamp.
            in.putHeartbeat(new Timestamp(System.currentTimeMillis()));

            Thread.sleep(1000);
        }
```

```
    } catch (SDPClientFreeInputQueueSizeLackException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientFreeInputQueueSizeThresholdOverException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientServerModeException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientSuspendException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientQueryGroupStopException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientQueryGroupHoldException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientException sce) {
        System.err.println(sce.getMessage());
    } catch (InterruptedException e) {
        e.printStackTrace();
    } finally {
        if (!con.isClosed()) {
            try {
                in.putEnd();
                in.close();
                con.close();
            } catch (SDPClientException sce2) {
                System.err.println(sce2.getMessage());
            }
        }
    }
}
```

7.3.5 一時停止/再開

一時停止機能は入力ストリームを無効にし，再開機能は一時停止された入力ストリームを再び有効にします。Streaming Data Platform は，カスタムアダプターが入力ストリームを一時停止および再開するために必要な API を提供します。カスタムアダプターを使用して入力ストリームを一時停止すると，入力ストリームのタプルが破棄され，入力ストリームがクエリ処理から削除されます。一時停止された入力ストリームは，入力タプルを破棄します。カスタムアダプターが API を使用して一時停止された入力ストリームを再開すると，一時停止された入力ストリームが再びクエリ処理の範囲に含まれます。

一時停止された入力ストリームを手動で，または自動的に再開できます。カスタムアダプターが再開 API を呼び出すと，一時停止された入力ストリームを手動で再開できます。一時停止された入力ストリームがタプルを受信すると，一時停止された入力ストリームは自動的に再開されます。入力ストリームを再開する方法は，一時停止 API によって指定できます。

> **メモ**
>
> 入力ストリームがサーバモードで実行されている場合，API は入力ストリームを一時停止および再開できません。

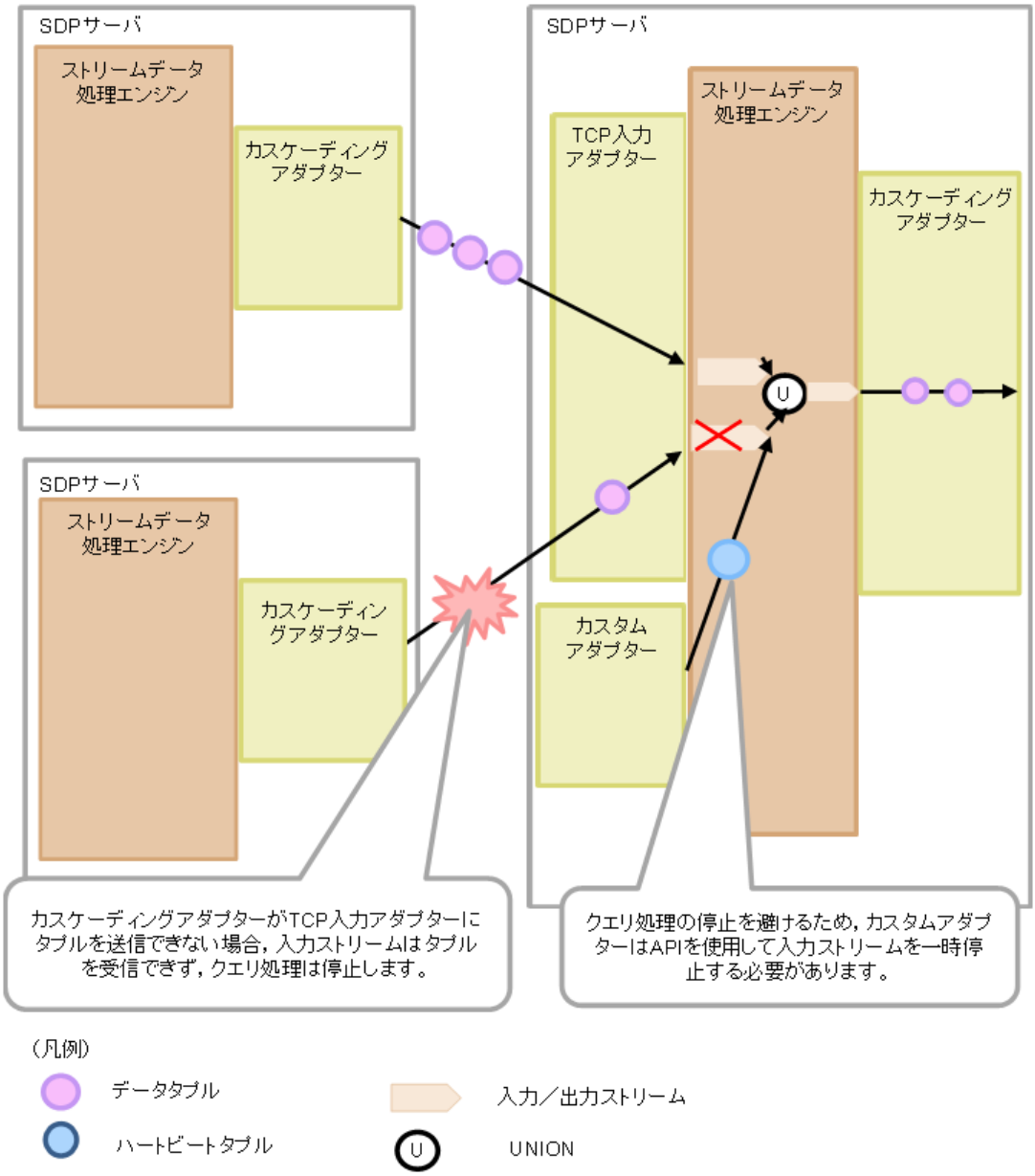
次の条件が満たされている場合は，一時停止機能と再開機能を使用することをお勧めします。

- データソースモードが選択されている。
- 1 つのクエリグループに複数の入力ストリームが登録されている。
- クエリグループが，UNION などの入力ストリームを結合する処理を実行する。
- クエリグループが処理を不定期に実行し，タプルがランダムな時間に入力ストリームに配置される。

これらの条件をすべて満たす場合に，タプルを受信していない入力ストリームが 1 つでもあるときは，その受信していない入力ストリームがタプルを受信するまでクエリ処理は停止します。

この問題を回避するには，カスタムアダプターは入力ストリームがタプルを受信していない間，入力ストリームを一時停止する必要があります。

図 7-9　一時停止機能と再開機能のユースケース



実装例

```
public class Inpro_SendSample implements StreamInprocessUP {
public void execute(SDPConnector con) {
    try {
        // Connect to the input stream "STREAM1" on the query group "GROUP".
        StreamInput in = con.openStreamInput("GROUP","STREAM1");

        for (int i = 0; i < 10; i++) {
            // Get the status of the input stream.
            SuspendStatus status = in.getSuspendStatus();

            switch (status) {
            case NOT_SUSPEND:
                // The input stream is not suspended.
                boolean autoResume = true;
                if (i / 3) {
                    autoResume = false
                }

                // Suspend the input stream.
                in.suspend(autoResume);
                break;
            case SUSPEND:
                // The input stream is suspended (ストリームは自動的に再開しません).
                // Resume the suspended input stream.
                in.resume();
                break;
            case SUSPEND_AUTO_RESUME:
                // The input stream is suspended (ストリームが自動的に再開します).
                // Put a tuple to resume the suspended input stream
                Object[] data = new Object[]{
                    new Integer(i),
                    new String("data:"+i)
                };

                StreamTuple tuple = new StreamTuple(data);

                streamInput.put(tuple);
                break;
            default:
                break;
            }

            Thread.sleep(1000);
        }
        in.putEnd();
    } catch (SDPClientServerModeException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientSuspendException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientQueryGroupStopException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientQueryGroupHoldException sce) {
        System.err.println(sce.getMessage());
    } catch (SDPClientException sce) {
        System.err.println(sce.getMessage());
    } finally {
```

```
        if (!connector.isClosed()) {
            try {
                in.close();
                con.close();
            } catch (SDPClientException sce2) {
                System.err.println(sce2.getMessage());
            }
        }
    }
}
```

7.4 コンパイル手順

カスタムアダプターのコンパイルは，次の手順で実行してください。

1. **準備として，環境変数を定義します。**

   Streaming Data Platform のセットアップ実行時に設定した運用ユーザーの環境変数PATH に，次のパスを追加してください。この手順は，2 度目以降のコンパイルでは不要な手順です。

   ```
   ¥opt¥hitachi¥hsdpbase¥system¥psb¥jdk¥bin
   ```

2. **コンパイルを実行します。**

   次のコマンドを実行してください。

   ```
   javac -cp インストールディレクトリ¥lib¥sdp.jar Javaソースプログラム
   ```

3. **jar ファイルを作成します（jar ファイルに複数のクラスをまとめたい場合）。**

   test ディレクトリ下のクラスファイル「A.class」と「B.class」を「test.jar」というファイルにまとめる場合のコマンドの実行例を次に示します。

   ```
   jar cf test.jar test¥A.class test¥B.class
   ```

7.5　カスタムアダプター作成時の留意事項

ここでは，カスタムアダプター作成時の留意事項について説明します。

カスタムアダプターのデータ送信 AP およびデータ受信 AP は，次の方針で作成してください。

- 同一ストリームに対して複数回接続しないでください。

  1 つのストリームは，それぞれ 1 回だけ接続するようにしてください。

  1 つのSDPConnector 型オブジェクトでは，同一のストリームに対して複数回接続できませんが，複数のSDPConnector 型オブジェクトを使用すると，同一ストリームに複数回接続できます。しかし，複数回接続した場合，ストリームデータ処理エンジンのスループットが下がります。
- データ受信 AP の処理負荷に対して，クエリ結果データの出力ストリームへの出力頻度が高い場合は，次のどちらかの方法を使用して，データ受信 AP の通信コストを抑えてください。

  複数のクエリ結果のポーリング方式での受信をgetAll メソッドで一括受信してください。これによって，データ転送の通信コストを抑えられます。

  データ受信 AP を SDP サーバと同一プロセスで動作するように，インプロセス連携カスタムアダプターとして作成してください。これによって，プロセス間通信に掛かる通信コストを抑えられます。

  通信コストを抑えることで，出力ストリームキューのタプルの滞留を防止できるため，キューあふれ防止にもなります。
- クエリ結果の出力頻度が小さい場合は，データ受信 AP を SDP サーバと同一プロセスで動作するインプロセス連携カスタムアダプターとして作成して，データ受信をコールバック受信方式にすることをお勧めします。これによって，システムの CPU 利用率を低減できます。

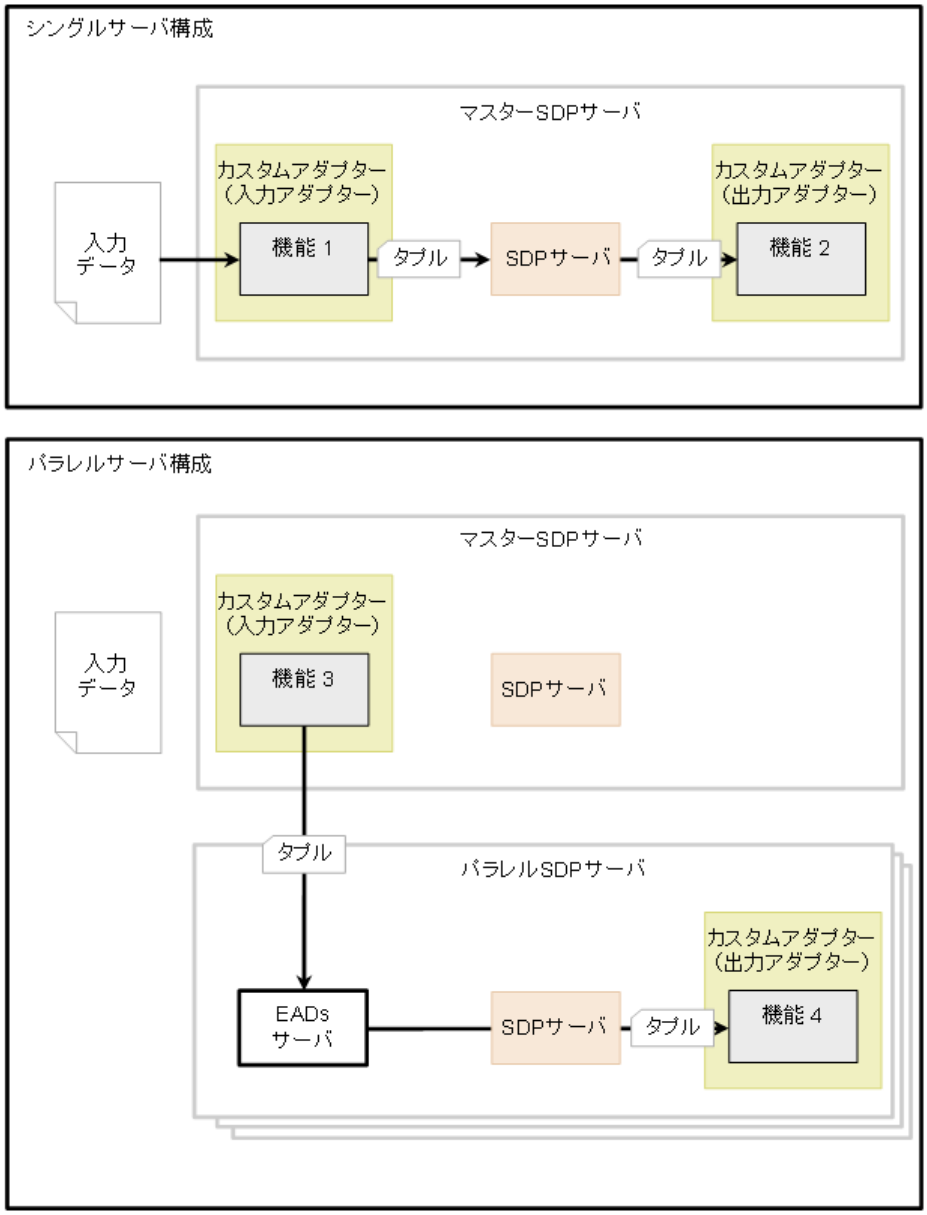
7.6　カスタムアダプターの機能

カスタムアダプターは，標準提供アダプターがサポートしていないデータを入力し，そのデータを SDP サーバに送信できます。カスタムアダプターには，次の表に示す機能があります。

表 7-3　カスタムアダプターの機能

図中の機能の番号	機能	説明
1	シングルサーバ構成でのデータ送信機能	入力データをタプルに変換し，そのタプルをマスター SDP サーバの SDP サーバに送信します。
2	シングルサーバ構成でのデータ受信機能	マスター SDP サーバの SDP サーバから，タプルを受信します。
3	パラレルサーバ構成でのデータ送信機能	入力データをタプルに変換し，そのタプルをパラレル SDP サーバの EADs サーバに送信します。
4	パラレルサーバ構成でのデータ受信機能	パラレル SDP サーバの SDP サーバから，タプルを受信します。この機能を開発する手順は，シングルサーバ構成のデータ受信機能と同じです。

図 7-10　シングルサーバ構成とパラレルサーバ構成の概要



7.6.1　パラレルサーバ構成でのデータ送信機能の開発

ここでは，パラレルサーバ構成でデータ送信機能を開発する方法について説明します。ほかの機能の開発の詳細については，「8.　データ送受信用 API」を参照してください。

この機能を開発する手順は，シングルサーバ構成でデータ送信機能を開発する手順と似ていますが，使用する API が異なります。シングルサーバ構成のデータ送信機能で使用する API と，パラレルサーバ構成のデータ送信機能で使用する API の対応を次の表に示します。

表 7-4　データ送信機能で使用する API の対応

タスク	シングルサーバ構成のデータ送信機能で使用する API		パラレルサーバ構成のデータ送信機能で使用する API	
	クラス	メソッド	クラス	メソッド
SDP サーバに接続する	SDPConnectorFactory	connect ()	SDPClientFactory	create (String, String, int, int, long, int, int)
入力ストリームに接続する	SDPConnector	openStreamInput (String, String)	SDPClient	openStreamInput (String, String)
入力ストリームにタプルを配置する	StreamInput	put (StreamTuple)	StreamParallelInput	put (String, StreamTuple)
入力ストリームにタプルの終了を配置する	StreamInput	putEnd ()	StreamParallelInput	putEnd ()
入力ストリームから切断する	StreamInput	close ()	StreamParallelInput	close ()
SDP サーバから切断する	SDPConnector	close ()	SDPClient	destroy ()

カスタムアダプターの開発に使用される API の詳細については，「7.6.2　カスタムアダプター用 API」または「7.　カスタムアダプターの作成」を参照してください。

7.6.2　カスタムアダプター用 API

(1)　API 一覧

Hitachi Streaming Data Platform は，次のようにカスタムアダプターを開発するための API を提供します。

表 7-5　インタフェース API

パッケージ	インタフェース名	説明
jp.co.Hitachi.soft.sdp.api.distributable	SDPClient	パラレル SDP サーバの入力ストリームへの接続を管理します。
jp.co.Hitachi.soft.sdp.api.distributable	StreamParallelInput	パラレル SDP サーバの入力ストリームにタプルを送信します。

表 7-6　クラス API

パッケージ	クラス名	説明
jp.co.Hitachi.soft.sdp.api.distributable	SDPClientFactory	パラレル SDP サーバに接続します。

ほかの API の詳細については，「8.　データ送受信用 API」を参照してください。

(2)　SDPClient

openStreamInput ()

説明

パラレル SDP サーバの入力ストリームに接続します。

形式

```
public StreamParallelInput openStreamInput (String groupName, String streamName)
throws SDPClientException
```

パラメーター

groupName

パラレル SDP サーバの CQL に登録されているクエリグループの名前を設定します。

streamName

パラレル SDP サーバの CQL に登録されている入力ストリームの名前を設定します。

例外

SDPClientException

次の場合に例外がスローされます。

- クエリグループまたは入力ストリームが，パラレル SDP サーバの CQL に登録されていない場合
- 同じ入力ストリームがすでに接続されている場合
- SDPClient.destroy()がすでに実行されている場合
- AP と SDP サーバ間の通信が切断されている場合

destroy ()

説明

パラレル SDP サーバから切断します。

形式

```
public void destroy ()
throws SDPClientException
```

パラメーター

なし。

例外

SDPClientException

次の場合に例外がスローされます。

- SDPClient.destroy ()がすでに実行されている場合
- AP と SDP サーバ間の通信が切断されている場合

(3) StreamParallelInput

put ()

説明

パラレル SDP サーバの入力ストリームにタプルを送信します。

形式

```
public void put (String key, StreamTuple tuple)
throws SDPClientException
```

パラメーター

key

コンシステントハッシュ法を使用して宛先のパラレル SDP サーバを決定するためのキーを設定します。

パラメーターの範囲：

文字列の長さは，1～1,024 の ASCII 文字です。

ASCII 文字は 0x20～0x7E です（ただし，0x3A（:）は除く）。

tuple

パラレル SDP サーバに送信するタプルを設定します。

例外

SDPClientException

次の場合に例外がスローされます。

- SDPClient.destroy()がすでに実行されている場合
- 入力ストリームがすでに閉じている場合
- パラメーターが null の場合
- パラメーターが有効範囲外の場合
- クエリグループがホールド状態の場合
- クエリグループがすでに削除されている場合
- クエリグループが停止状態の場合
- AP と SDP サーバ間の通信が切断されている場合

putEnd ()

説明

パラレル SDP サーバの入力ストリームにタプルの終了を送信します。

形式

```
public void putEnd ()
throws SDPClientException
```

パラメーター

なし。

例外

SDPClientException

次の場合に例外がスローされます。

- SDPClient.destroy ()がすでに実行されている場合
- 入力ストリームがすでに閉じている場合

close ()

説明

パラレル SDP サーバの入力ストリームから切断します。

形式

```
public void close ()
throws SDPClientException
```

パラメーター

なし。

例外

SDPClientException

次の場合に例外がスローされます。

- SDPClient.destroy ()がすでに実行されている場合
- 入力ストリームがすでに閉じている場合
- クエリグループがホールド状態の場合
- クエリグループがすでに削除されている場合
- クエリグループが停止状態の場合
- AP と SDP サーバ間の通信が切断されている場合

(4) SDPClientFactory

create ()

説明

パラレル SDP サーバに接続します。

形式

```
public static SDPClient create (String confFileName, String clientName, int unit, int bufferSize, long waitTime, int retryCnt, int retryInterval)
throws SDPClientException
```

パラメーター

confFileName

クライアント定義ファイルの絶対パスを指定するか，Hitachi Streaming Data Platform の運用ディレクトリからの相対パスを指定します。

clientName

クライアントを特定する名前を指定します。

各クライアントで一意の名前を指定してください。

パラメーターの範囲：

1～32 文字の英数字とアンダースコアです。

最初の文字は英数字である必要があります。

unit

SDP サーバに送信するタプルの単位（グループ）を設定します。このパラメーターは，1 回設定します。

パラメーターの範囲：

1～4,096 の整数です。

bufferSize

タプルを SDP サーバに送信する前に，AP がタプルのグループを一時的に配置するバッファのサイズを設定します。

パラメーターの範囲：

1～1,048,576 の整数です。

waitTime

送信するタプルのグループが格納されるまで待機する時間を，ミリ秒単位で設定します。

この時間が経過したときに，送信するタプルのグループが格納されていない場合，この時間のあとにタプルが SDP サーバに送信されます。

パラメーターの範囲：

0～60,000 の整数です。

retryCnt

SDP サーバのキューがいっぱいになったときにタプルの送信をリトライする回数を指定します。

パラメーターの範囲：

0～10,000 の整数です。

retryInterval

SDP サーバのキューがいっぱいになったときにタプルの送信をリトライする間隔を，ミリ秒単位で指定します。

パラメーターの範囲：

1～1,000 の整数です。

例外

`SDPClientException`

次の場合に例外がスローされます。

- パラメーターが null または空文字の場合
- パラメーターが有効範囲外の場合
- AP と SDP サーバ間の通信が切断されている場合

8 データ送受信用 API

この章では，カスタムアダプターの作成で使用するデータ送受信用 API の文法について説明します。

8.1　データ送受信用 API の記述形式

ここでは，データ送受信用 API の記述形式について説明します。

各 API で説明する項目は次のとおりです。ただし，API によっては説明しない項目もあります。

形式

API の記述形式を示します。

説明

API の機能について説明します。

パラメーター

API のパラメーターについて説明します。

例外

API を利用する際に発生する例外について説明します。

戻り値

API の戻り値について説明します。

注意事項

API を利用する上での注意事項について説明します。

8.2　データ送受信用 API の一覧

データ送受信用 API のインタフェース，クラス，および列挙型の一覧を示します。

表 8-1　データ送受信用 API 一覧（インタフェース）

項番	パッケージ名	インタフェース名	説明
1	jp.co.Hitachi.soft.sdp.api	SDPConnector	SDP サーバとカスタムアダプターを結ぶコネクターのインタフェースです。入力ストリームへ接続する場合，または出力ストリームへ接続する場合に使用します。
2		StreamInput	SDP サーバにストリームデータを送信する場合，および入力ストリームとの接続を切断する場合に使用するインタフェースです。
3		StreamOutput	SDP サーバからのクエリ結果データの受信，コールバック用リスナーオブジェクトの登録・解除，および出力ストリームとの接続を閉じるために使用するインタフェースです。
4	jp.co.Hitachi.soft.sdp.api.inprocess	StreamEventListener	コールバックされるメソッドの処理を実装するためのインタフェースです。
5		StreamInprocessUP	カスタムアダプターの main メソッドに相当するメイン処理を実装するためのインタフェースです。SDP サーバ起動時に StreamInprocessUP インタフェースの実装クラスがロードされ，実行されます。
6	jp.co.Hitachi.soft.sdp.api.logger	SDPClientLogger	ログの出力に関する機能を提供するクラスのインタフェースです。

表 8-2　データ送受信用 API 一覧（クラス）

項番	パッケージ名	クラス名	説明
1	jp.co.Hitachi.soft.sdp.api.logger	SDPClientLoggerFactory	SDPClientLogger インタフェースを取得するためのファクトリクラスです。
2	jp.co.Hitachi.soft.sdp.common.util	StreamTime	日付・時刻指定処理のためのクラスです。
3	jp.co.Hitachi.soft.sdp.common.data	StreamTuple	ストリームタプルを表現するクラスです。

表 8-3　データ送受信用 API 一覧（列挙型）

項番	パッケージ名	列挙型	説明
1	jp.co.Hitachi.soft.sdp.api.logger.util	LogKind	ログ種別を定義した列挙型です。

以降では，インタフェース，クラス，および列挙型をアルファベット順に説明します。

8.3 SDPClientLogger インタフェース

説明

ログの出力に関する機能を提供するクラスのインタフェースです。

メソッド

SDPClientLogger インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
void	initialize(String *id*)	ロガーを初期化します。
void	putMessage(LogKind *kind*, String *messageString*)	内部カスタムアダプター用メッセージログファイルに，メッセージログを出力します。
void	putStackTrace(Throwable *t*)	内部カスタムアダプター用スタックトレースログファイルに，トレースログを出力します。
void	terminate()	ロガーの終了処理を実行します。

8.3.1 initialize(String id)メソッド

形式

```
public void initialize (String id)
```

説明

ロガーを初期化します。このメソッドを実行すると，ログファイルにメッセージを出力できるようになります。

パラメーター

id

内部カスタムアダプターログファイル用の識別子を指定します。

識別子は内部カスタムアダプターログファイル名の一部として使われます。識別子には，1～32 文字の半角英数字とアンダースコア（_）だけを使用できます。先頭には半角英数字だけが指定できます。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
IllegalArgumentException	パラメーターに指定できない文字列を指定した場合
SDPClientInitializeException	ロガーの初期化に失敗した場合
NullPointerException	パラメーターにnull を指定した場合

戻り値

なし。

注意事項

- このメソッドは 1 つのロガーインスタンスに対して 1 回だけ実行してください。複数回実行した場合の動作は保証されません。
- 内部カスタムアダプターログファイル用の識別子には，アダプターグループ間でログファイルが重複しないアダプターグループ名を指定してください。
- カスタムアダプター内でスレッドを利用する場合など，複数のクラスから同一のログファイルに書き込みたいときは，このメソッド実行後の同一のインスタンスを，それぞれのクラスから利用する実装にしてください。

8.3.2 putMessage(LogKind kind, String messageString)メソッド

形式

```
public void putMessage(LogKind kind, String messageString)
```

説明

SDP サーバのログファイル出力用プロパティファイル（logger.properties）のlogger.logfileDir パラメーターに指定したディレクトリに，メッセージログを出力します。

パラメーター

kind

メッセージのログ種別を指定します。

指定するログ種別の値に応じて，次に示すメッセージ ID でメッセージログを出力します。

パラメーターの指定値	メッセージ ID
INFO	KFSP82004-I
WARNING	KFSP42047-W
ERROR	KFSP42048-E

messageString

ログファイルに出力するメッセージ本文を指定します。

メッセージ本文は，バイト列に変換した場合に 4,095 バイト以下になる長さにしてください。4,095 バイトを超える場合は 4,095 バイトまでがメッセージ本文に使用されます。

例外

なし。

戻り値

なし。

注意事項

- メッセージログファイルへのメッセージの書き込みに失敗した場合，メッセージは出力されません。
- メッセージ本文に指定する文字列の文字コードにはASCII だけを使用できます。その他の文字コードの文字列をパラメーターに指定した場合は，意図した出力結果とならないことがあります。
- *kind* パラメーターにnull を指定した場合は，メッセージは出力されません。

8.3.3 putStackTrace(Throwable t)メソッド

形式

```
public void putStackTrace(Throwable t)
```

説明

SDP サーバのログファイル出力用プロパティファイル（logger.properties）のlogger.logfileDir パラメーターに指定したディレクトリに，トレースログ（例外情報）を出力します。

パラメーター

t

java.lang.Throwable クラスを継承した任意の例外クラスを指定します。指定された例外クラスのスタックトレース情報は，Throwable クラスのprintStackTrace()メソッドで表示される改行文字を含む文字列のうち 4,071 バイトまでを使用してトレースログに出力します。トレースログにスタックトレース情報を出力する際のメッセージ ID はKFSP81099-E となります。

例外

なし。

戻り値

なし。

注意事項

- ログファイルへのメッセージの書き込みに失敗した場合，メッセージは出力されません。
- *t* パラメーターにnull を指定した場合，「Argument is null」を埋め字とした，NullPointerException のスタックトレース情報がメッセージとして出力されます。

8.3.4 terminate()メソッド

形式

```
public void terminate()
```

説明

ロガーを終了します。

パラメーター

なし。

例外

なし。

戻り値

なし。

注意事項

- このメソッドは，ロガーを使用したプログラムの終了処理として，initialize(String *id*)メソッドを実行したインスタンスに対して必ず実行してください。
- このメソッドは，1 つのロガーインスタンスに対して 1 回だけ実行してください。複数回実行した場合の動作は保証されません。

8.4 SDPConnector インタフェース

説明

SDP サーバとの接続で使用するコネクターとしての機能を持つインタフェースです。コネクターは，SDPConnector 型オブジェクトとして生成されます。

SDP サーバに登録されているクエリグループの入力ストリームまたは出力ストリームには，コネクターを使用して接続します。

ストリームには，SDPConnector インタフェースのopenStreamInput メソッドまたはopenStreamOutput メソッドを使用して接続します。ただし，1 つのコネクターで同一名称のストリームに複数回接続することはできません。

コネクターは，SDP サーバとカスタムアダプター間のデータ連携の方法ごとに，次の処理によって生成されます。

- SDP サーバとインプロセスで連携するコネクターの生成

  hsdpstartinpro コマンド実行時に，SDP サーバとインプロセスで連携したコネクターが生成されます。インプロセス連携カスタムアダプターでは，生成されたSDPConnector 型オブジェクトを StreamInprocessUP インタフェースのexecute メソッドのパラメーターとして受け取ります。

フィールド

なし。

コンストラクター

なし。

メソッド

SDPConnector インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
void	close()	コネクターを閉じます。
boolean	isClosed()	コネクターが閉じているかどうかを確認します。
StreamInput	openStreamInput(String *group_name*,String *stream_name*)	パラメーターに指定したグループ名およびストリーム名に対応する入力ストリームに接続します。 同一のSDPConnector 型オブジェクトでは，同一名称のストリームに複数回接続できません。
StreamOutput	openStreamOutput(String *group_name*,String *stream_name*)	パラメーターに指定したグループ名およびストリーム名に対応する出力ストリームへ接続します。同一のSDPConnector 型オブジェクトでは，同一名称のストリームに複数回接続できません。

注意事項

なし。

8.4.1　close()メソッド

形式

```
void close()
```

説明

コネクターを閉じます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	コネクターがすでに閉じていた場合

戻り値

なし。

8.4.2　isClosed()メソッド

形式

```
boolean isClosed()
```

説明

コネクターが閉じているかどうかを確認します。

パラメーター

なし。

例外

なし。

戻り値

`true`

コネクターは閉じています。

`false`

コネクターは開いています。

8.4.3 openStreamInput(String group_name,String stream_name)メソッド

形式

```
StreamInput openStreamInput(String group_name,String stream_name)
```

説明

パラメーターに指定したグループ名およびストリーム名に対応する入力ストリームに接続します。同一のSDPConnector 型オブジェクトでは，同一名称のストリームに複数回接続できません。

パラメーター

group_name

クエリグループ名を指定します。

stream_name

ストリーム名を指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合
SDPClientException	• 存在しないグループ名またはストリーム名を指定した場合 • 同じSDPConnector 型オブジェクトを使用して，同一の入力ストリームとすでに接続している場合 • 入力ストリームではないストリームを指定した場合 • コネクターがすでに閉じている場合

戻り値

接続した入力ストリーム（StreamInput 型オブジェクト）。

8.4.4 openStreamOutput(String group_name,String stream_name)メソッド

形式

```
StreamOutput openStreamOutput(String group_name,String stream_name)
```

説明

パラメーターに指定したグループ名およびストリーム名に対応する出力ストリームに接続します。同一の SDPConnector 型オブジェクトでは，同一名称のストリームに複数回接続できません。

パラメーター

group_name

クエリグループ名を指定します。

stream_name

ストリーム名を指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合
SDPClientException	• 存在しないグループ名またはストリーム名を指定した場合 • 同じSDPConnector 型オブジェクトを使用して，同一の出力ストリームにすでに接続をしている場合 • コネクターがすでに閉じている場合
SDPClientRelationStateException	指定したストリームがリレーション状態である場合

戻り値

接続した出力ストリーム（StreamOutput 型オブジェクト）。

8.5 StreamEventListener インタフェース

説明

SDP サーバ上で，タプルが生成されたときにコールバックされるメソッドの処理を実装するインタフェースです。

カスタムアダプターでは，StreamEventListener インタフェースを実装したクラスを作成して，onEvent メソッドにコールバック時の処理を記述します。

StreamOutput インタフェースのregisterForNotification メソッドを使用して，StreamEventListener インタフェースを実装したクラスのオブジェクトを登録すると，以降，SDP サーバ上でタプルが生成されたときに，登録したオブジェクトのonEvent メソッドがコールバックされます。

メソッド

StreamEventListener インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
void	onEvent(StreamTuple *tuple*)	SDP サーバ上でタプルが生成されたときにコールバックされるメソッドです。

注意事項

StreamEventListener インタフェースはjava.rmi.Remote クラスを継承しています。

8.5.1 onEvent(StreamTuple tuple)メソッド

形式

```
void onEvent(StreamTuple tuple)
```

説明

SDP サーバ上でタプルが生成されたときにコールバックされるメソッドです。

パラメーター

tuple

生成されたタプルが指定されます。

例外

なし。

戻り値

なし。

8.6 StreamInprocessUP インタフェース

説明

カスタムアダプターが SDP サーバとインプロセスで連携するためには，StreamInprocessUP インタフェースを実装するクラスを作成する必要があります。

StreamInprocessUP インタフェースには，execute メソッドとstop メソッドがあります。

execute メソッドには，カスタムアダプターのmain メソッドに相当するメイン処理を記述します。stop メソッドには，カスタムアダプターの終了処理を記述します。

hsdpstartinpro コマンドを実行してインプロセス連携カスタムアダプターを起動すると，SDP サーバによってStreamInprocessUP インタフェースを実装したクラスがロードされ，execute メソッドを実行するためのスレッド（カスタムアダプター実行スレッド）が生成されます。execute メソッドはこのスレッドから呼び出されるメソッドです。なお，SDP サーバによってロードされる実装クラス名，およびその実装クラスが格納されたjar ファイルのパス名は，プロパティファイルuser_app.*インプロセス連携カスタムアダプター名*.properties に記述します。プロパティファイルについては，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

stop メソッドは，hsdpstopinpro コマンドを実行してインプロセス連携カスタムアダプターを停止するときに，SDP サーバ側のスレッドから呼び出されます。

メソッド

StreamInprocessUP インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
void	execute(SDPConnector *con*)	カスタムアダプターのメイン処理を記述します。 このメソッドは，hsdpstartinpro コマンド実行時に，SDP サーバが生成したカスタムアダプター実行スレッドによって呼び出されます。
void	stop()	カスタムアダプターの終了処理を記述します。 このメソッドは，hsdpstopinpro コマンド実行時に，SDP サーバ側のスレッドによって呼び出されます。

注意事項

- StreamInprocessUP インタフェースのexecute メソッドで起動したユーザースレッドは，stop メソッドの中で必ず停止してください。stop メソッドに停止処理を記述しなかった場合，またはstop メソッドで停止できない無限ループなどの処理がユーザースレッドに含まれる場合，ユーザースレッドが停止されないで残ることがあります。
- StreamInprocessUP インタフェースのexecute メソッドに無限ループなどの処理を記述した場合，execute メソッド自身の実行スレッドが残ってしまうことがあります。

- カスタムアダプターに任意の引数を渡す場合は，String 型の可変長引数を持つコンストラクターを実装してください。リフレクション経由で作成した String 型の可変長引数を持つコンストラクターが呼ばれます。その引数として，hsdpstartinpro コマンドで定義した任意の引数が渡されます。なお，String 型の可変長引数を持つコンストラクターを実装していない場合は，デフォルトコンストラクターが呼ばれます（ただし，引数は渡されません）。
- カスタムアダプターに任意の引数を渡さない場合は，String 型の可変長引数を持つコンストラクターまたはデフォルトコンストラクターのどちらかを実装してください。どちらも実装している場合は，String 型の可変長引数を持つコンストラクターが呼ばれます。
- String 型の可変長引数を持つコンストラクターまたはデフォルトコンストラクターのどちらも実装していない場合は，エラーとなり，カスタムアダプターは起動しません。

8.6.1 execute(SDPConnector con)メソッド

形式

```
void execute(SDPConnector con)
```

説明

カスタムアダプターのメイン処理を記述します。

このメソッドは，hsdpstartinpro コマンド実行時に，SDP サーバが生成したカスタムアダプター実行スレッドによって呼び出されます。

パラメーター

con

SDPConnector 型オブジェクトを指定します。

例外

なし。

戻り値

なし。

8.6.2　stop()メソッド

形式

```
void stop()
```

説明

カスタムアダプターの終了処理を記述します。

このメソッドは，hsdpstopinpro コマンド実行時に，SDP サーバ側のスレッドによって呼び出されます。

パラメーター

なし。

例外

なし。

戻り値

なし。

8.7 StreamInput インタフェース

説明

カスタムアダプターが SDP サーバに対してタプルを送信するために使用するインタフェースです。

タプルを送信するには，put メソッドを使用します。put メソッドには，複数のタプルをまとめて送信するメソッドと，単一のタプルを送信するメソッドの 2 種類があります。

メソッド

StreamInput インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
void	close()	入力ストリームとの接続を閉じます。
int	getFreeQueueSize()	入力ストリームキューの空きサイズを取得します。
int	getMaxQueueSize()	入力ストリームキューの最大サイズを取得します。
SuspendStatus	getSuspendStatus()	データソースモードで実行中の入力ストリームのステータスを取得します。
boolean	isStarted()	データソースモードの場合に，入力ストリームが新たなストリームデータの受付を開始しているかどうかを確認します。
void	put(ArrayList<StreamTuple> *tuple_list*)	複数のタプルを送信します。
void	put(StreamTuple *tuple*)	単一のタプルを送信します。
void	putControl(Object... *args*)	サーバモードの場合に，制御タプルを送信します。制御タプルに任意の数のオブジェクトを付与できます。時刻制御には SDP サーバに到達した時刻が使用されます。
void	putControl(Timestamp *ts*, Object... *args*)	データソースモードの場合に，制御タプルを送信します。制御タプルに任意の数のオブジェクトを付与できます。時刻制御には引数に設定した*ts* が使用されます。 サーバモードの場合に使用すると，設定した*ts* の値が SDP サーバ到達時の時刻で上書きされます。
void	putEnd()	入力ストリームに対して，ストリームデータ入力が終了したことを通知します。 このメソッドによって入力完了を通知したあと，クエリグループの再開前にput メソッドやputEnd メソッドを実行した場合は，例外が返されます。
void	putHeartbeat()	データソースモードで実行中の入力ストリーム内にハートビートタプルを置きます。

戻り値	メソッド名	説明
void	resume()	データソースモードで実行中の入力ストリームを再開します。
void	suspend()	データソースモードで実行中の入力ストリームを一時停止します。タイムスタンプ調整機能が有効な場合，入力ストリームが一時中断される前に，タイムスタンプの調整に必要な予約済みタプルを消去します。

注意事項

なし。

8.7.1 close()メソッド

形式

```
void close()
```

説明

入力ストリームとの接続を閉じます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	入力ストリームがすでに閉じていた場合

戻り値

なし。

8.7.2 getFreeQueueSize()メソッド

形式

```
int getFreeQueueSize()
```

説明

入力ストリームキューの空きサイズを取得します。

入力ストリームキューの空きサイズは，system_config.properties のengine.maxQueueSize パラメーターに指定した値から，使用中のサイズを引いた値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	入力ストリームがすでに閉じている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合（SDPClientQueryGroupStateException の詳細例外）
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合（SDPClientQueryGroupStateException の詳細例外）

戻り値

入力ストリームキューの空きサイズ（int）。

8.7.3 getMaxQueueSize()メソッド

形式

```
int getMaxQueueSize()
```

説明

入力ストリームキューの最大サイズを取得します。

入力ストリームキューの最大サイズは，system_config.properties のengine.maxQueueSize パラメーターに指定した値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	入力ストリームがすでに閉じている場合

戻り値

入力ストリームキューの最大サイズ（int）。

8.7.4 getSuspendStatus()メソッド

形式

```
SuspendStatus getSuspendStatus()
```

説明

データソースモードで実行中の入力ストリームのステータスを取得します。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがクローズしています。 • 入力ストリームはストリームデータの受信を停止します。
SDPClientServerModeException	入力ストリームがサーバモードで実行中です。

例外	発生条件
SDPClientQueryGroupHoldException	クエリグループが保留されています。
SDPClientQueryGroupNotExistException	クエリグループが削除されています。
SDPClientQueryGroupStateException	クエリグループが実行されていません。
SDPClientQueryGroupStopException	クエリグループが停止されています。

戻り値

NOT_SUSPEND

入力ストリームが一時停止していません。

SUSPEND

入力ストリームが一時停止しています（ストリームは自動的に再開しません）。

SUSPEND_AUTO_RESUME

入力ストリームが一時停止しています（ストリームが自動的に再開します）。

8.7.5 isStarted()メソッド

形式

```
boolean isStarted()
```

説明

データソースモードの場合に，入力ストリームが新たなストリームデータの受付を開始しているかどうかを確認します。

パラメーター

なし。

例外

例外の一覧を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがすでに閉じている場合 • サーバモードで動作中のストリームの場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合

例外	発生条件
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

true
　ストリームデータの受付を開始しています。

false
　ストリームデータの受付を開始していません。

8.7.6　put(ArrayList<StreamTuple> tuple_list)メソッド

形式

```
void put(ArrayList<StreamTuple> tuple_list)
```

説明

複数のタプルを送信します。

パラメーター

tuple_list
　送信するタプル（StreamTuple 型オブジェクト）のリストを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
ClassCastException	パラメーターのリスト内にStreamTuple 以外の要素が含まれていた場合
NullPointerException	• パラメーターにnull またはnull を含むリストを指定した場合 • リストの要素であるデータオブジェクト配列内にnull を含むStreamTuple 型オブジェクトを指定した場合
SDPClientException	• タプルのリストに含まれる要素のデータ型がストリーム定義と異なる場合 • 入力ストリームがすでに閉じていた場合
SDPClientFreeInputQueueSizeLackException	入力ストリームキューの空きサイズが不足している場合（この場合，入力ストリームキューにタプルは投入されません）

例外	発生条件
SDPClientFreeInputQueueSizeThresholdOverException	入力ストリームキューの空きサイズがstream.freeInputQueueSizeThresholdパラメーターで指定したしきい値以下になった場合（この場合，入力ストリームキューにタプルは投入されます）
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

なし。

8.7.7 put(StreamTuple tuple)メソッド

形式

```
void put(StreamTuple tuple)
```

説明

単一のタプルを送信します。

パラメーター

tuple

送信するタプル（StreamTuple 型オブジェクト）を指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	• パラメーターにnull を指定した場合 • データオブジェクト配列内にnull を含むStreamTuple 型オブジェクトを指定した場合
SDPClientException	• タプルの要素のデータ型がストリーム定義と異なる場合 • 入力ストリームがすでに閉じていた場合
SDPClientFreeInputQueueSizeLackException	入力ストリームキューの空きサイズが不足している場合（この場合，入力ストリームキューにタプルは投入されません）

例外	発生条件
SDPClientFreeInputQueueSizeThresholdOverException	入力ストリームキューの空きサイズがstream.freeInputQueueSizeThresholdパラメーターで指定したしきい値以下になった場合（この場合，入力ストリームキューにタプルは投入されます）
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

なし。

8.7.8 putControl(Object... args)メソッド

形式

```
void putControl(Object... args)
```

説明

サーバモードの場合に，制御タプルを送信します。制御タプルに任意の数のオブジェクトを付与できます。時刻制御には SDP サーバに到達した時刻が使用されます。

データソースモードの場合，時刻制御には制御タプルが SDP サーバに到達した時刻が使用されます。

パラメーター

args

制御タプルに，任意に設定できるオブジェクト（Object 型オブジェクト）を指定します。設定するオブジェクトには，「3.3.1　CQL のデータ型と Java のデータ型のマッピング」に記載された Java のデータ型だけを使用してください。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがすでに閉じている場合 • ストリームデータの受付を停止している場合 • Object 型の引数に Java の基本データ型以外の型を指定している場合

例外	発生条件
SDPClientFreeInputQueueSizeLackException	入力ストリームキューの空きサイズが不足している場合（この場合，入力ストリームキューにタプルは投入されません）
SDPClientFreeInputQueueSizeThresholdOverException	入力ストリームキューの空きサイズがstream.freeInputQueueSizeThresholdパラメーターで指定したしきい値以下になった場合（この場合，入力ストリームキューにタプルは投入されます）
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合
NullPointerException	パラメーターにnull を指定した場合

戻り値

なし。

注意事項

特になし。

8.7.9 putControl(Timestamp ts, Object... args)メソッド

形式

```
void putControl(Timestamp ts, Object... args)
```

説明

データソースモードの場合に，制御タプルを送信します。制御タプルに任意の数のオブジェクトを付与できます。時刻制御には引数に設定した*ts* が使用されます。

サーバモードの場合に使用すると，設定した*ts* の値が SDP サーバ到達時の時刻で上書きされます。

パラメーター

ts

時刻制御に使用するタイムスタンプ情報（Timestamp 型オブジェクト）です。

args

制御タプルに，任意に設定できるオブジェクト（Object 型オブジェクト）を指定します。設定するオブジェクトには，「3.3.1　CQL のデータ型と Java のデータ型のマッピング」に記載された Java のデータ型だけを使用してください。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがすでに閉じている場合 • ストリームデータの受付を停止している場合 • Object 型の引数に Java の基本データ型以外の型を指定している場合
SDPClientFreeInputQueueSizeLackException	入力ストリームキューの空きサイズが不足している場合（この場合，入力ストリームキューにタプルは投入されません）
SDPClientFreeInputQueueSizeThresholdOverException	入力ストリームキューの空きサイズがstream.freeInputQueueSizeThresholdパラメーターで指定したしきい値以下になった場合（この場合，入力ストリームキューにタプルは投入されます）
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合
NullPointerException	パラメーターにnull を指定した場合

戻り値

なし。

注意事項

特になし。

8.7.10　putEnd()メソッド

形式

```
void putEnd()
```

説明

入力ストリームに対して，ストリームデータの入力が終了したことを通知します。

このメソッドによって入力完了を通知したあと，クエリグループの再開前にput メソッドやputEnd メソッドを実行した場合は，例外が返されます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがすでに閉じている場合 • 入力ストリームがすでにストリームデータの受付を停止している場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	クエリグループが停止中の場合

戻り値

なし。

8.7.11　putHeartbeat()メソッド

形式

```
void putHeartbeat(Timestamp ts)
```

説明

データソースモードで実行中の入力ストリーム内にハートビートタプルを置きます。

パラメーター

ts

ハートビートタプルにタイムスタンプを設定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターを null として設定します。
SDPClientException	• 入力ストリームがクローズしています。 • 入力ストリームはストリームデータの受信を停止します。
SDPClientFreeInputQueueSizeLackException	入力ストリームキューに十分な空きサイズがありません。 そのため，ハートビートタプルは入力ストリームに配置されません。
SDPClientFreeInputQueueSizeThresholdOverException	入力ストリームキューの空きサイズが，しきい値※以下です。
SDPClientServerModeException	入力ストリームがサーバモードで実行中です。
SDPClientSuspendException	入力ストリームが一時停止しています (ストリームは自動的に再開しません)。
SDPClientQueryGroupHoldException	クエリグループが保留されています。
SDPClientQueryGroupNotExistException	クエリグループが削除されています。
SDPClientQueryGroupStateException	クエリグループが実行されていません。
SDPClientQueryGroupStopException	クエリグループが停止されています。
注※ しきい値は，stream.freeInputQueueSizeThreshold パラメーターで指定されます。この結果，ハートビートタプルが入力ストリームに配置されます。	

戻り値

なし。

8.7.12 resume()メソッド

形式

```
void resume()
```

説明

データソースモードで実行中の入力ストリームを再開します。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがクローズしています。 • 入力ストリームはストリームデータの受信を停止します。
SDPClientServerModeException	入力ストリームがサーバモードで実行中です。
SDPClientQueryGroupHoldException	クエリグループが保留されています。
SDPClientQueryGroupNotExistException	クエリグループが削除されています。
SDPClientQueryGroupStateException	クエリグループが実行されていません。
SDPClientQueryGroupStopException	クエリグループが停止されています。

戻り値

なし。

8.7.13 suspend()メソッド

形式

```
void suspend(boolean autoResume)
```

説明

データソースモードで実行中の入力ストリームを一時停止します。タイムスタンプ調整機能が有効な場合，入力ストリームが一時中断される前に，タイムスタンプの調整に必要な予約済みタプルを消去します。

パラメーター

autoResume

入力ストリームを自動的に再開するかどうかを指定します。

- true：自動的に再開
- false：手動による再開

例外

例外とその発生条件を次の表に示します。

例外	生成条件
SDPClientException	• 入力ストリームがクローズしています。 • 入力ストリームはストリームデータの受信を停止します。
SDPClientServerModeException	入力ストリームがサーバモードで実行中です。
SDPClientQueryGroupHoldException	クエリグループが保留されています。
SDPClientQueryGroupNotExistException	クエリグループが削除されています。
SDPClientQueryGroupStateException	クエリグループが実行されていません。
SDPClientQueryGroupStopException	クエリグループが停止されています。

戻り値

なし。

8.8 StreamOutput インタフェース

説明

カスタムアダプターが SDP サーバからタプルを受信するために使用するインタフェースです。

タプルを受信するための API には，次の 2 種類があります。

- ポーリング用メソッド
- コールバック用メソッド

ポーリング用メソッドは，カスタムアダプターが SDP サーバから能動的にタプルを取得するためのメソッドです。get メソッドとgetAll メソッドがあります。get メソッドは，単一のタプルを受信するメソッドです。getAll メソッドは，複数のタプルをまとめて受信するメソッドです。

コールバック用メソッドは，SDP サーバ上でタプルが生成されたときに呼び出され，受動的にタプルを取得するためのメソッドです。コールバック方式でデータを受信するには，SDP サーバ上でタプルが生成されたときに SDP サーバから呼び出されるメソッドを持つオブジェクト（**リスナーオブジェクト**）を SDP サーバに登録しておく必要があります。

リスナーオブジェクトは，StreamEventListener インタフェースを実装したオブジェクトです。StreamOutput インタフェースには，このリスナーオブジェクトを SDP サーバに登録するためのメソッドとして，registerForNotification メソッドがあります。また，リスナーオブジェクトの SDP サーバへの登録を解除するためのメソッドとして，unregisterForNotification メソッドがあります。

メソッド

StreamOutput インタフェースのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
void	close()	出力ストリームとの接続を閉じます。
StreamTuple	get()	SDP サーバに登録されているタプルを 1 つだけ取得します。
ArrayList<StreamTuple>	get(int *count*)	SDP サーバに登録されているデータをcount パラメーターに指定した数だけ取得します。
ArrayList<StreamTuple>	get(int *count*, long *timeout*)	SDP サーバに登録されているデータをcount パラメーターに指定した数だけ取得します。SDP サーバに結果データがない場合，結果データが到着するか，またはtimeout パラメーターに指定した時間が経過するまで待機します。
ArrayList<StreamTuple>	getAll()	SDP サーバに登録されているすべてのタプルを取得します。
ArrayList<StreamTuple>	getAll(long *timeout*)	SDP サーバに登録されているすべてのタプルを取得します。SDP サーバに結果データがない場合，結果データが到着するか，またはtimeout パラメーターに指定した時間が経過するまで待機します。

戻り値	メソッド名	説明
int	getFreeQueueSize()	出力ストリームキューの空きサイズを取得します。
int	getMaxQueueSize()	出力ストリームキューの最大サイズを取得します。
void	registerForNotification(StreamEventListener *n*)	コールバック用リスナーオブジェクトを登録します。
void	unregisterForNotification(StreamEventListener *n*)	登録したリスナーオブジェクトを解除して，以降のコールバック処理の実行を解除します。

注意事項

なし。

8.8.1 close()メソッド

形式

```
void close()
```

説明

出力ストリームとの接続を閉じます。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	出力ストリームがすでに閉じている場合

戻り値

なし。

8.8.2 get()メソッド

形式

```
StreamTuple get()
```

説明

SDP サーバに登録されているタプルを 1 つだけ取得します。

出力ストリームキューにタプルがない場合は，null が返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合，またはコマンドによってクエリグループ停止が通知された場合），次のように処理されます。

- 停止通知後，最初のメソッド呼び出しの場合
  null が返却されます。
- 停止通知後，2 回目以降のメソッド呼び出しの場合
  例外がスローされます。

このメソッドはポーリング用メソッドです。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	• 出力ストリームがすでに閉じている場合 • コールバック用リスナーオブジェクトがすでに登録されている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中ではない場合
SDPClientQueryGroupStopException	結果データがなく，クエリグループが停止中の場合

戻り値

取得したタプル（StreamTuple 型オブジェクト）。

8.8.3　get(int count)メソッド

形式

```
ArrayList<StreamTuple> get(int count)
```

説明

SDP サーバに登録されているデータをcount パラメーターに指定した数だけ取得します。

SDP サーバに結果データがない場合，空のArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合，またはコマンドによってクエリグループ停止が通知された場合），次のように処理されます。

- 停止通知後，最初のメソッド呼び出しの場合
  空のArrayList 型オブジェクトが返却されます。
- 停止通知後，2 回目以降のメソッド呼び出しの場合
  例外がスローされます。

このメソッドは，ポーリング用メソッドです。

パラメーター

count

SDP サーバから取得するデータの数を指定します。指定可能範囲は1～1,048,576 です。

なお，実際に取得できるデータの数は，このパラメーターと出力ストリームキューの状態によって次に示すようになります。

条件	取得できる数
count の指定値 ≦ 出力ストリームキューのデータ数	*count* パラメーターに指定した数
count の指定値 ＞ 出力ストリームキューのデータ数	出力ストリームキューのデータ数
count の指定値 ＞ 出力ストリームキューの最大サイズ	

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	• 出力ストリームがすでに閉じている場合

例外	発生条件
SDPClientException	• コールバック用リスナーオブジェクトがすでに登録されている場合 • パラメーターが不正の場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合（SDPClientQueryGroupStateException の詳細例外）
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく，かつクエリグループが停止中の場合（SDPClientQueryGroupStateException の詳細例外）

戻り値

タプルのリスト（ArrayList<StreamTuple>型オブジェクト）。

8.8.4 get(int count, long timeout)メソッド

形式

```
ArrayList<StreamTuple> get(int count, long timeout)
```

説明

SDP サーバに登録されているデータを *count* パラメーターに指定した数だけ取得します。

SDP サーバに結果データがない場合，結果データが到着するか，または *timeout* パラメーターに指定した時間が経過するまで待機します。待機中に結果データが到着した場合，到着したデータを格納したArrayList型オブジェクトが返却されます。*timeout* パラメーターに指定した時間が経過した場合，または待機中のスレッドに割り込みが発生した場合は，次のどちらかのオブジェクトが返却されます。

- データが登録されている場合
  登録されたデータが格納されたArrayList 型オブジェクトが返却されます。
- データが登録されていない場合
  空のArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合，またはコマンドによってクエリグループ停止が通知された場合），次のように処理されます。

- 停止通知後，最初のメソッド呼び出しの場合
  空のArrayList 型オブジェクトが返却されます。

- 停止通知後，2 回目以降のメソッド呼び出しの場合
  例外がスローされます。

このメソッドは，ポーリング用メソッドです。

パラメーター

count

SDP サーバから取得するデータの数を指定します。指定可能範囲は1～1,048,576 です。

なお，実際に取得できるデータの数は，このパラメーターと出力ストリームキューの状態によって次に示すようになります。

条件	取得できる数
count の指定値 ≦ 出力ストリームキューのデータ数	*count* パラメーターに指定した数
count の指定値 ＞ 出力ストリームキューのデータ数	出力ストリームキューのデータ数
count の指定値 ＞ 出力ストリームキューの最大サイズ	

timeout

データがない場合に，待機する最大時間をミリ秒で指定します。

指定した値によって，次の処理が実行されます。

指定した値	実行される処理
負の数	待機しません。
0	結果データが到着するか，またはストリームが終了するまで待機します。
そのほかの値	結果データが到着するか，または指定時間が経過するまで待機します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	• 出力ストリームがすでに閉じている場合 • コールバック用リスナーオブジェクトがすでに登録されている場合 • パラメーターが不正の場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合

例外	発生条件
SDPClientQueryGroupStopException	結果データがなく，かつクエリグループが停止中の場合（SDPClientQueryGroupStateException の詳細例外）

戻り値

タプルのリスト（ArrayList<StreamTuple>型オブジェクト）。

8.8.5 getAll()メソッド

形式

```
ArrayList<StreamTuple> getAll()
```

説明

SDP サーバに登録されているすべてのタプルを取得します。

SDP サーバに結果データがない場合，空のArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合，またはコマンドによってクエリグループ停止が通知された場合），次のように処理されます。

- 停止通知後，最初のメソッド呼び出しの場合
  空のArrayList 型オブジェクトが返却されます。
- 停止通知後，2 回目以降のメソッド呼び出しの場合
  例外がスローされます。

このメソッドはポーリング用メソッドです。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	• 出力ストリームがすでに閉じている場合

例外	発生条件
SDPClientException	• コールバック用リスナーオブジェクトがすでに登録されている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく，クエリグループが停止中の場合

戻り値

タプルのリスト（ArrayList<StreamTuple>型オブジェクト）。

8.8.6 getAll(long timeout)メソッド

形式

```
ArrayList<StreamTuple> getAll(long timeout)
```

説明

SDP サーバに登録されているすべてのタプルを取得します。

SDP サーバに結果データがない場合，結果データが到着するか，または*timeout* パラメーターに指定した時間が経過するまで待機します。待機中に結果データが到着した場合，到着したデータを格納したArrayList型オブジェクトが返却されます。*timeout* パラメーターに指定した時間が経過した場合，または待機中のスレッドに割り込みが発生した場合は，次のどちらかのオブジェクトが返却されます。

- データが登録されている場合
  登録されたデータが格納されたArrayList 型オブジェクトが返却されます。
- データが登録されていない場合
  空のArrayList 型オブジェクトが返却されます。

このメソッドを呼び出した時点でクエリグループに対して停止が通知されていた場合（メソッドによってデータ送信終了が通知された場合，またはコマンドによってクエリグループ停止が通知された場合），次のように処理されます。

- 停止通知後，最初のメソッド呼び出しの場合
  空のArrayList 型オブジェクトが返却されます。
- 停止通知後，2 回目以降のメソッド呼び出しの場合
  例外がスローされます。

このメソッドは，ポーリング用メソッドです。

パラメーター

timeout

データがない場合に，待機する最大時間をミリ秒で指定します。

指定した値によって，次の処理が実行されます。

指定した値	実行される処理
負の数	待機しません。
0	結果データが到着するか，またはストリームが終了するまで待機します。
そのほかの値	結果データが到着するか，または指定時間が経過するまで待機します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientEndOfStreamException	送信データの処理が終了している場合
SDPClientException	• 出力ストリームがすでに閉じている場合 • コールバック用リスナーオブジェクトがすでに登録されている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合（SDPClientQueryGroupStateException の詳細例外）
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合
SDPClientQueryGroupStopException	結果データがなく，かつクエリグループが停止中の場合 （SDPClientQueryGroupStateException の詳細例外）

戻り値

タプルのリスト（ArrayList<StreamTuple>型オブジェクト）。

8.8.7 getFreeQueueSize()メソッド

形式

```
int getFreeQueueSize()
```

説明

出力ストリームキューの空きサイズを取得します。

出力ストリームキューの空きサイズは，system_config.properties のengine.maxQueueSize パラメーターに指定した値から，使用中のサイズを引いた値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	出力ストリームがすでに閉じている場合
SDPClientQueryGroupHoldException	クエリグループが閉塞中の場合 (SDPClientQueryGroupStateException の詳細例外)
SDPClientQueryGroupNotExistException	クエリグループが削除された場合
SDPClientQueryGroupStateException	クエリグループが実行中でない場合

戻り値

出力ストリームキューの空きサイズ（int)。

8.8.8　getMaxQueueSize()メソッド

形式

```
int getMaxQueueSize()
```

説明

出力ストリームキューの最大サイズを取得します。

入力ストリームキューの最大サイズは，system_config.properties のengine.maxQueueSize パラメーターに指定した値です。

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	出力ストリームがすでに閉じている場合

戻り値

出力ストリームキューの最大サイズ（int）。

8.8.9 registerForNotification(StreamEventListener n)メソッド

形式

```
void registerForNotification(StreamEventListener n)
```

説明

コールバック用リスナーオブジェクトを登録します。

このメソッド実行以降は，ポーリング用メソッド（get メソッドまたはgetAll メソッド）を実行できません。

1 つのストリームに対して 1 つのリスナーオブジェクトしか登録できません。また，登録済みのリスナーオブジェクトは，複数のストリームに登録できません。

パラメーター

n

リスナーオブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合
SDPClientException	• 出力ストリームがすでに閉じている場合 • ストリームにリスナーオブジェクトが登録済みの場合 • 指定したリスナーオブジェクトがほかのストリームで登録済みの場合
SDPClientQueryGroupNotExistException	クエリグループが削除された場合

戻り値

なし。

8.8.10　unregisterForNotification(StreamEventListener n)メソッド

形式

```
void unregisterForNotification(StreamEventListener n)
```

説明

登録したリスナーオブジェクトを解除して，以降のコールバック処理を解除します。

パラメーター

n

リスナーオブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合
SDPClientException	• リスナーオブジェクトが未定義の場合 • 出力ストリームがすでに閉じている場合

戻り値

なし。

8.9 SDPClientLoggerFactory クラス

説明

ログの出力に関する機能を提供するインタフェースを取得するためのファクトリクラスです。

メソッド

SDPClientLoggerFactory クラスのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
SDPClientLogger	getLogger(Class *clazz*)	メッセージやトレースを出力するためのSDPClientLogger インタフェースを取得します。

8.9.1 getLogger(Class clazz)メソッド

形式

```
public static SDPClientLogger getLogger(Class clazz)
```

説明

メッセージやトレースをログファイルに出力するためのSDPClientLogger オブジェクトを取得します。

パラメーター

clazz

このメソッドの呼び出し元クラスのクラスオブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合

戻り値

取得したSDPClientLogger オブジェクト。

注意事項

なし。

8.10 StreamTime クラス

クラス階層

```
java.lang.Object
  └java.util.Date
      └jp.co.Hitachi.soft.sdp.common.util.StreamTime
```

説明

StreamTime クラスは，Date クラスを拡張したクラスです。

タプルの特定の時刻を表すためのクラスです。

メソッド

StreamTime クラスのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
boolean	equals(StreamTime *when*)	自身の時刻と，パラメーターに指定したStreamTime 型オブジェクトが示す時刻が等しいかどうか判定します。
long	getTimeMillis()	自身が保持するミリ秒単位の時刻を取得します。
int	hashCode()	自身を示すハッシュコードを取得します。
java.lang.String	toString()	自身の保持する時刻の情報を文字列表現にして取得します。

注意事項

なし。

8.10.1 equals(StreamTime when)メソッド

形式

```
boolean equals(StreamTime when)
```

説明

自身の時刻と，パラメーターに指定したStreamTime 型オブジェクトの時刻が等しいかどうか判定します。

パラメーター

when

比較対象のStreamTime 型オブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合

戻り値

true

パラメーターで指定したオブジェクトがStreamTime 型オブジェクトであり，かつ，自身と同じ時刻です。

false

パラメーターで指定したオブジェクトがStreamTime 型オブジェクトではありません。または，自身と同じ時刻ではありません。

8.10.2 getTimeMillis()メソッド

形式

```
long getTimeMillis()
```

説明

自身が保持するミリ秒単位の時刻を取得します。

パラメーター

なし。

例外

なし。

戻り値

自身が保持するミリ秒単位の時刻（long）。

8.10.3　hashCode()メソッド

形式

```
int hashCode()
```

説明

自身を示すハッシュコードを取得します。

パラメーター

なし。

例外

なし。

戻り値

自身を示すハッシュコード（int）。

8.10.4　toString()メソッド

形式

```
java.lang.String toString()
```

説明

自身の保持する時刻の情報を文字列表現にして取得します。

パラメーター

なし。

例外

なし。

戻り値

自身の保持する時刻の情報を次に示す形式に変換した文字列表現（java.lang.String 型。△は半角スペースを示します）。

aaa△*bbb*△*cc*△*dd*:*ee*:*ff*△*ggg*△*hhhh*[timeMillis:*iii*]

出力内容を次の表に示します。

出力項目	内容	出力形式（範囲）
aaa	曜日	Sun，Mon，Tue，Wed，Thu，Fri，Sat のどれかが出力されます。
bbb	月	Jan，Feb，Mar，Apr，May，Jun，Jul，Aug，Sep，Oct，Nov，Dec のどれかが出力されます。
cc	日	2 桁の 10 進数（01～31）で出力されます。
dd	時	2 桁の 10 進数（00～23）で出力されます。
ee	分	2 桁の 10 進数（00～59）で出力されます。
ff	秒	2 桁の 10 進数（00～59）で出力されます。
ggg	タイムゾーン	タイムゾーンが出力されます。 タイムゾーンの設定がなければ空白が出力されます。
hhhh	年	4 桁の 10 進数で出力されます。
iii	自身の保持するミリ秒単位の時刻	自身の保持するミリ秒単位の時刻が出力されます。

出力例を次に示します。

```
"Thu Jan 01 09:00:01 JST 1970[timeMillis:1234]"
```

8.11 StreamTuple クラス

クラス階層

```
java.lang.Object
  └jp.co.Hitachi.soft.sdp.common.data.StreamTuple
```

説明

StreamTuple クラスは，タプルを表現するためのクラスです。

コンストラクター

StreamTuple クラスのコンストラクター一覧を次の表に示します。

コンストラクター名	説明
StreamTuple(Object[] *dataArray*)	パラメーターにデータオブジェクト配列を指定して，StreamTuple クラスのインスタンス（StreamTuple 型オブジェクト）を新しく生成します。

メソッド

StreamTuple クラスのメソッド一覧を次の表に示します。

戻り値	メソッド名	説明
boolean	equals(Object *obj*)	パラメーターで指定したStreamTuple 型オブジェクトのデータオブジェクト配列および時刻が，自身（StreamTuple 型オブジェクト）とすべて同じ内容かどうかを判定します。
Object[]	getDataArray()	自身（StreamTuple 型オブジェクト）のデータオブジェクト配列を取得します。
StreamTime	getSystemTime()	システム時刻を取得します。
jp.co.Hitachi.soft.sdp.common.data.TupleType	getTupleType()	次のタプル種別を判別します。 • CONTROL_TUPLE：制御タプル • PLUS_TUPLE：制御タプル以外のタプル
int	hashCode()	ハッシュコードを取得します。
java.lang.String	toString()	自身（StreamTuple 型オブジェクト）のタプル情報を表す文字列を取得します。

注意事項

タプルの時刻（StreamTime 型オブジェクト）には，タプルが SDP サーバに到着したときのシステム時刻が自動的に設定されます。カスタムアダプターでタプルを生成したときには，初期値としてタイムオブジェクトにはnull が設定されています。

使用例

Java のデータ型のデータを使用して data を作成して，StreamTuple 型オブジェクトを生成する例を示します。

この例では，Integer 型データおよびString 型データを要素とするタプルを生成します。

```
Object[] data = new Object[] {
    new Integer (1),
    new String (”AAA”)
};
tuple = new StreamTuple(data);
```

CQL のデータ型に対応する Java のデータ型については，「3.3　CQL のデータ型」を参照してください。

8.11.1　StreamTuple(Object[] dataArray)コンストラクター

形式

```
public StreamTuple(Object[] dataArray)
```

説明

パラメーターにデータオブジェクト配列を指定して，StreamTuple クラスのインスタンス（StreamTuple 型オブジェクト）を新しく生成します。

パラメーター

dataArray

タプルを構成するデータオブジェクトの配列を指定します。データは Java のデータ型で指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnull を指定した場合

8.11.2　equals(Object obj)メソッド

形式

```
boolean equals(Object obj)
```

説明

パラメーターで指定したStreamTuple型オブジェクトのデータオブジェクト配列および時刻が，自身（StreamTuple型オブジェクト）とすべて同じ内容かどうかを判定します。

パラメーター

obj

比較するStreamTuple型オブジェクトを指定します。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
NullPointerException	パラメーターにnullを指定した場合

戻り値

true

タプルの内容（データオブジェクト配列および時刻）がすべて同じです。

false

タプルの内容が異なります（内容が1つでも異なる場合はfalseになります）。

8.11.3　getDataArray()メソッド

形式

```
Object[] getDataArray()
```

説明

自身（StreamTuple型オブジェクト）のデータオブジェクト配列を取得します。

パラメーター

なし。

例外

なし。

戻り値

Java のデータ型のデータを含んだデータオブジェクト配列（Object[]型）。

8.11.4　getSystemTime()メソッド

形式

```
StreamTime getSystemTime()
```

説明

システム時刻を取得します。

パラメーター

なし。

例外

なし。

戻り値

システム時刻を表すオブジェクト（StreamTime 型オブジェクト）。

8.11.5　getTupleType()メソッド

形式

```
public jp.co.Hitachi.soft.sdp.common.data.TupleType getTupleType()
```

説明

次のタプル種別を判別します。

- CONTROL_TUPLE：制御タプル
- PLUS_TUPLE：制御タプル以外のタプル

パラメーター

なし。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
SDPClientException	• 入力ストリームがすでに閉じている場合 • ストリームデータの受付を停止している場合

戻り値

タプル種別（enum）。

注意事項

特になし。

8.11.6 hashCode()メソッド

形式

```
int hashCode()
```

説明

ハッシュコードを取得します。

このメソッドは，java.util.Hashtable クラスによって提供されるようなハッシュテーブルで使用するために用意されているメソッドです。

パラメーター

なし。

例外

なし。

戻り値

StreamTuple 型オブジェクトのハッシュコード（int）。

8.11.7　toString()メソッド

形式

```
java.lang.String toString()
```

説明

自身（StreamTuple 型オブジェクト）のタプル情報を表す文字列を取得します。次のデータを文字列に変換したデータが取得できます。

- データオブジェクト配列（Object[]型オブジェクト）
- システム時刻（StreamTime 型オブジェクト）

パラメーター

なし。

例外

なし。

戻り値

自身のタプル情報を表す文字列（java.lang.String 型）。

8.12　LogKind 列挙型

形式

```
public enum LogKind
```

説明

putMessage メソッドのログ種別の指定に使用するためのenum 列挙型です。

要素名およびログ種別を次の表に示します。

要素名	ログ種別
INFO	通常
WARNING	警告
ERROR	エラー

この列挙型で使用できるメソッドは，java.lang.Enum クラスで使用できるメソッドに従います。

注意事項

なし。

8.13　例外クラス

クラス階層

```
java.lang.Object
  |_java.lang.Throwable
      |_java.lang.Exception
          |_jp.co.Hitachi.soft.sdp.common.exception.
          |  SDPClientException
          |_jp.co.Hitachi.soft.sdp.common.exception.
          |  SDPClientFreeInputQueueSizeThresholdOverException
          |_jp.co.Hitachi.soft.sdp.common.exception.
             SDPClientFreeInputQueueSizeLackException
```

8.13.1　SDPClientException クラス

クラス階層

```
java.lang.Object
 |_java.lang.Throwable
   |_java.lang.Exception
     |_jp.co.Hitachi.soft.sdp.common.exception.SDPClientException
```

説明

ストリームデータ処理システムの SDP サーバで検知された例外の内容をカスタムアダプターに返す例外クラスです。

フィールド

なし。

コンストラクター

なし。

メソッド

java.lang.Exception クラスを継承しているため，getMessage メソッドなどのjava.lang.Exception クラスのメソッドを使用できます。

サブクラス

SDPClientException クラスのサブクラスを次の表に示します。

表 8-4　SDPClientException クラスのサブクラス

項番	サブクラス	説明
1	SDPClientQueryGroupException	カスタムアダプター実行時に，SDP サーバで検知したクエリグループの例外をカスタムアダプターに返すための例外クラスです。
2	SDPClientQueryGroupNotExistException	指定したクエリグループがない場合に生成される例外クラスです。
3	SDPClientQueryGroupStateException	指定したクエリグループが実行中ではない場合に生成される例外クラスです。
4	SDPClientQueryGroupHoldException	閉塞状態のクエリグループに対してメソッドを呼び出した場合に生成される例外クラスです。
5	SDPClientQueryGroupStopException	停止状態のクエリグループに対してメソッドを呼び出した場合に生成される例外クラスです。
6	SDPClientEndOfStreamException	入力ストリームデータが終了している場合に生成される例外クラスです。
7	SDPClientRelationStateException	指定したストリームがリレーション状態の場合に生成される例外クラスです。
8	SDPClientInitializeException	カスタムアダプターのロガーの初期化に失敗した場合に生成される例外クラスです。

サブクラスのクラス階層を次に示します。

```
jp.co.Hitachi.soft.sdp.common.exception.SDPClientException
 |_jp.co.Hitachi.soft.sdp.common.exception
 |  .SDPClientQueryGroupException
 |     |_jp.co.Hitachi.soft.sdp.common.exception.
 |     |  SDPClientQueryGroupNotExistException
 |     |_jp.co.Hitachi.soft.sdp.common.exception.
 |     |  SDPClientQueryGroupStateException
 |     |_jp.co.Hitachi.soft.sdp.common.exception.
 |     |  SDPClientQueryGroupHoldException
 |     |_jp.co.Hitachi.soft.sdp.common.exception.
 |        SDPClientQueryGroupStopException
 |_jp.co.Hitachi.soft.sdp.common.exception.
 |  SDPClientEndOfStreamException
 |_jp.co.Hitachi.soft.sdp.common.exception.
 |  SDPClientRelationStateException
 |_jp.co.Hitachi.soft.sdp.common.exception.
    SDPClientInitializeException
```

8.13.2 SDPClientFreeInputQueueSizeThresholdOverException クラス

クラス階層

```
java.lang.Object
  |_java.lang.Throwable
      |_java.lang.Exception
         |_jp.co.Hitachi.soft.sdp.common.exception.
            SDPClientFreeInputQueueSizeThresholdOverException
```

説明

入力ストリームキューの空きサイズがsystem_config.properties のstream.freeInputQueueSizeThreshold キーで指定したしきい値以下になったことを示す例外クラスです。

フィールド

なし。

コンストラクター

なし。

サブクラス

なし。

8.13.3 SDPClientFreeInputQueueSizeLackException クラス

クラス階層

```
java.lang.Object
  |_java.lang.Throwable
      |_java.lang.Exception
         |_jp.co.Hitachi.soft.sdp.common.exception.
            SDPClientFreeInputQueueSizeLackException
```

説明

入力ストリームキューの空きサイズが不足していることを示す例外クラスです。

フィールド

なし。

コンストラクター

なし。

サブクラス

なし。

9 外部アダプター用 API

この章では，外部アダプターの作成で使用する外部アダプター用 API の文法について説明します。

9.1　外部アダプター用 API の記述形式

ここでは，外部アダプター用 API の記述形式について説明します。

各 API で説明する項目は次のとおりです。ただし，API によっては説明しない項目もあります。

形式

API の記述形式を示します。

説明

API の機能について説明します。

パラメーター

API のパラメーターについて説明します。

例外

API を利用する際に発生する例外について説明します。

戻り値

API の戻り値について説明します。

注意事項

API を利用する上での注意事項について説明します。

9.2 外部アダプターの API の一覧

パッケージcom.Hitachi.soft.hsdp.exadaptor.api

パッケージcom.Hitachi.soft.hsdp.exadaptor.api のインタフェース，クラス，および列挙型を次に示します。

インタフェース	説明
HSDPColumn	ストリームの列を示すインタフェースです。
HSDPDestInfo	宛先情報を保持するクラスのインタフェースです。
HSDPDispatch	送信するデータの宛先を決定するクラス（カスタムディスパッチャー）のインタフェースです。
HSDPDispatchInfo	データの宛先を決める情報（ディスパッチ情報）を保持するクラスのインタフェースです。
HSDPEvent	受信イベント情報を保持するクラスのインタフェースです。
HSDPEventListener	SDP サーバ上でタプルが生成された場合に呼び出されるメソッドを実装するインタフェースです。
HSDPStreamInput	入力ストリームを操作するためのインタフェースです。
HSDPStreamOutput	出力ストリームを操作するためのインタフェースです。
HSDPStreamSchema	ストリームのスキーマを示すクラスのインタフェースです。

クラス	説明
HSDPAdaptorManager	外部アダプターを操作するクラスです。

列挙型	説明
HSDPColumnType	カラムタイプの列挙型です。
HSDPConnStatus	接続の状態の列挙型です。

パッケージcom.Hitachi.soft.hsdp.exadaptor.exception

パッケージcom.Hitachi.soft.hsdp.exadaptor.exception に含まれる例外を次に示します。

例外	説明
HSDPAdaptorDispatchException	送信するデータの宛先の割り当てに失敗したことを示す例外クラスです。
HSDPAdaptorException	外部アダプターの API の実行時に，エラーが検出されたことを示す例外クラスです。
HSDPAdaptorQueueSizeLackException	送信するキューに十分な空きエントリがないことを示す例外クラスです。

9.3 HSDPColumn インタフェース

説明

ストリームの列を示すインタフェースです。

メソッド

次の表に，HSDPColumn インタフェースのメソッドの一覧を示します。

項番	メソッド	説明
1	getName	列名を取得します。
2	getType	カラムタイプを取得します。
3	getSize	最大列サイズ※1 を取得します。
4	getDigitNumber	列の桁数※2 を取得します。

注※1
外部アダプターの場合，TIMESTAMP 型には，常に 12 バイトの形式が使用されます。
クエリ内の次の列は，常に32,767 バイトのVARCHAR 型として扱われます。

- 文字列定数
- 文字列定数を返すスカラ関数

注※2
カラムタイプがTIMESTAMP 型の場合，秒の小数部の桁数が示されます。

9.3.1 getName()メソッド

形式

```
java.lang.String getName()
```

説明

取得するタプルが通常のタプルの場合，列名を取得します。

取得するタプルが制御タプルの場合，オブジェクトの型名（getType()メソッドで取得できる値を文字列化した値）を取得します。

パラメーター

なし

例外

なし

戻り値

オブジェクトの型情報(java.lang.String)

9.3.2 getType()メソッド

形式

```
HSDPColumnType getType()
```

説明

オブジェクトの型名を取得します。

パラメーター

なし

例外

なし

戻り値

オブジェクトの型情報(com.Hitachi.soft.hsdp.exadaptor.api.HSDPColumnType)

9.3.3 getSize()メソッド

形式

```
int getSize()
```

説明

取得するタプルが通常のタプルの場合，最大の列サイズを取得します。

取得するタプルが制御タプルの場合，オブジェクトのサイズをバイト数で取得します。

パラメーター

なし

例外

なし

戻り値

オブジェクトのサイズ（int 型）

表 9-1　getSize()メソッドが返す値の一覧

項番	CQL Type（Java データ型）	通常のタプル	制御タプル
1	BYTE（java.lang.Byte）	1	1
2	SHORT（java.lang.Short）	2	2
3	INT（java.lang.Integer）	4	4
4	LONG（java.lang.Long）	8	8
5	FLOAT（java.lang.Float）	4	4
6	DOUBLE（java.lang.Double）	8	8
7	CHAR（java.lang.String）	CQL，または外部定義関数定義ファイル（ReturnInformation タグのtype 属性）に指定したCHAR〔ACTER〕〔'('*n*')'〕の*n*。	制御タプルの引数オブジェクトとして指定したString 型をバイト配列にしたときのバイト長。※
8	VARCHAR（java.lang.String）	CQL，または外部定義関数定義ファイル（ReturnInformation タグのtype 属性）に指定したVARCHAR〔ACTER〕〔'('*n*')'〕の*n*。 CQL に，文字列定数または文字列を返すスカラ関数を指定した場合は，常に32,767 が返却されます。	
9	TIMESTAMP（java.sql.Timestamp）	12	12
10	DATE（java.sql.Date）	8	8
11	TIME（java.sql.Time）	8	8
12	BIG_DECIMAL（java.math.BigDecimal）	CQL，または外部定義関数定義ファイル（ReturnInformation タグのtype 属性）に指定したDEC〔IMAL〕〔'('*m*')'〕，またはNUMERIC〔'('*m*')'〕の*m*。 CQL にBigDecimal 型を返すスカラ関数を指定した場合は，常に32,767 が返却されます。	制御タプルの引数オブジェクトとして指定したBigDecimal 型をString 型としてバイト配列にしたときのバイト長。 例：3.14 の場合，4

注※
制御タプルでは CQL のCHAR 型とVARCHAR 型の区別がないため，文字列はすべて可変長型文字列として処理されます。

9.3.4 getDigitNumber()メソッド

形式

```
int getDigitNumber()
```

説明

取得するタプルが通常のタプルの場合，列の桁数を取得します。

取得するタプルが制御タプルの場合で，オブジェクト型が数データ型のときに，列の桁数を取得します。

パラメーター

なし

例外

なし

戻り値

オブジェクトの桁数（int 型)。

表 9-2 getDigitNumber()メソッドが返す値の一覧

項番	CQL Type（Java データ型)	通常のタプル	制御タプル
1	BYTE（java.lang.Byte)	0	0
2	SHORT（java.lang.Short)	0	0
3	INT（java.lang.Integer)	0	0
4	LONG（java.lang.Long)	0	0
5	FLOAT（java.lang.Float)	0	0
6	DOUBLE（java.lang.Double)	0	0
7	CHAR（java.lang.String)	0	0
8	VARCHAR（java.lang.String)	0	0
9	TIMESTAMP（java.sql.Timestamp)	CQL，または外部定義関数定義ファイル（ReturnInformation タグのtype 属性）に指定したTIMESTAMP〔'('*p*')'〕の*p*。	9
10	DATE（java.sql.Date)	0	0
11	TIME（java.sql.Time)	0	0

項番	CQL Type（Java データ型）	通常のタプル	制御タプル
12	BIG_DECIMAL （java.math.BigDecimal）	CQL，または外部定義関数定義ファイル（ReturnInformation タグのtype 属性）に指定したDEC〔IMAL〕〔'('*m*')'〕，またはNUMERIC〔'('*m*')'〕の*m*。 CQL にBigDecimal 型を返すスカラ関数を指定した場合は，常に0 が返却されます。	制御タプルの引数オブジェクトとして指定したBigDecimal 型をString 型としてバイト配列にしたときのバイト長。 例：3.14 の場合，4

9.4 HSDPDestInfo インタフェース

説明

宛先情報を保持するクラスのインタフェースです。

メソッド

次の表に，HSDPDestInfo インタフェースのメソッドの一覧を示します。

項番	メソッド	説明
1	getAddress	接続先のホスト名または IP アドレスを取得します。
2	get Port	接続先のポート番号を取得します。
3	getWorkingDirectoryName	運用ディレクトリのパスを取得します。
4	getServerClusterName	サーバクラスター名を取得します。
5	getQueryGroupName	クエリグループ名を取得します。
6	getStreamName	ストリーム名を取得します。
7	getID	1 から始まる接続先の ID 番号を取得します。
8	getConnStatus	接続の状態※を取得します。

注※

メソッドが実行された場合の接続の状態を返します。

9.4.1 getAddress()メソッド

形式

```
java.lang.String getAddress()
```

説明

接続先のホスト名または IP アドレスを取得します。

パラメーター

なし

例外

なし

戻り値

接続先のホスト名または IP アドレス。

9.4.2 getPort()メソッド

形式

```
int getPort()
```

説明

接続先のポート番号を取得します。

パラメーター

なし

例外

なし

戻り値

接続先のポート番号。

9.4.3 getWorkingDirectoryName()メソッド

形式

```
java.lang.String getWorkingDirectoryName()
```

説明

運用ディレクトリの名前を取得します。

パラメーター

なし

例外

なし

戻り値

運用ディレクトリの名前。

9.4.4 getServerClusterName()メソッド

形式

```
java.lang.String getServerClusterName()
```

説明

サーバクラスター名を取得します。

パラメーター

なし

例外

なし

戻り値

サーバクラスター名。

9.4.5 getQueryGroupName()メソッド

形式

```
java.lang.String getQueryGroupName()
```

説明

クエリグループ名を取得します。

パラメーター

なし

例外

なし

戻り値

クエリグループ名。

9.4.6　getStreamName()メソッド

形式

```
java.lang.String getStreamName()
```

説明

ストリーム名を取得します。

パラメーター

なし

例外

なし

戻り値

ストリーム名。

9.4.7　getID()メソッド

形式

```
int getID()
```

説明

1 から始まる接続先の ID 番号を取得します。

パラメーター

なし

例外

なし

戻り値

1 から始まる接続先の ID 番号。

9.4.8 getConnStatus()メソッド

形式

```
HSDPConnStatus getConnStatus()
```

説明

接続の状態を取得します。

> **メモ**
>
> メソッドが実行された場合の接続の状態を返します。

パラメーター

なし

例外

なし

戻り値

接続の状態。

9.5 HSDPDispatch インタフェース

説明

送信するデータの宛先を決定するクラス（カスタムディスパッチャー）のインタフェースです。

メソッド

次の表に，HSDPDispatch インタフェースのメソッドの一覧を示します。

メソッド	形式	説明
dispatch	int dispatch(HSDPDispatchInfo *dispatchInfo*,byte[] *data*)	送信するデータの宛先を決めます。

9.5.1 dispatch(HSDPDispatchInfo dispatchInfo,byte[] data)メソッド

形式

```
int dispatch(HSDPDispatchInfo dispatchInfo,byte[] data)
```

説明

送信するデータの宛先を決めます。

パラメーター

dispatchInfo

ディスパッチ情報です。

data

put()メソッドに指定される送信対象のデータです。

例外

なし

戻り値

ディスパッチ先を示すために使用される ID 番号です。この番号は，接続先の数に従って1 から割り当てられます。

9.6 HSDPDispatchInfo インタフェース

説明

データの宛先を決める情報（ディスパッチ情報）を保持するクラスのインタフェースです。

メソッド

次の表に，HSDPDispatchInfo インタフェースのメソッドの一覧を示します。

項番	メソッド	形式	説明
1	getSchema	HSDPStreamSchema getSchema()	ストリームのスキーマを取得します。
2	getDestNum	int getDestNum()	宛先のパラレル実行数を取得します。
3	getDestInfo	java.util.ArrayList<HSDPDestInfo> getDestInfo()	宛先の情報を取得します。

9.6.1 getSchema()メソッド

形式

```
HSDPStreamSchema getSchema()
```

説明

ストリームのスキーマを取得します。

パラメーター

なし

例外

なし

戻り値

ストリームのスキーマ。

9.6.2　getDestNum()メソッド

形式

```
int getDestNum()
```

説明

宛先のパラレル実行数を取得します。

パラメーター

なし

例外

なし

戻り値

宛先のパラレル実行数。

9.6.3　getDestInfo()メソッド

形式

```
java.util.ArrayList<HSDPDestInfo> getDestInfo()
```

説明

宛先の情報を取得します。

パラメーター

なし

例外

なし

戻り値

宛先の情報。

9.7 HSDPEvent インタフェース

説明

受信イベント情報を保持するクラスのインタフェースです。

メソッド

次の表に，HSDPEvent インタフェースのメソッドの一覧を示します。

項番	メソッド	形式	説明
1	getSchema	HSDPStreamSchema getSchema()	ストリームのスキーマを取得します。
2	getData	byte[] getData()	受信したデータを取得します。
3	getSystemTime	java.sql.Timestamp getSystemTime()	受信したデータのシステム時刻を取得します。

9.7.1 getSchema()メソッド

形式

```
HSDPStreamSchema getSchema()
```

説明

ストリームのスキーマを取得します。

パラメーター

なし

例外

なし

戻り値

ストリームのスキーマ。

9.7.2　getData()メソッド

形式

```
byte[] getData()
```

説明

受信したデータを取得します。

パラメーター

なし

例外

なし

戻り値

受信したデータ。

9.7.3　getSystemTime()メソッド

形式

```
java.sql.Timestamp getSystemTime()
```

説明

受信したデータのシステム時刻を取得します。

パラメーター

なし

例外

なし

戻り値

受信したデータのシステム時刻。

9.8 HSDPEventListener インタフェース

説明

SDP サーバ上でタプルが生成された場合に呼び出されるメソッドを実装するインタフェースです。

メソッド

次の表に，HSDPEventListener インタフェースのメソッドの一覧を示します。

項番	メソッド	形式	説明
1	onEvent	void onEvent(HSDPEvent event)	SDP サーバ上でタプルが生成された場合に呼び出されるメソッドです。 HSDPEvent インタフェースのgetData()メソッドで取得した受信データの形式は，HSDPStreamInput インタフェースのput()メソッドの引数に指定した形式と同じです。

9.8.1 onEvent(HSDPEvent event)メソッド

形式

```
void onEvent(HSDPEvent event)
```

説明

SDP サーバ上でタプルが生成された場合に呼び出されるメソッドです。

HSDPEvent インタフェースのgetData()メソッドで取得した受信データの形式は，HSDPStreamInput インタフェースのput()メソッドの引数に指定した形式と同じです。

パラメーター

event

受信イベント情報です。

例外

なし

戻り値

なし

9.9 HSDPStreamInput インタフェース

説明

入力ストリームを操作するためのインタフェースです。

メソッド

次の表に，HSDPStreamInput インタフェースのメソッドの一覧を示します。

項番	メソッド	形式	説明
1	close	void close() throws HSDPAdaptorException	ストリームをクローズします。
2	put	void put(byte[] *data*)throws HSDPAdaptorException	データを送信します。
3	putControl(Object... *args*)	public void putControl(Object... *args*)	サーバモードの場合に，制御タプルを送信します。
4	putControl(Timestamp *ts*, Object... *args*)	public void putControl(Timestamp *ts*, Object... *args*)	データソースモードの場合に，制御タプルを送信します。
5	heartbeat	void heartbeat(java.sql.Timestamp *ts*)throws HSDPAdaptorException	ハートビートを送信します。
6	suspend	void suspend(boolean *autoResume*)throws HSDPAdaptorException	一時停止を要求します。
7	resume	void resume()throws HSDPAdaptorException	再開を要求します。
8	loadDispatcher	void loadDispatcher(java.lang.String *classPath*,java.lang.String *className*)throws HSDPAdaptorException	カスタムディスパッチャーをロードします。
9	unloadDispatcher	void unloadDispatcher()throws HSDPAdaptorException	カスタムディスパッチャーをアンロードします。
10	getSchema	HSDPStreamSchema getSchema()	ストリームのスキーマを取得します。
11	getDestInfo	java.util.ArrayList<HSDPDestInfo> getDestInfo()	宛先の情報を取得します。

9.9.1　close()メソッド

形式

```
void close()
throws HSDPAdaptorException
```

説明

ストリームをクローズします。

送信キューに残っているデータの送信を試みます。送信がビジー状態または切断によってデータを送信できない場合，送信は中止されます。

パラメーター

なし

例外

名前	説明
HSDPAdaptorException	ストリームがクローズ済みです。

戻り値

なし

9.9.2　put(byte[] data)メソッド

形式

```
void put(byte[] data)
throws HSDPAdaptorException
```

説明

データを送信します。

TCP 入力アダプターの受信データの型である「0: Normal data」の形式でデータを送信します。このメソッドのパラメーターとして，ヘッダー（2 バイト）と予約済み領域（2 バイト）に続けてデータを指定します。オフセットはすべてのデータでゼロと見なされます。

ビッグエンディアン形式のバイトオーダーに変換されるデータを指定します。

12バイトの形式はTIMESTAMP型にだけ使用できます。最初の８バイトには，ミリ秒単位で１ミリ秒より大きい値の時刻を指定します。最後の４バイトには，ナノ秒単位で１秒より小さい値の時刻を指定します。最初の８バイトに指定された１秒より小さい値の時刻は，最後の４バイトに指定された時刻で上書きされます。最後の４バイトには，送信先のクエリに定義されるTIMESTAMP型の値の秒の小数部の値を指定します。例えば，値がTIMESTAMP(6)として定義される場合，最後の３桁が0となる123456000（ナノ秒）を指定します。

TIMESTAMP型の設定例：

```
java.sql.Timestamp ts = new java.sql.Timestamp(System.currentTimeMillis());
byteBuffer.putLong(ts.getTime());
byteBuffer.putInt(ts.getNanos());
```

パラメーター

data

送信対象のデータです。

例外

例外	発生条件
HSDPAdaptorDispatchException	送信するデータの宛先の割り当てに失敗しました。送信するデータは，送信キューに登録されません。 送信するデータの宛先への接続に，外部アダプター定義ファイルのhsdp.connection.retry.countに指定された数と同じ回数失敗したため，接続が終了しました。送信するデータは，送信キューに登録されません。※
HSDPAdaptorException	ストリームがクローズ済みです。
HSDPAdaptorQueueFullException	データを保存するキューが上限に達しました。送信キューにデータは登録されません。 外部アダプター定義ファイルのhsdp.send.queuefull.actionパラメーターの値がdiscardの場合に発生します。
HSDPAdaptorQueueSizeLackException	データを保存するキューが上限に達しました。送信キューから古いデータが削除され，新しいデータが登録されます。 外部アダプター定義ファイルのhsdp.send.queuefull.actionパラメーターの値がoverwriteの場合に発生します。
注※ 送信先のクエリグループが並列実行をしているか，HSDPAdaptorManagerクラスのopenStreamInput()に複数の宛先がある場合に，一部のキューが条件に一致すると，この例外がスローされます。 この場合，宛先が正常に割り当てられた送信対象データは，送信キューに登録されます。 この例外をスローする条件と，HSDPAdaptorQueueSizeLackExceptionをスローする条件が同時に満たされると，HSDPAdaptorQueueSizeLackExceptionがスローされます。	

戻り値

なし

9.9.3　putControl(Object... args)メソッド

形式

```
void putControl(Object... args)
```

説明

サーバモードの場合に，制御タプルを送信します。制御タプルに任意の数のオブジェクトを付与できます。時刻制御には SDP サーバに到達した時刻が使用されます。

データソースモードの場合，時刻制御には外部アダプターの送信時刻が使用されます。

パラメーター

args

制御タプルに，任意に設定できるオブジェクト（Object 型オブジェクト）を指定します。設定するオブジェクトには，「3.3.1　CQL のデータ型と Java のデータ型のマッピング」に記載された Java のデータ型だけを使用してください。String 型および char 型のオブジェクトを設定した場合，データ送信対象となるクエリグループのエンコーディング値に従って送信されます。文字コード範囲外の文字列が入力された場合，Java の仕様に従って，指定したエンコーディング値に対応する Java の文字セットのデフォルトの置換バイト配列に置き換えて処理されます。この際，エラーは発生しません。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
HSDPAdaptorDispatchException	• 送信データへの宛先の割り当てに失敗した場合，送信データは送信キューに登録されません。 • 外部アダプター定義ファイルのhsdp.connection.retry.count パラメーターに指定した回数分，送信データの宛先との接続の再試行に失敗した場合，接続が切断されます。
HSDPAdaptorException	• ストリームがクローズ済みです。 • 引数に Java の基本データ型以外の型を指定しています。
HSDPAdaptorQueueFullException	データを保存するキューが上限に達しました。送信キューにデータは登録されません。 外部アダプター定義ファイルのhsdp.send.queuefull.action パラメーターの値がdiscard の場合に発生します。

例外	発生条件
HSDPAdaptorQueueSizeLackException	データを保存するキューが上限に達した場合，送信キューから古いデータが削除され，新しいデータが登録されます。 外部アダプター定義ファイルのhsdp.send.queuefull.action パラメーターの値がoverwrite の場合に発生します。
NullPointerException	パラメーターにnull を指定した場合

戻り値

なし。

注意事項

特になし。

9.9.4 putControl(Timestamp ts, Object... args)メソッド

形式

```
public void putControl(Timestamp ts, Object... args)
```

説明

データソースモードの場合に，制御タプルを送信します。制御タプルに任意の数のオブジェクトを付与できます。時刻制御には引数に設定した*ts* が使用されます。

サーバモードの場合に使用すると，設定した*ts* の値がサーバ到達時の時刻で上書きされます。

パラメーター

ts

時刻制御に使用するタイムスタンプ情報（Timestamp 型オブジェクト）です。

args

制御タプルに，任意に設定できるオブジェクト（Object 型オブジェクト）を指定します。設定するオブジェクトには，「3.3.1 CQL のデータ型と Java のデータ型のマッピング」に記載された Java のデータ型だけを使用してください。String 型およびchar 型のオブジェクトを設定した場合，データ送信対象となるクエリグループのエンコーディング値に従って送信されます。文字コード範囲外の文字列が入力された場合，Java の仕様に従って，指定したエンコーディング値に対応する Java の文字セットのデフォルトの置換バイト配列に置き換えて処理されます。この際，エラーは発生しません。

例外

例外とその発生条件を次の表に示します。

例外	発生条件
HSDPAdaptorDispatchException	• 送信データへの宛先の割り当てに失敗した場合，送信データは送信キューに登録されません。 • 外部アダプター定義ファイルのhsdp.connection.retry.count パラメーターに指定した回数分，送信データの宛先との接続の再試行に失敗した場合，接続が切断されます。
HSDPAdaptorException	• ストリームがクローズ済みです。 • 引数に Java の基本データ型以外の型を指定しています。
HSDPAdaptorQueueFullException	データを保存するキューが上限に達しました。送信キューにデータは登録されません。 外部アダプター定義ファイルのhsdp.send.queuefull.action パラメーターの値がdiscard の場合に発生します。
HSDPAdaptorQueueSizeLackException	データを保存するキューが上限に達した場合，送信キューから古いデータが削除され，新しいデータが登録されます。 外部アダプター定義ファイルのhsdp.send.queuefull.action パラメーターの値がoverwrite の場合に発生します。
NullPointerException	パラメーターにnull を指定した場合

戻り値

なし。

注意事項

特になし。

9.9.5 heartbeat(java.sql.Timestamp ts)メソッド

形式

```
void heartbeat(java.sql.Timestamp ts)
throws HSDPAdaptorException
```

説明

ハートビートを送信します。

パラメーター

ts

タイムスタンプです。

例外

例外	発生条件
HSDPAdaptorDispatchException	送信するデータの宛先との接続の再試行に失敗したため，接続を終了しました。送信するデータは，送信キューに登録されません。
HSDPAdaptorException	ストリームがクローズ済みです。 宛先のクエリがサーバモードで実行されました。
HSDPAdaptorQueueSizeLackException	データを保存するキューが上限に達しました。送信キューから古いデータが削除され，新しいデータが登録されます。

戻り値

なし

9.9.6 suspend(boolean autoResume)メソッド

形式

```
void suspend(boolean autoResume)
throws HSDPAdaptorException
```

説明

一時停止を要求します。

パラメーター

autoResume

入力ストリームを自動的に再開するかどうかを指定します。

- true：自動的に再開
- false：手動による再開

例外

例外	発生条件
HSDPAdaptorDispatchException	送信するデータの宛先との接続の再試行に失敗したため，接続を終了しました。送信するデータは，送信キューに登録されません。
HSDPAdaptorException	ストリームがクローズ済みです。 宛先のクエリがサーバモードで実行されました。
HSDPAdaptorQueueSizeLackException	データを保存するキューが上限に達しました。送信キューから古いデータが削除され，新しいデータが登録されます。

戻り値

なし

9.9.7　resume()メソッド

形式

```
void resume()
throws HSDPAdaptorException
```

説明

再開を要求します。

パラメーター

なし

例外

例外	発生条件
HSDPAdaptorDispatchException	送信するデータの宛先との接続の再試行に失敗したため，接続を終了しました。送信するデータは，送信キューに登録されません。
HSDPAdaptorException	ストリームがクローズ済みです。 宛先のクエリがサーバモードで実行されました。
HSDPAdaptorQueueSizeLackException	データを保存するキューが上限に達しました。送信キューから古いデータが削除され，新しいデータが登録されます。

戻り値

なし

9.9.8 loadDispatcher(java.lang.String classPath, java.lang.String className)メソッド

形式

```
void loadDispatcher(java.lang.String classPath, java.lang.String className)
throws HSDPAdaptorException
```

説明

カスタムディスパッチャーをロードします。

パラメーター

classPath

クラスのパスです。カスタムディスパッチャーのクラスパスを jar ファイルパスまたはディレクトリパスで指定します。相対パスを使用する場合，外部アダプターが実行されるディレクトリからの相対パスを指定します。

className

クラスの名前です。

例外

例外	発生条件
HSDPAdaptorException	ストリームがクローズ済みです。 負荷分散方式がCUSTOM ではありません。 カスタムディスパッチャーがすでに登録されています。 指定したクラスのパスが見つかりません。 指定したクラスが見つかりません。 クラスファイルの読み込みに失敗しました。 指定したクラスがインタフェースに実装されていません。 カスタムディスパッチャーのインスタンスの生成に失敗しました。

戻り値

なし

9.9.9　unloadDispatcher()メソッド

形式

```
void unloadDispatcher()
throws HSDPAdaptorException
```

説明

カスタムディスパッチャーをアンロードします。

パラメーター

なし

例外

例外	発生条件
HSDPAdaptorException	ストリームがクローズ済みです。 カスタムディスパッチャーが登録されていません。

戻り値

なし

9.9.10　getSchema()メソッド

形式

```
HSDPStreamSchema getSchema() throws HSDPAdaptorException
```

説明

ストリームのスキーマを取得します。

パラメーター

なし

例外

名前	説明
HSDPAdaptorException	ストリームがクローズ済みです。

戻り値

ストリームのスキーマ。

9.9.11　getDestInfo()メソッド

形式

```
java.util.ArrayList<HSDPDestInfo> getDestInfo()
```

説明

宛先の情報を取得します。

パラメーター

なし

例外

名前	説明
HSDPAdaptorException	ストリームがクローズ済みです。

戻り値

宛先の情報。

9.10　HSDPStreamOutput インタフェース

説明

出力ストリームを操作するためのインタフェースです。

メソッド

次の表に，HSDPStreamOutput インタフェースのメソッドの一覧を示します。

項番	メソッド	形式	説明
1	close	void close()throws HSDPAdaptorException	ストリームをクローズします。
2	register	void register(HSDPEventListener *listener*)throws HSDPAdaptorException	コールバックオブジェクトを登録します。
3	unregister	void unregister(HSDPEventListener *listener*)throws HSDPAdaptorException	登録したコールバックオブジェクトを登録解除します。 受信キュー内のデータを破棄し，register()メソッドによって確立された接続を終了します。
4	getSchema	HSDPStreamSchema getSchema()throws HSDPAdaptorException	ストリームのスキーマを取得します。
5	getDestInfo	java.util.List<HSDPDestInfo> getDestInfo(HSDPEventListener *listener*)throws HSDPAdaptorException	宛先の情報を取得します。

9.10.1　close()メソッド

形式

```
void close() throws HSDPAdaptorException
```

説明

ストリームをクローズします。

受信キュー内のデータを破棄し，register()メソッドによるすべての接続を終了します。

パラメーター

なし

例外

名前	説明
HSDPAdaptorException	ストリームがクローズ済みです。

戻り値

なし

9.10.2 register(HSDPEventListener listener)メソッド

形式

```
void register(HSDPEventListener listener)
throws HSDPAdaptorException
```

説明

コールバックオブジェクトを登録します。

HSDPAdaptorManager クラスのopenStreamOutput()メソッドで取得した検索結果を基に，出力ストリームとの接続を確立します。

出力ストリームとの接続を最大hsdp.tcp.connect.timeout 秒待ちます。接続に成功すると，メソッドは正常に終了します。

複数のコールバックオブジェクトを 1 つのストリームに登録する場合，受信データは関連するコールバックオブジェクトに通知されます。

同じコールバックオブジェクトは 1 つのストリームに複数回登録できません。

同じコールバックオブジェクトを複数のストリームに登録する場合，同じコールバックオブジェクトが同時に実行されるため，コールバックオブジェクトはスレッドセーフでなければなりません。

パラメーター

listener

コールバックオブジェクトです。

例外

例外	発生条件
HSDPAdaptorException	ストリームがクローズ済みです。 指定したコールバックがすでに登録されています。 出力ストリームとの接続に失敗しました。

戻り値

なし

9.10.3 unregister(HSDPEventListener listener)メソッド

形式

```
void unregister(HSDPEventListener listener)
throws HSDPAdaptorException
```

説明

コールバックオブジェクトの登録を解除します。

受信キュー内のデータを破棄し，register()メソッドによって確立された接続を終了します。

パラメーター

listener

register()メソッドに指定されるコールバックオブジェクトです。

例外

例外	発生条件
HSDPAdaptorException	ストリームがクローズ済みです。 指定したコールバックが登録されていません。

戻り値

なし

9.10.4　getSchema()メソッド

形式

```
HSDPStreamSchema getSchema() throws HSDPAdaptorException
```

説明

ストリームのスキーマを取得します。

パラメーター

なし

例外

名前	説明
HSDPAdaptorException	ストリームがクローズ済みです。

戻り値

ストリームのスキーマ。

9.10.5　getDestInfo(HSDPEventListener listener)メソッド

形式

```
java.util.List<HSDPDestInfo> getDestInfo(HSDPEventListener listener) throws HSDPAdaptorExcep
tion
```

説明

宛先の情報を取得します。

パラメーター

listener

register()メソッドに指定されるコールバックオブジェクトです。

例外

名前	説明
HSDPAdaptorException	ストリームがクローズ済みです。

名前	説明
HSDPAdaptorException	指定したコールバックが登録されていません。

戻り値

宛先の情報。

9.11　HSDPStreamSchema インタフェース

説明

ストリームのスキーマを示すクラスのインタフェースです。

メソッド

次の表に，HSDPStreamSchema インタフェースのメソッドの一覧を示します。

項番	メソッド	形式	説明
1	getColumnList	java.util.List<HSDPColumn> getColumnList()	スキーマの列情報を取得します。
2	getEncoding	java.lang.String getEncoding()	クエリグループ用プロパティファイルのquerygroup.encoding に設定した値を取得します。 例えば，ASCII や UTF-8 などの文字コードの名称を文字列として返します。
3	getQueryGroupName	java.lang.String getQueryGroupName()	クエリグループ名を取得します。
4	getServerClusterName	java.lang.String getServerClusterName()	サーバクラスター名を取得します。
5	getStreamName	java.lang.String getStreamName()	ストリーム名を取得します。
6	getTupleType	com.Hitachi.soft.hsdp.exadaptor.common.data.TupleType getTupleType()	次のタプル種別を判別します。 • CONTROL_TUPLE：制御タプル • NORMAL_TUPLE：制御タプル以外のタプル

9.11.1　getColumnList()メソッド

形式

```
java.util.List<HSDPColumn> getColumnList()
```

説明

スキーマの列情報を取得します。

制御タプルの場合，データ入力時に付与したオブジェクトの情報を取得します。

パラメーター

なし

例外

なし

戻り値

スキーマの列情報。

9.11.2 getEncoding()メソッド

形式

```
java.lang.String getEncoding()
```

説明

データを送受信するストリームのクエリグループ用プロパティファイルのquerygroup.encoding に定義している値を取得します。

例えば，ASCII や UTF-8 などの文字コードの名称を文字列として取得します。

パラメーター

なし。

例外

なし。

戻り値

エンコーディングに使用する文字コードの名称（java.lang.String)。

注意事項

特になし。

9.11.3　getQueryGroupName()メソッド

形式

```
java.lang.String getQueryGroupName()
```

説明

クエリグループ名を取得します。

パラメーター

なし

例外

なし

戻り値

クエリグループ名。

9.11.4　getServerClusterName()メソッド

形式

```
java.lang.String getServerClusterName()
```

説明

サーバクラスター名を取得します。

パラメーター

なし

例外

なし

戻り値

サーバクラスター名。

9.11.5　getStreamName()メソッド

形式

```
java.lang.String getStreamName()
```

説明

ストリーム名を取得します。

パラメーター

なし

例外

なし

戻り値

ストリーム名。

9.11.6　getTupleType()メソッド

形式

```
com.Hitachi.soft.hsdp.exadaptor.common.data.TupleType getTupleType()
```

説明

次のタプル種別を判別します。

- CONTROL_TUPLE：制御タプル
- NORMAL_TUPLE：制御タプル以外のタプル

パラメーター

なし。

例外

なし。

戻り値

タプル種別（enum）。

注意事項

特になし。

9.12　HSDPAdaptorManager クラス

クラスの階層

```
java.lang.Object
  |_com.Hitachi.soft.hsdp.exadaptor.api.HSDPAdaptorManager
```

説明

外部アダプターを操作するクラスです。

メソッド

次の表に，HSDPAdaptorManager クラスのメソッド一覧を示します。

項番	メソッド名	説明
1	init	外部アダプター※を初期化します。
2	term	外部アダプターを停止します。
3	getValue	外部アダプター定義ファイルの設定値を取得します。
4	openStreamInput	入力ストリームを開きます。
5	openStreamOutput	出力ストリームを開きます。

注※
複数の外部アダプターが同じマシン上で同時に実行される場合，アダプターごとに別のログ出力先を指定してください。これは，同じプロセス内でこのメソッドを複数回実行する場合にも適用されます。同じログ出力先を指定すると，操作を確実に実施できません。
このメソッドの後にterm()メソッドを実行しないと，メモリリークが発生します。

9.12.1　init(java.lang.String filePath)メソッド

形式

```
public static HSDPAdaptorManager init(java.lang.String filePath)
throws HSDPAdaptorException
```

説明

外部アダプターを初期化します。

メモ

複数の外部アダプターが同じマシン上で同時に実行される場合，アダプターごとに別のログ出力先を指定してください。これは，同じプロセス内でこのメソッドを複数回実行する場合にも適用されます。同じログ出力先を指定すると，操作を確実に実施できません。

このメソッドの後にterm()メソッドを実行しないと，メモリリークが発生します。

パラメーター

filePath

ファイルのパスです。相対パスを使用する場合，外部アダプターが実行されるディレクトリからの相対パスを指定します。

例外

例外	発生条件
HSDPAdaptorException	外部アダプター定義ファイルが存在しません。 外部アダプター定義ファイルの読み込みに失敗しました。 外部アダプター定義ファイルの設定値が誤っています。 ログファイルの初期化でエラーが発生しました。

戻り値

HSDPAdaptorManager クラスのインスタンス。

9.12.2 term()メソッド

形式

```
public void term()
throws HSDPAdaptorException
```

説明

外部アダプターを停止します。

送信および受信キューのデータを破棄し，すべての接続を終了して外部アダプターを停止します。

パラメーター

なし

例外

例外	発生条件
HSDPAdaptorException	外部アダプターはすでに停止しています。

戻り値

なし

9.12.3 getValue(java.lang.String key)メソッド

形式

```
public java.lang.String getValue(java.lang.String key)
throws HSDPAdaptorException
```

説明

外部アダプター定義ファイルの設定値を取得します。

パラメーター

key

定義パラメーターです。外部アダプター定義ファイルを指定せずにパラメーターを指定した場合，デフォルト値が返されます（デフォルト値がない場合は，null が返されます）。

例外

例外	発生条件
HSDPAdaptorException	外部アダプターはすでに停止しています。

戻り値

パラメーターに対応する値。デフォルト値がない場合は，null になります。

9.12.4 openStreamInput(java.lang.String targetName)メソッド

形式

```
public HSDPStreamInput openStreamInput(java.lang.String targetName)
throws HSDPAdaptorException
```

説明

入力ストリームを開きます。

SDP ブローカーから指定された宛先名に対応する入力ストリームを検索し，検索結果を基に入力ストリームとの接続を確立します。

入力ストリームとの接続を最大hsdp.tcp.connect.timeout 秒待ちます。接続に成功すると，メソッドは正常に終了します。

パラメーター

targetName

宛先名です。外部アダプター定義ファイルのhsdp.target.name.*n* パラメーターの設定値です。

例：openStreamInput (manager.getValue("hsdp.target.name.1"))

例外

例外	発生条件
HSDPAdaptorException	外部アダプターはすでに停止しています。 指定した引数の形式が正しくありません。 指定したストリームが入力ストリームではありません。 SDP ブローカーとの接続の試行回数が上限に達しました。 メソッドのパラメーターが，同一ストリームを含める複数のストリームで指定されています。 引数に指定されたストリーム間で情報が一致しません。 入力ストリームとの接続に失敗しました。

戻り値

HSDPStreamInput のインスタンス。

9.12.5 openStreamOutput(java.lang.String targetName)メソッド

形式

```
public HSDPStreamOutput openStreamOutput(java.lang.String targetName)
throws HSDPAdaptorException
```

説明

出力ストリームを検索します。

SDP ブローカーから指定された宛先名に対応する出力ストリームを検索します。出力ストリームの検索に成功すると，メソッドは正常に終了します。

パラメーター

targetName

宛先名です。外部アダプター定義ファイルのhsdp.target.name.*n* パラメーターの設定値です。

例：openStreamOutput (manager.getValue("hsdp.target.name.1"))

例外

例外	発生条件
HSDPAdaptorException	外部アダプターはすでに停止しています。 指定した引数の形式が正しくありません。 指定したストリームが出力ストリームではありません。 SDP ブローカーとの接続の試行回数が上限に達しました。 パラメーターに複数のストリームが指定されています。

戻り値

HSDPStreamOutput のインスタンス。

9.13 HSDPColumnType 列挙型

説明

列挙型定数の詳細
BYTE public static final HSDPColumnType BYTE BYTE 型。
SHORT public static final HSDPColumnType SHORT SHORT 型。
INT public static final HSDPColumnType INT INT 型。
LONG public static final HSDPColumnType LONG LONG 型。
FLOAT public static final HSDPColumnType FLOAT FLOAT 型。
DOUBLE public static final HSDPColumnType DOUBLE DOUBLE 型。
CHAR public static final HSDPColumnType CHAR CHAR 型。
VARCHAR public static final HSDPColumnType VARCHAR VARCHAR 型。
TIMESTAMP public static final HSDPColumnType TIMESTAMP TIMESTAMP 型。
DATE public static final HSDPColumnType DATE DATE 型。
TIME public static final HSDPColumnType TIME TIME 型。
BIG_DECIMAL

列挙型定数の詳細
public static final HSDPColumnType BIG_DECIMAL BIG_DECIMAL 型。

次の表に，HSDPColumnType 列挙型メソッドの一覧を示します。

項番	メソッド名	説明
1	values	宣言された順に，この列挙型定数を含むシーケンスが返されます。
2	valueOf	指定した名前の列挙型定数が返されます。
3	getNumber	列挙型に割り当てられた番号が返されます。

9.13.1 values()メソッド

形式

```
public static HSDPColumnType[] values()
```

説明

宣言された順に，この列挙型定数を含むシーケンスが返されます。このメソッドは，次のように定数を繰り返す場合に使用できます。

```
for (HSDPColumnType c : HSDPColumnType.values())
System.out.println(c);
```

パラメーター

なし。

例外

なし。

戻り値

宣言された順の，この列挙型の定数を含むシーケンス。

9.13.2 valueOf(java.lang.String name)メソッド

形式

```
public static HSDPColumnType valueOf(java.lang.String name)
```

説明

指定した名前の列挙型定数が返されます。文字列は，この型の列挙型定数を宣言するときに使用された識別子と完全に一致させる必要があります（余分な空白文字は指定できません）。

パラメーター

name

返された列挙型定数の名前です。

例外

名前	説明
java.lang.IllegalArgumentException	この列挙型に指定した名前の定数がない場合です。
java.lang.NullPointerException	引数が null の場合です。

戻り値

指定した名前の列挙型定数。

9.13.3 getNumber()メソッド

形式

```
public int getNumber()
```

説明

列挙型に割り当てられた番号が返されます。

パラメーター

なし。

例外

なし。

戻り値

列挙型に対応する数値。

9.14 HSDPConnStatus 列挙型

説明

列挙型定数の詳細
BUSY ビジー状態です。
CONNECTED 接続します。
CONNECTING 接続中です。
DISCONNECTED 切断します。
INITIALIZING 初期のステータスです。

次の表に，HSDPCoonnStatus 列挙型メソッドの一覧を示します。

項番	メソッド名	説明
1	values	宣言された順に，この列挙型定数を含むシーケンスが返されます。
2	valueOf	指定した名前の列挙型定数が返されます。

9.14.1 values()メソッド

形式

```
public static HSDPConnStatus[] values()
```

説明

宣言された順に，この列挙型定数を含むシーケンスが返されます。このメソッドは，次のように定数を繰り返す場合に使用できます。

```
for (HSDPConnStatus c : HSDPConnStatus.values())System.out.println(c);
```

パラメーター

なし。

例外

なし。

戻り値

宣言された順の，この列挙型の定数を含むシーケンス。

9.14.2 valueOf(java.lang.String name)メソッド

形式

```
public static HSDPConnStatus valueOf(java.lang.String name)
```

説明

指定した名前の列挙型定数が返されます。文字列は，この型の列挙型定数を宣言するときに使用された識別子と完全に一致させる必要があります（余分な空白文字は指定できません）。

パラメーター

name

返された列挙型定数の名前です。

例外

名前	説明
java.lang.IllegalArgumentException	この列挙型に指定した名前の定数がない場合です。
name.java.lang.NullPointerException	引数が null の場合です。

戻り値

指定した名前の列挙型定数。

9.15 例外

表 9-3 public class HSDPAdaptorDispatchException extends HSDPAdaptorException

コンストラクターの詳細
HSDPAdaptorDispatchException public HSDPAdaptorDispatchException(java.lang.String message)
HSDPAdaptorDispatchException public HSDPAdaptorDispatchException(java.lang.Throwable cause)

表 9-4 public class HSDPAdaptorException

コンストラクターの詳細
HSDPAdaptorException HSDPAdaptorException(java.lang.String message)
HSDPAdaptorException HSDPAdaptorException(java.lang.Throwable cause)

表 9-5 public class HSDPAdaptorException extends HSDPAdaptorException

コンストラクターの詳細
HSDPAdaptorQueueSizeLackException HSDPAdaptorQueueSizeLackException(java.lang.String message)
HSDPAdaptorQueueSizeLackException HSDPAdaptorQueueSizeLackException(java.lang.Throwable cause)

10 データ送受信用 API を使用したサンプルプログラム

この章では，データ送受信 API を使用したサンプルプログラムについて説明します。

10.1　インプロセス連携カスタムアダプターのサンプルプログラム

この節では，インプロセス連携カスタムアダプターのサンプルプログラムについて説明します。

10.1.1　クエリグループの定義内容（インプロセス連携カスタムアダプター）

ここでは，クエリグループの定義の内容について説明します。

このクエリグループでは，送信先ストリーム名としてdata0 を定義して，受信元ストリーム名として，ポーリング受信用にfilter1 を定義しています。

定義内容を次に示します。なお，大文字小文字の表記はサンプルと異なります。

```
REGISTER STREAM data0(name VARCHAR(10), num BIGINT);
REGISTER QUERY filter1 ISTREAM(SELECT * FROM data0[ROWS 1] WHERE data0.num <= 24);
```

10.1.2　インプロセス連携送受信制御 AP の内容

ここでは，インプロセス連携送受信制御 AP（Inprocess_Main.java）の内容について説明します。このプログラムは，タプルを送受信するアプリケーションを制御する，メインプログラムです。

ソースコードを次に示します。なお，「//【1】」「//～【1】」などのコメントは，以降の説明の番号と対応しています。

サンプルプログラムの内容

```
package samples;

import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.inprocess.StreamInprocessUP;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientInitializeException;
import jp.co.Hitachi.soft.sdp.api.logger.SDPClientLoggerFactory;
import jp.co.Hitachi.soft.sdp.api.logger.SDPClientLogger;
import jp.co.Hitachi.soft.sdp.api.logger.util.LogKind;

public class Inprocess_Main implements StreamInprocessUP {

  // 受信用スレッドオブジェクト
  private Inprocess_Receiver receiver = null;

  // 送信用スレッドオブジェクト
  private Inprocess_Sender sender = null;

  // SDPサーバとのコネクター
  private SDPConnector connector = null;

  // ロガー
```

```
private SDPClientLogger logger = null;

// 【1】
public void execute(SDPConnector sc) {
  //～【1】

  connector = sc;

  // 【2】
  // ロガーを初期化します
  logger = SDPClientLoggerFactory.getLogger(Inprocess_Main.class);
  try {
      logger.initialize("InprocessAPTest");
  } catch (SDPClientInitializeException e) {
      throw new RuntimeException(e);
  }
  //～【2】

  // 【3】
  // ポーリング用受信スレッドを開始します
  receiver = new Inprocess_Receiver(sc, logger);
  receiver.start();

  // 送信スレッドを開始します
  sender = new Inprocess_Sender(sc, logger);
  sender.start();
  //～【3】
}

// 【1】
public void stop() {
//～【1】

  // 【4】
  try {
    // 送信スレッドを停止します
    if(sender != null) {
      sender.terminate();
      sender.join();
    }
    // 受信スレッドを停止します
    if(receiver != null){
      receiver.terminate();
      receiver.join();
    }
  } catch (InterruptedException e) {
    logger.putMessage(LogKind.ERROR, "Main : " + e.getMessage());
  }

  try {
    // コネクターを閉じます
    connector.close();
  } catch (SDPClientException e) {
    logger.putMessage(LogKind.ERROR, "Main : " + e.getMessage());
  }
      // ロガーを終了します
      logger.terminate();
  //～【4】
```

```
  }
}
```

ソースコードの内容について説明します。

1. このアプリケーションのInprocess_Main クラスは，StreamInprocessUP インタフェースを実装したクラスです。このため，StreamInprocessUP インタフェースで定義されているexecute メソッドとstop メソッドを実装します。
2. アプリケーション内で使用するロガーを初期化します。
3. アプリケーション開始時に実行されるexecute メソッドで，データ受信スレッドとデータ送信スレッドを開始します。この際，データ受信スレッドとデータ送信スレッドにロガーを渡します。
4. アプリケーション停止時に実行されるstop メソッドでは，3.で開始したデータ受信スレッドとデータ送信スレッドを停止します。その後，SDPConnector 型オブジェクトのclose メソッドを呼び出して，SDPConnector を閉じたあと，ロガーを終了します。

10.1.3 インプロセス連携データ送信 AP の内容

ここでは，インプロセス連携データ送信 AP（Inprocess_Sender.java）のサンプルの内容について説明します。

メモ

このサンプルプログラムでは，送信専用のスレッドクラスを別に定義して，クライアント AP のメインクラス（StreamInprocessUP インタフェースを実装したInprocess_Main クラス）側からそのスレッドを生成して送信を委譲する例を示しています。これ以外の方法として，Inprocess_Main クラス自身が SDP サーバにタプルを送信する処理もできます。

ソースコードを次に示します。なお，「//【1】」「//～【1】」などのコメントは，以降の説明の番号と対応しています（これらのコメントは，実際のサンプルプログラムには記載されていません）。

サンプルプログラムの内容

```
package samples;

import java.util.ArrayList;

import jp.co.Hitachi.soft.sdp.common.data.StreamTuple;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupHoldException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStopException;
import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.StreamInput;
import jp.co.Hitachi.soft.sdp.api.logger.SDPClientLogger;
import jp.co.Hitachi.soft.sdp.api.logger.util.LogKind;

public class Inprocess_Sender extends Thread{
```

```
// SDPサーバとのコネクター
private SDPConnector connector;
// ロガー
private SDPClientLogger logger;
//【6】
// スレッドの起動状態
private volatile boolean running = true;
//～【6】

// 送信スレッドの生成処理
public Inprocess_Sender(SDPConnector sc, SDPClientLogger logger) {
  this.connector = sc;
  this.logger = logger;
}

ArrayList<StreamTuple> list = new ArrayList<StreamTuple>();

public void run()  {

  final String group_name = "Inprocess_QueryGroupTest";
  final String stream_name = "DATA0";

  //【1】
  // 送信ストリームオブジェクト
  StreamInput si = null;

  // 送信ストリームに接続します
  try {
    si = connector.openStreamInput(group_name,stream_name);
  } catch (SDPClientException e) {
    //  【2】
    logger.putMessage(LogKind.ERROR, "Sender :" + e.getMessage());
    // ～【2】
    running = false;
  }
  //～【1】

  //【3】
  // データを50個送信します
  for(int i = 0; i < 50;i++){
    if(!running) {
      break;
    }

    // 送信データを1個生成します
    Object[] data = new Object[]{
      new String("ad"+i),
      new Long(i)
    };

    // タプルオブジェクトを1個生成します
    StreamTuple tuple = new StreamTuple(data);

    try {
      //1秒10件（100ミリ秒に1件）の送信頻度となるようにスリープします
      Thread.sleep(100);
    } catch (InterruptedException e) {
```

```
        logger.putMessage(LogKind.ERROR, "Sender : Thread is interrupted");
        logger.putStackTrace(e);
      }
      //～【3】

      //【4】
      // 単一のタプルを送信します
      try {
        si.put(tuple);
      } catch (SDPClientQueryGroupStopException e) {
        //クエリグループが停止しています(hsdpcqlstopコマンド)
        //何もせずに処理を続行します
      } catch (SDPClientQueryGroupHoldException e) {
        //クエリグループが閉塞しています
        //何もせずに処理を続行します
      } catch (SDPClientException e) {
        logger.putMessage(LogKind.ERROR, "Sender :" + e.getMessage());
        running = false;
      }
    }

    if(running) {
      try {
        si.putEnd();
      } catch (SDPClientQueryGroupStopException e) {
        //クエリグループが停止しています(hsdpcqlstopコマンド)
        //何もせずに処理を続行します
      } catch (SDPClientQueryGroupHoldException e) {
        //クエリグループが閉塞しています
        //何もせずに処理を続行します
      } catch (SDPClientException e) {
        logger.putMessage(LogKind.ERROR, "Sender :" + e.getMessage());
      }
    }
    //～【4】

    //【5】
    try {
      if (si != null) {
        // 送信ストリームを閉じます
        si.close();
      }
    } catch (SDPClientException e) {
      logger.putMessage(LogKind.ERROR, "Sender :" + e.getMessage());
    }
    //～【5】

  }
  //【6】
  public void terminate() {
    logger.putMessage(LogKind.INFO, "Sender : terminate called");
    this.running = false;
  }
  //～【6】
}
```

ソースコードの内容について説明します。

1. 送信先ストリームの入力ストリームに接続して，StreamInput 型オブジェクトを取得します。
2. ロガーを使用して，発生したエラーをログへ出力します。
3. 送信データをオブジェクトとして生成して，タプルに設定します。
4. StreamInput 型オブジェクトを使用してput メソッドを呼び出し，1.で接続した入力ストリームに対してデータ（タプル）を送信します。すべてのデータの送信が完了したあと，putEnd メソッドで送信完了を通知します。
5. StreamInput 型オブジェクトを使用してclose メソッドを呼び出して，入力ストリームとの接続を切断します。
6. terminate メソッドにはタプルの送信を停止させる処理，つまりInprocess_Sender スレッドの終了条件を記述しています。

   terminate メソッドは，SDP サーバのスレッドから呼び出されるため，Inprocess_Receiver クラスのrunning フィールドは 2 つのスレッドから同時に読み取られる場合があります。このため，running フィールドは，volatile 属性にする必要があります。

10.1.4 インプロセス連携データ受信 AP の内容（ポーリング方式）

ここでは，インプロセス連携データ受信 AP（Inprocess_Receiver.java）のサンプルの内容について説明します。このプログラムでは，Inprocess_Receiver スレッドが100 ミリ秒間隔で SDP サーバに対してポーリング方式でタプル取得処理を実行します。このスレッドは，送受信制御プログラムであるInprocess_Main クラスによって起動されます。

ソースコードを次に示します。なお，「//【1】」「//～【1】」などのコメントは，以降の説明の番号と対応しています。

サンプルプログラムの内容

```
package samples;

import jp.co.Hitachi.soft.sdp.common.data.StreamTuple;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupHoldException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientQueryGroupStopException;
import jp.co.Hitachi.soft.sdp.common.exception.SDPClientEndOfStreamException;
import jp.co.Hitachi.soft.sdp.common.util.StreamTime;
import jp.co.Hitachi.soft.sdp.api.SDPConnector;
import jp.co.Hitachi.soft.sdp.api.StreamOutput;
import jp.co.Hitachi.soft.sdp.api.logger.SDPClientLogger;
import jp.co.Hitachi.soft.sdp.api.logger.util.LogKind;

public class Inprocess_Receiver extends Thread {

  // SDPサーバとのコネクター
  private SDPConnector connector;
  // ロガー
  private SDPClientLogger logger;
```

```
// 【3】
// スレッドの起動状態
private volatile boolean running = true;
//～【3】

// 受信スレッドの生成処理
public Inprocess_Receiver(SDPConnector sc, SDPClientLogger logger) {
  this.connector = sc;
  this.logger = logger;
}

public void run() {

  final String groupName = "Inprocess_QueryGroupTest";
  final String streamName = "FILTER1";

  // 結果ストリームオブジェクト
  StreamOutput so = null;

  // 結果ストリームに接続します
  try {
    so = connector.openStreamOutput(groupName,streamName);
  } catch (SDPClientException sce) {
      // 【2】
      logger.putMessage(LogKind.ERROR, "Receiver : " + sce.getMessage());
      // ～【2】
    running = false;
  }

  // 【1】
  // タプルの受信処理を開始します
  while(running){
  //～【1】
    try {
      // 送信側のスリープ時間に合わせてスリープします
      Thread.sleep(100);
    } catch (InterruptedException e) {
      logger.putMessage(LogKind.ERROR, "Receiver : Thread is interrupted");
      logger.putStackTrace(e);
    }

    // タプルオブジェクト
    StreamTuple tuple = null;

    // タプルをポーリングで取得します
    try {
      tuple = so.get();
    } catch (SDPClientEndOfStreamException e) {
      //送信ストリームデータが終了しました
      logger.putMessage(LogKind.INFO, "Receiver : " + e.getMessage());
      break;
    } catch (SDPClientQueryGroupStopException e) {
      //クエリグループが停止しています(hsdpcqlstopコマンド)
      logger.putMessage(LogKind.ERROR, "Receiver : " + e.getMessage());
      break;
    } catch (SDPClientQueryGroupHoldException e) {
      //クエリグループが閉塞しています
      logger.putMessage(LogKind.ERROR, "Receiver : " + e.getMessage());
```

```
        break;
      } catch (SDPClientException e) {
        logger.putMessage(LogKind.ERROR, "Receiver : " + e.getMessage());
        break;
      }

      //受信したタプルを表示します
      if (tuple != null) {
        Object[] data = tuple.getDataArray();
        String val = (String) data[0];
        Long id = (Long) data[1];
        StreamTime time = tuple.getSystemTime();
        logger.putMessage(LogKind.INFO, "Receiver : Tuple Get on " + streamName
          +" [ VAL="+val+", ID="+id+", TIME="+time.toString()+" ]");
      }
    }

    try {
      if (so != null) {
        // 結果ストリームを閉じます
        so.close();
      }
    } catch (SDPClientException e) {
      logger.putMessage(LogKind.ERROR, "Receiver : " + e.getMessage());
    }

  }


  // 【3】
  public void terminate() {
      logger.putMessage(LogKind.INFO, "Receiver : terminate called");
    this.running = false;
  }
  //～ 【3】
}
```

ソースコードの内容について説明します。

1. データ受信処理は，3.のterminate メソッドによって終了条件が設定されるまで，実行されます。
2. ロガーを使用して，発生したエラーをログへ出力します。
3. terminate メソッドには，タプルの受信を停止させる処理，つまりInprocess_Receiver スレッドの終了条件を記述しています。

   terminate メソッドは SDP サーバのスレッドから呼び出されるため，Inprocess_Receiver クラスの running フィールドは 2 つのスレッドから同時に読み取られる場合があります。このため，running フィールドは，volatile 属性にする必要があります。

11 HSDP クライアントライブラリの使用方法

この章では，HSDP クライアントライブラリの使用方法について説明します。

11.1 HSDP クライアントライブラリの概要

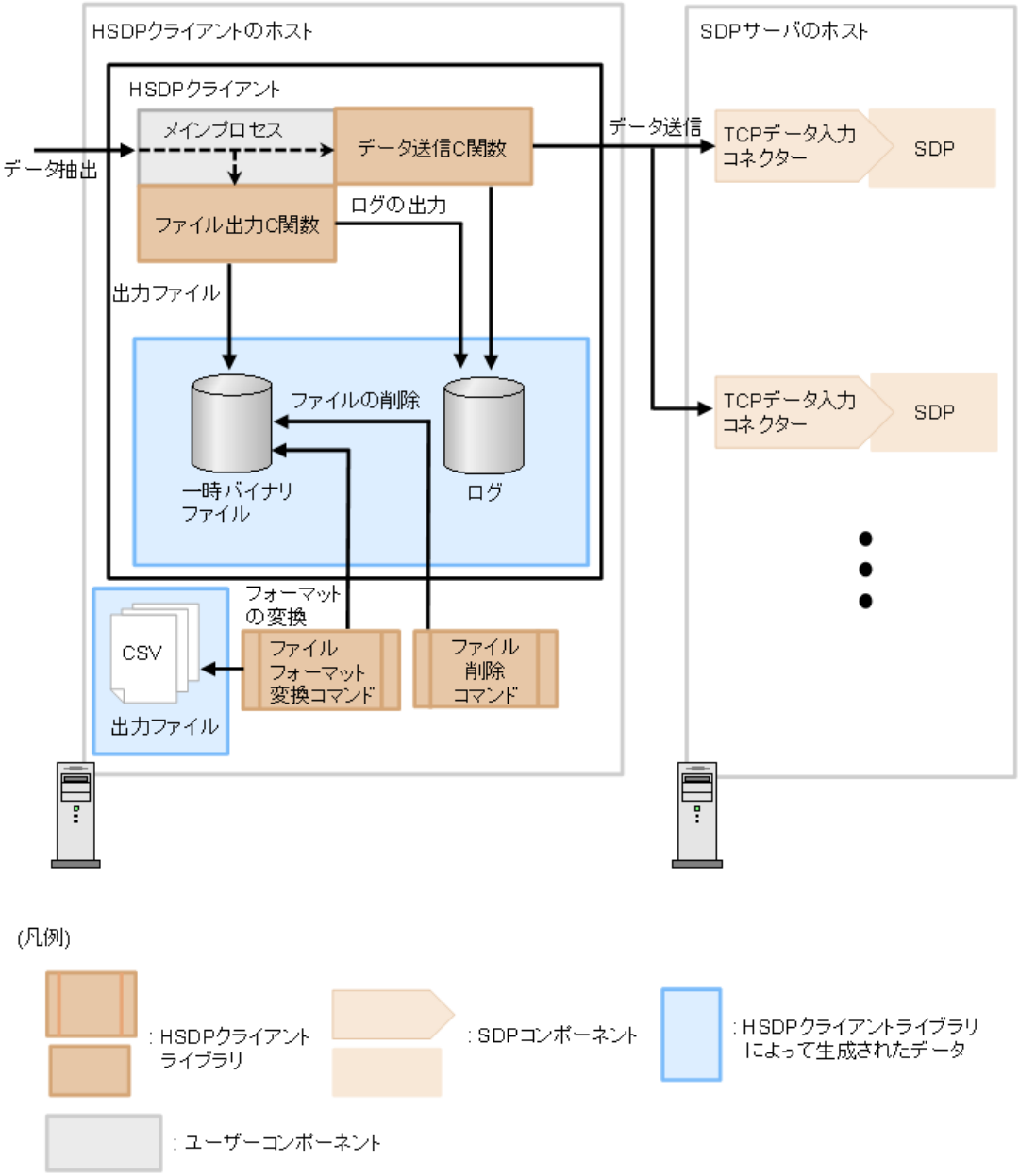
HSDP クライアントライブラリは，ファイルの出力や，HSDP 上の TCP データ入力コネクターへの高性能（高速）なデータ送信を実行する，HSDP クライアントを開発するためのライブラリです。

HSDP クライアントライブラリは，次の 3 つの機能を提供します。

- データ送信
- ファイル出力
- トラブルシュート

HSDP クライアントライブラリには，C 関数とコマンドが含まれています。C 関数には，初期化，終了，データ送信，およびファイル出力用アドレスを取得するための関数が含まれています。HSDP クライアントの開発者は，これらの関数をインストールする必要があります。コマンドは，ファイルのフォーマット変換とファイルの消去（削除）に使用されます。これらのコマンドは，HSDP クライアントで出力されたファイルをほかのプログラムで読み込めるよう任意のフォーマットに変換したり，これらのファイルを削除したりする場合に使用できます。次の図に，HSDP クライアントライブラリの概要を示します。

図 11-1　HSDP クライアントライブラリの概要



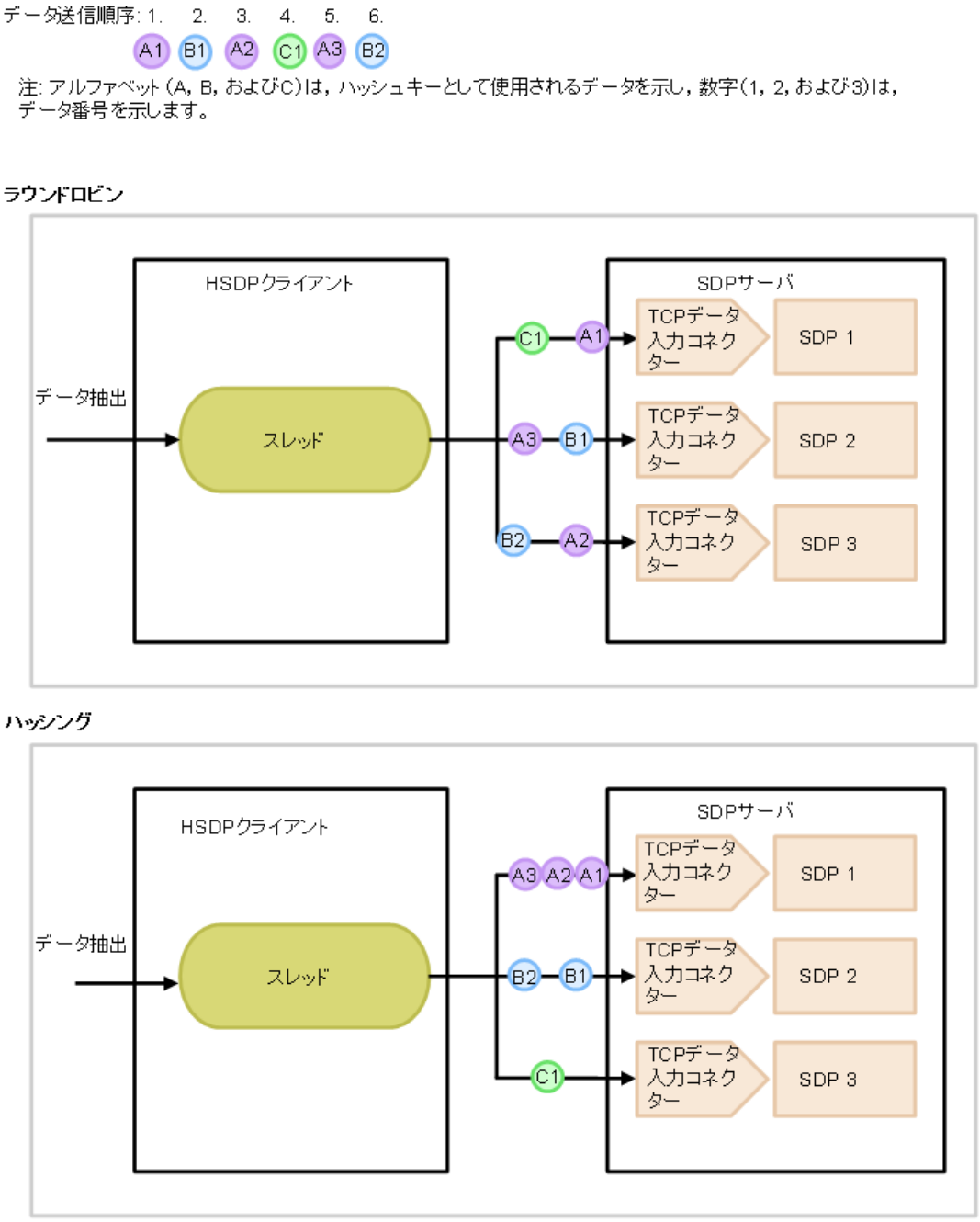
11.2 データ送信機能

HSDP クライアントのデータ送信機能は，TCP データ入力コネクターへの高性能（高速）なデータ送信に使用されます。

次の表でデータ送信機能について説明します。

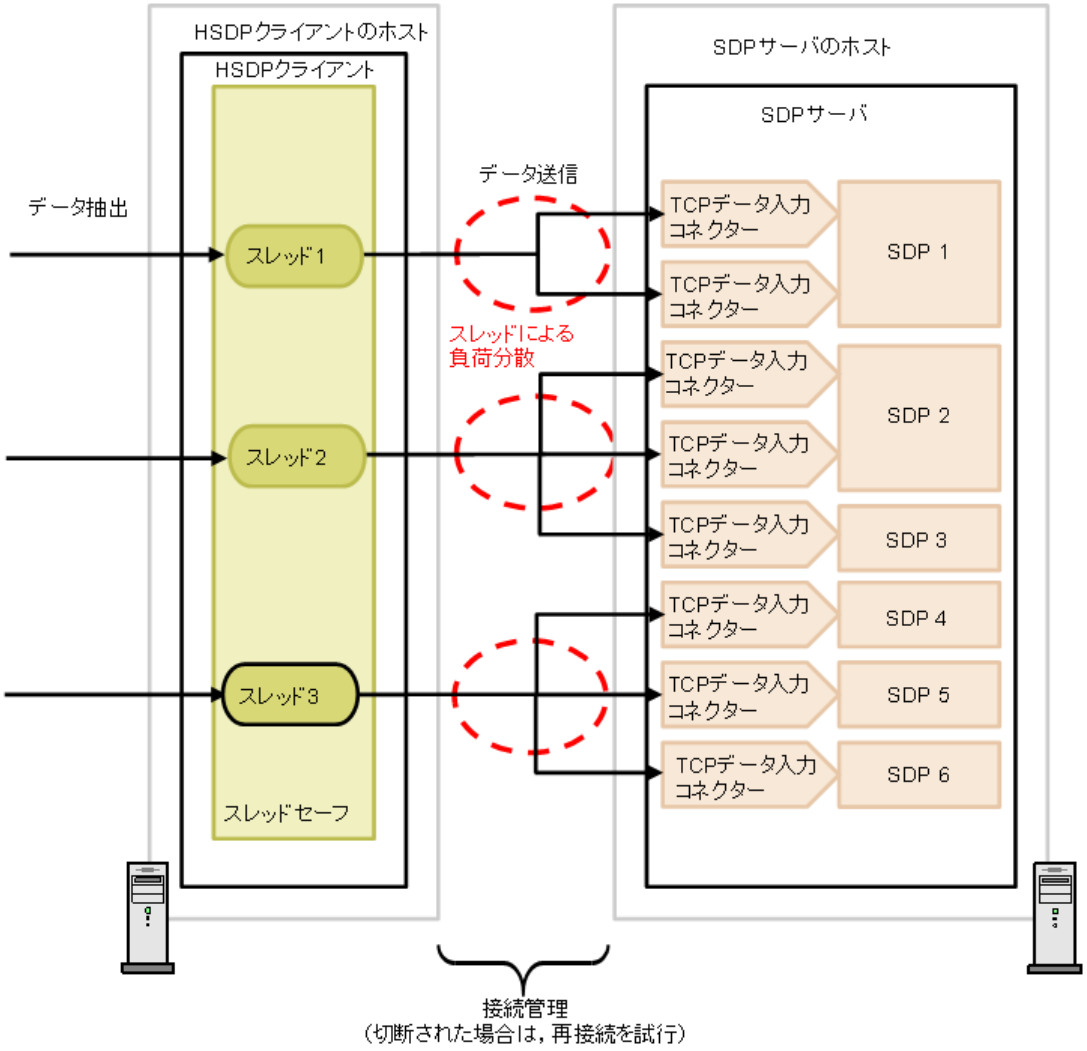
機能	説明
HSDP へのデータ送信	HSDP クライアントは，TCP データ入力コネクターにデータを送信します。HSDP クライアントと HSDP は，別々のホストにインストールできます。送信するデータの形式を指定できます。
接続管理	HSDP クライアントと TCP データ入力コネクターの接続が切断されると，HSDP クライアントは，自動的に再接続を試みます。
スレッドセーフ	HSDP クライアントは，マルチスレッドを使用することで，複数の TCP データ入力コネクターにデータを送信できます。HSDP クライアントライブラリは，マルチスレッドをサポートします。
負荷分散	データ送信スレッドは，次の 2 種類の負荷分散の方法によって，複数の HSDP にデータを送信できます。 ラウンドロビン データを各 TCP データ入力コネクターに順番に送信することで，作業負荷を分散します。これは，同等の作業負荷を SDP サーバに分散させる場合に使用できます。 ハッシング 複数の列から成るハッシュキーに基づき，複数の TCP データ入力コネクターにデータを送信して，作業負荷を分散します。同じキーを持つデータは同じ SDP サーバに送信できます。例えば，モニター ID が A のデータは，SDP サーバ X に送信できます。 負荷分散については，次の図を参照してください。

図 11-2　負荷分散の方法の概要（HSDP クライアントの 1 スレッドから 3 つの TCP データ入力コネクターに分散する場合）



次の図に，データ送信機能の概要を示します。

図 11-3　データ送信機能の概要



11.3 ファイル出力機能

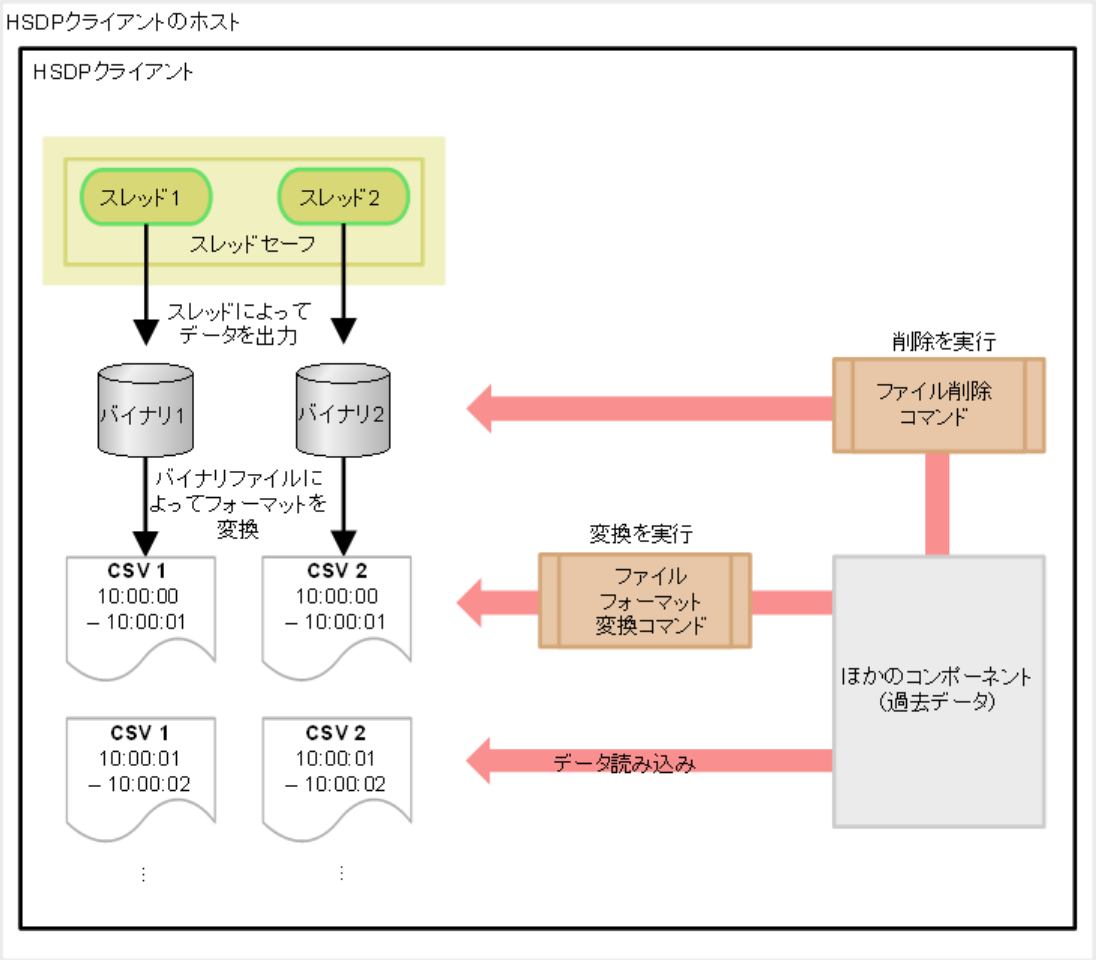
HSDP クライアントのファイル出力機能は，高性能（高速）でバイナリファイルにデータを出力します。ほかのプログラムで使用するために，ファイルをほかのフォーマットに変換するコマンドと，ファイルを削除するコマンドも提供されます。

次の表でファイルの出力機能について示します。

機能	説明
バイナリファイルのデータ出力	HSDP クライアントは，データのコピーだけを使用して，バイナリファイルにデータを出力できます。HSDP クライアントの各スレッドは，ファイル出力機能によって返されたアドレスにデータをコピーします。これによって，指定したディレクトリにバイナリファイルとしてデータが出力されます。
スレッドセーフ	HSDP クライアントは，マルチスレッドを使用することで，データをバイナリファイルとして出力できます。HSDP クライアントライブラリは，マルチスレッドをサポートします。
ファイルフォーマット変換コマンド（hsdpcliconv コマンド）	ファイルフォーマット変換コマンド（hsdpcliconv コマンド）は，HSDP クライアントのファイル出力機能によって出力されたバイナリファイルを別のファイルフォーマットに変換する場合に使用できます。ファイルフォーマット変換では，次のフォーマットがサポートされています。 • CSV
ファイル削除コマンド（hsdpclirm コマンド）	ファイル削除コマンド（hsdpclirm コマンド）は，HSDP クライアントのファイル出力機能によって出力されたバイナリファイルを削除する場合に使用できます。このコマンドには，次の 2 種類の条件を設定できます。 時間に基づく条件 指定した時間の前にバイナリファイルが削除されます。 容量に基づく条件 バイナリファイルは，バイナリファイルの合計サイズが指定したサイズより小さくなるよう削除されます。

次の図に，ファイル出力機能の概要を示します。

図 11-4　ファイル出力機能の概要



11.4　トラブルシュート機能

HSDP クライアントライブラリは，C 関数が呼び出された場合や，データ送信時やファイルの出力時にエラーが発生した場合などに，独自のメッセージおよびトレースログを出力します。

11.5　HSDP クライアントライブラリを使用する

ここでは，HSDP クライアントライブラリを使用して，HSDP クライアントを開発する方法を説明します。

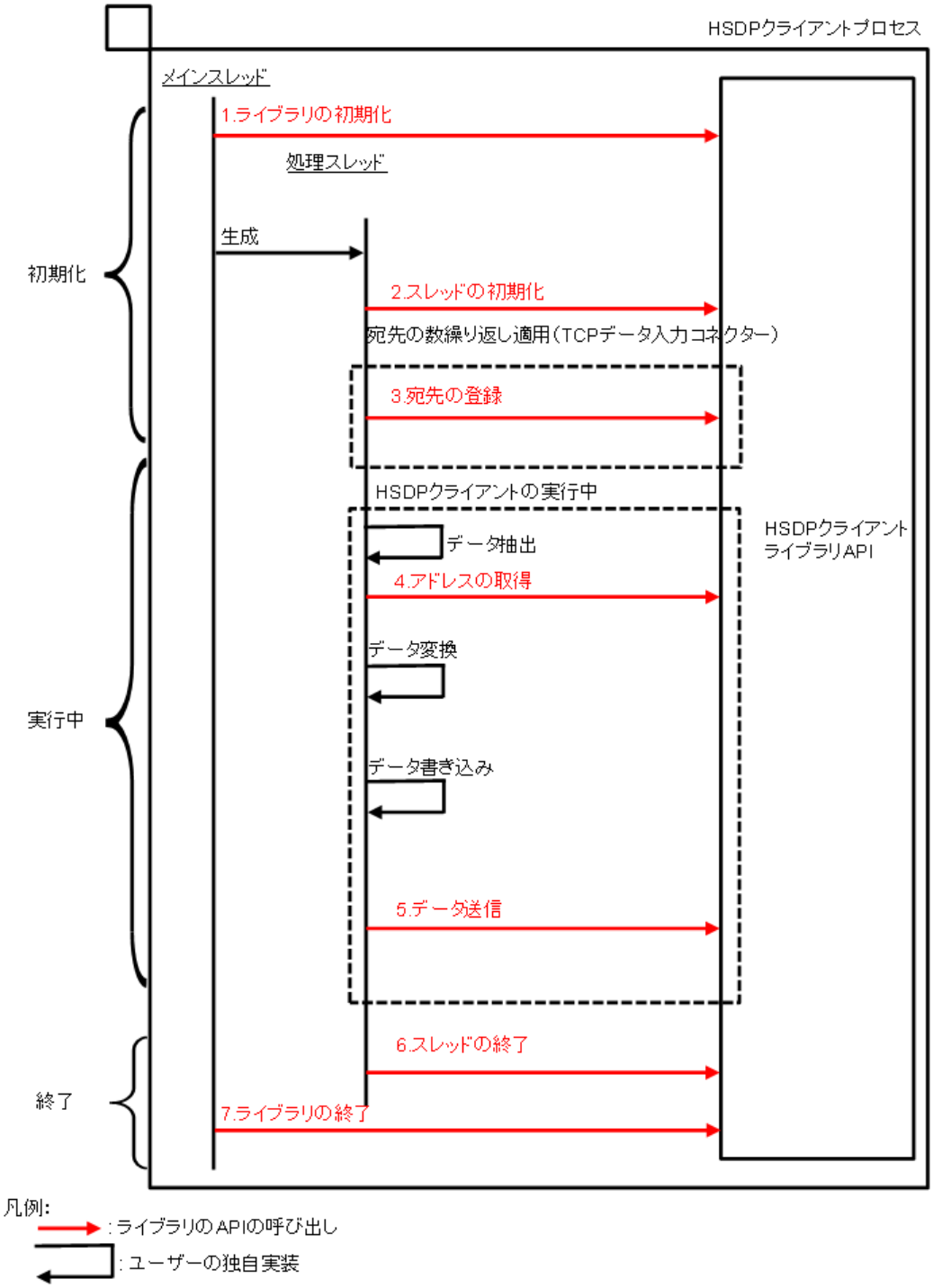
11.5.1　HSDP クライアントの開発

ここでは，HSDP クライアントライブラリを使用して，HSDP クライアントを開発する方法を説明します。

前提条件

HSDP クライアントの開発者は，次の図に示す手順を実行します。

図 11-5　HSDP クライアントの実装手順の概要



操作手順

1. ライブラリを初期化します。

   HSDP クライアントの処理の開始時には，hsdp_cli_init()を 1 度呼び出します。hsdp_cli_init()は，HSDP クライアントライブラリによって使用されるパラメーターを指定します。

2. スレッドを初期化します。

   HSDP クライアントの処理に対する処理スレッドの生成時には，hsdp_cli_init_thd()を呼び出します。

hsdp_cli_init_thd()によって返されるスレッド ID を手順 3.から手順 6.の関数の引数に設定します。

3. 宛先を登録します。

スレッドを初期化後，hsdp_cli_reg_dst()をデータの各送信先（TCP データ入力コネクター）に対して 1 度呼び出します。hsdp_cli_reg_dst()は，接続元および接続先のアドレス情報（IP アドレスとポート番号）を登録し，登録した宛先との接続を確立します。

登録した宛先を管理するため，HSDP クライアントは，hsdp_cli_reg_dst()によって返された宛先 ID を保存します。

接続を確立できない場合は，指定した宛先を見直してください。

4. アドレスを取得します。

hsdp_cli_get_addr()を呼び出し，ファイル出力のためのメモリアドレスを取得します。データがそのメモリアドレスにコピーされると，OS はそのデータをバイナリファイルとして出力します。データ形式は，ネットワークバイトオーダーでなければなりません。

送信エラーやその他のエラーでデータを失わないよう，次のタスクを完了します。

- TCP データ入力コネクターに送信したデータをバイナリファイルに出力します。
- TCP データ入力コネクターにデータを送信する前に，hsdp_cli_get_addr()を呼び出します。
- 出力データは，データソースのタイムスタンプで昇順である必要があります。
- 出力されたバイナリファイル名は，変更できません。

5. データを送信します。

hsdp_cli_send()を呼び出して，TCP データ入力コネクターにデータを送信します。マニュアル『Hitachi Streaming Data Platform 概説』に記載されているとおり，データは，TCP データの形式である必要があります。

HSDP クライアントは，上記の条件や送信を確認できないため，データがファイルに出力されたあとにhsdp_cli_send()を呼び出します。

データをコピーしないでデータを送信するには，hsdp_cli_send()の引数としてhsdp_cli_get_addr()から取得した，送信するデータが格納されているメモリアドレスを使用します。

接続の切断または接続がビジー状態であることが原因で送信エラーが発生する場合は，負荷分散方式によって，次の手順（図中に示す）のうちの 1 つを実行します。

図 11-6　データ送信中にエラーが発生した場合のプロセスフロー（ラウンドロビン）

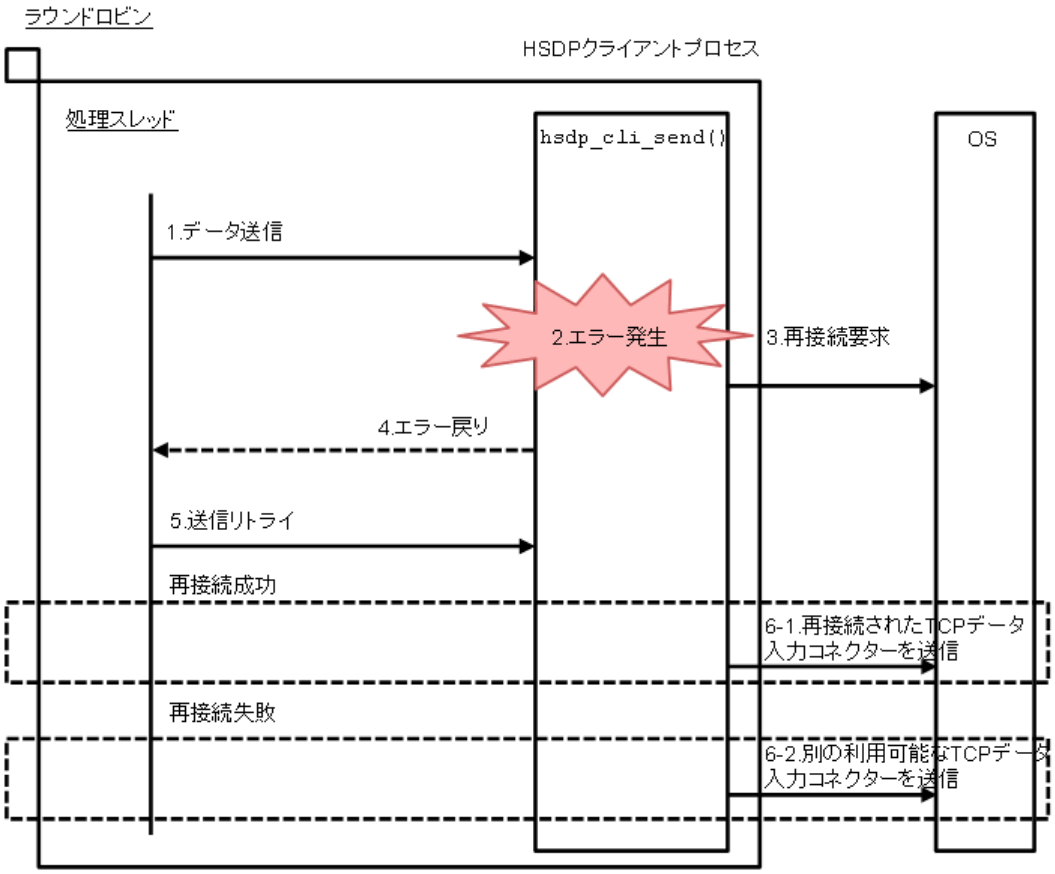
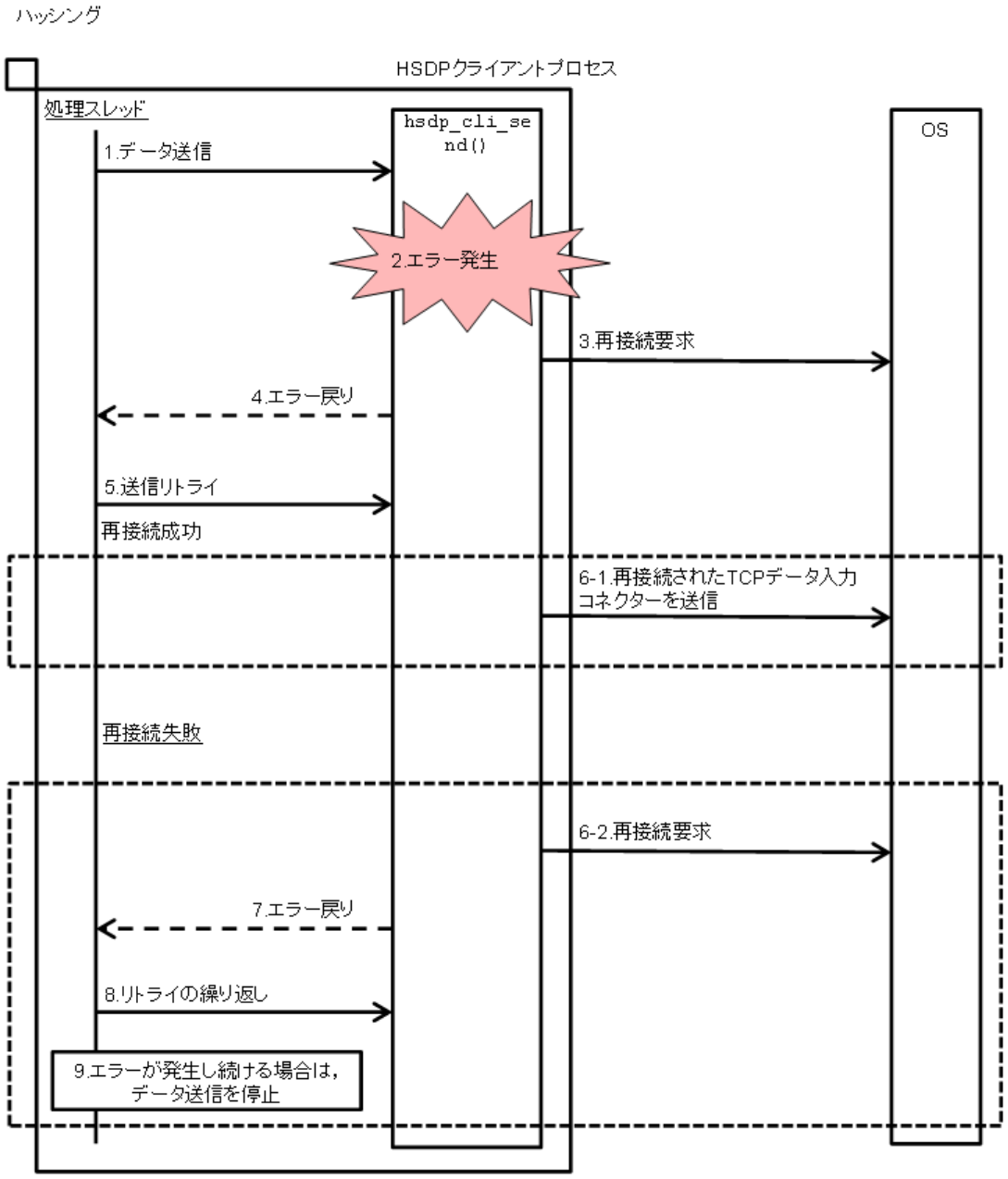


図 11-7　データ送信中にエラーが発生した場合のプロセスフロー（ハッシング）



6. スレッドを終了します。

スレッドの終了時には，hsdp_cli_term_thd()を呼び出し，初期化時に登録した宛先とともに接続を解放します。

7. ライブラリを終了します。

HSDP クライアントプロセスの終了時には，hsdp_cli_term()を呼び出します。

すべてのスレッドが終了するのを待ってから，hsdp_cli_term()を呼び出します。hsdp_cli_term()は，1 つ以上のスレッドが終了していないとエラーを返します。

C 言語で記述された HSDP クライアントプログラムの例を「11.8　HSDP クライアントプログラムのサンプルプログラム」に示しています。

11.5.2 HSDP クライアントのコンパイル

操作手順

1. ソースファイルをコンパイルします。
   コンパイラーを使用して，HSDP クライアントのソースファイルをコンパイルします。C 言語には，GCC コンパイラーを使用します。

2. リンケージを確立します。
   実行可能ファイルを作成するには，次のダイナミックライブラリにリンクします。：libhsdpcli.so
   GCC の使用によるコンパイルとリンクの例を次に示します。

```
gcc -o client client.c -I/var/hsdpcli -L/var/hsdpcli -Wl,-Bdynamic -lpthread
-lhsdpcli
```

   この例の条件は，次のとおりです。

   - 実行可能ファイルの名前は，client です。
   - ソースファイルの名前は，client.c です。
   - hsdpcli.h が含まれるディレクトリのパスは，/var/hsdpcli です。
   - libhsdpcli.so が含まれるディレクトリのパスは，/var/hsdpcli です。
   - ダイナミックライブラリにリンクした実行可能ファイルを作成します。
   - pthread ライブラリにリンクします。

11.5.3 HSDP クライアントの操作

環境変数を設定する

HSDP クライアントライブラリを呼び出すには，次の環境変数を設定します。

環境変数	値
LD_LIBRARY_PATH	libhsdpcli.so が格納されている絶対ディレクトリパスを指定します (hsdpclilib.tar.gz が展開されたディレクトリ)。

HSDP クライアントを起動，停止する

HSDP クライアントを起動，停止する手順は，HSDP クライアントの実装によって異なります。

HSDP クライアントを起動する前に SDP サーバを起動し，HSDP クライアントを停止したあとで SDP サーバを停止する必要があります。HSDP クライアントを再起動する場合は，前回のトレースログファイルが残っているため，これらを削除する必要があります。

HSDP クライアントの構成を変更する

HSDP クライアントライブラリの初期化時およびスレッドの初期化時に指定された HSDP クライアントパラメーターを変更するには，次の手順を実行します。

1. HSDP クライアントを停止します。
2. 構成を変更します。
3. HSDP クライアントを再起動します。

HSDP クライアントライブラリの初期化時とスレッドの初期化時に指定したパラメーターについては，「11.6.1　関数」の「(1)　hsdp_cli_init ()」と「(2)　hsdp_cli_init_thd ()」を参照してください。

出力ファイルフォーマットを変換する

出力ファイルをバイナリフォーマットから別のフォーマット（CSV など）に変換するには，hsdpcliconv コマンドを実行します。出力データビューアーがデータを読み込み続ける場合は，1 秒ごとなど，定期的にhsdpcliconv コマンドを実行します。

コマンドの実行例を次に示します。

```
hsdpcliconv -d conv_sample.cfg /var/hsdpcli/data
```

hsdpcliconv コマンドの詳細については，「16.1　hsdpcliconv」を参照してください。

出力データを削除する

出力バイナリファイル（hsdpcliconv コマンドで変換されたファイルを含まない）を削除するには，hsdpclirm コマンドを実行します。ディスクがいっぱいにならないよう，1 時間に 1 回など，定期的に hsdpclirm コマンドを実行します。

コマンドの実行例を次に示します。

```
hsdpclirm -s 10G /var/hsdpcli/data
```

hsdpclirm コマンドの詳細については，「16.2　hsdpclirm」を参照してください。

11.6 HSDP クライアントライブラリインタフェース

Hitachi Streaming Data Platform software development kit は，HSDP クライアントプログラムの実装に必要な関数，構造体，型，および定数を提供します。次の表に提供する関数，構造体，型，および定数を示します。使用するには，クライアントプログラムにhsdpcli.h ヘッダーを含めます。

関数	説明
hsdp_cli_init ()	HSDP クライアントライブラリを初期化します。
hsdp_cli_init_thd ()	HSDP クライアントのスレッドを初期化します。
hsdp_cli_reg_dst ()	データを送信する宛先（TCP データ入力コネクター）を登録します。
hsdp_cli_get_addr ()	データ出力のためのメモリアドレスを返します。
hsdp_cli_send ()	TCP データ入力コネクターにデータを送信します。
hsdp_cli_term_thd ()	HSDP クライアントのスレッドを終了します。
hsdp_cli_term ()	HSDP クライアントライブラリを終了します。

構造体	説明
HSDPCLI_COM_PARM_TBL	HSDP クライアントライブラリの共通パラメーターです。
HSDPCLI_SEND_PARM_TBL	データ送信機能のパラメーターです。
HSDPCLI_FILE_PARM_TBL	ファイル出力機能のパラメーターです。
HSDPCLI_ERR_INFO_TBL	エラー情報です。
HSDPCLI_ADDR_INFO_TBL	アドレス情報です。
HSDPCLI_HASHKEY_TBL	ハッシュキーです。

型	説明
HSDPSHORT	short
HSDPUSHORT	unsigned short
HSDPINT	int
HSDPUINT	unsigned int
HSDPLONG	long
HSDPULONG	unsigned long
HSDPCLI_THD_ID	void *

定数	説明
HSDP_OK	成功を示す戻り値です。

定数	説明
HSDP_NG	失敗を示す戻り値です。
HSDPCLI_NO_FLG	拡張フラグはありません。
HSDPCLI_ON	関数を有効にします。
HSDPCLI_OFF	関数を無効にします。
HSDPCLI_OSDEFAULT	OS のデフォルト設定を関数に適用します。
HSDPCLI_NO_MONITOR	監視設定で，監視しません。
HSDPCLI_SEND_ROUND	負荷分散方式にラウンドロビンを適用します。
HSDPCLI_SEND_HASH	負荷分散方式にハッシングを適用します。

11.6.1　関数

(1)　hsdp_cli_init ()

形式

```
HSDPINT hsdp_cli_init (
    HSDPCLI_COM_PARM_TBL *init_com_parm_pt,
    HSDPCLI_SEND_PARM_TBL *init_send_parm_pt,
    HSDPCLI_FILE_PARM_TBL *init_file_parm_pt,
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，HSDP クライアントライブラリを初期化します。HSDP クライアントの初期化処理の開始時に，この関数を呼び出します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
init_com_parm_pt	共通パラメーターが格納されている場所を指定します。	HSDPCLI_COM_PARM_TBL 詳細については，「11.6.2　構造体」の「(1) HSDPCLI_COM_PARM_TBL」を参照してください。	入力

パラメーター	説明	値の範囲	入力／出力※
init_send_parm_pt	データ送信機能のパラメーターが格納されている場所を指定します。	HSDPCLI_SEND_PARM_TBL 詳細については，「11.6.2 構造体」の「(2) HSDPCLI_SEND_PARM_TBL」を参照してください。	入力
init_file_parm_pt	ファイル出力機能のパラメーターが格納されている場所を指定します。	HSDPCLI_SEND_PARM_TBL 詳細については，「11.6.2 構造体」の「(3) HSDPCLI_FILE_PARM_TBL」を参照してください。	入力
err_pt	エラー情報の格納に使用するポインター変数を指定します。 **err_pt* のdst_id メンバーは，0 に設定されます。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2 構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLG を指定します。	HSDPCLI_NO_FLG を指定します	入力

注※
入力／出力は，入力パラメーター，または出力パラメーターを示します。

- 入力
  変数の値を入力できますが，変更できません。
- 出力
  インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。

戻り値

値	意味
HSDP_OK	HSDP クライアントライブラリが正常に初期化されました。
HSDP_NG	初期化中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_SEQUENCE	関数が呼び出されたシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_INVALID_ARGUMENT	指定した引数が無効です。	適切な引数を指定します。
HSDPCLIER_OUT_OF_MEMORY	必要なメモリの割り当てに失敗しました。	次のアクションを実行します。 割り当てるメモリの量を確認し，必要に応じて修正します。処理スレッドの最大数（HSDPCLI_COM_PARM_TBL の thd_max_num）と宛先の最大数

エラー内容	説明	対処方法
HSDPCLIER_OUT_OF_MEMORY	必要なメモリの割り当てに失敗しました。	(HSDPCLI_SEND_PARM_TBL の dst_max_num) に，現在の指定値よりも小さい値を指定します。
HSDPCLIER_LOG_INIT	ログの初期化に失敗しました。	指定したログの出力ディレクトリが存在し，正しい権限が設定されていることを確認します。
HSDPCLIER_MAKE_DIRECTORY	ディレクトリの作成に失敗しました。	指定したパスがディレクトリであり，正しい権限が設定されていることを確認します。
HSDPCLIER_SYSTEM_FAULT	初期化に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル (HSDPCLI_ERR_INFO_TBL) のエラー内容 (err_content) とエラーの原因 (err_cause) を指します。

注意事項

- HSDP クライアントライブラリの終了 (hsdp_cli_term ()の呼び出し) 後に，hsdp_cli_init ()を呼び出さないでください。

(2) hsdp_cli_init_thd ()

形式

```
HSDPINT hsdp_cli_init_thd (
    HSDPCLI_THD_ID *thd_id_pt,
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，HSDP クライアントの処理スレッドを初期化します。スレッドの初期化処理の開始時に，この関数を呼び出します。

前提条件

HSDP クライアントライブラリを初期化する必要があります。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
thd_id_pt	スレッド ID のアドレスの格納に使用するポインター変数を指定します。	－	出力
err_pt	エラー情報の格納に使用するポインター変数を指定します。 **err_pt* の dst_id メンバーは，0 に設定されます。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2 構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLG を指定します。	HSDPCLI_NO_FLG	入力

（凡例）
－：該当しません。
注※
入力／出力は，入力パラメーター，または出力パラメーターを示します。

- 入力
  変数の値を入力できますが，変更できません。
- 出力
  インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。

戻り値

値	意味
HSDP_OK	HSDP クライアントの処理スレッドが正常に初期化されました。
HSDP_NG	初期化中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_SEQUENCE	関数が呼び出されたシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_INVALID_ARGUMENT	1 つ以上の指定した引数が無効です。	正しい引数を指定します。
HSDPCLIER_OVER_THREAD_MAX	処理スレッド数が，処理スレッドの最大数を超えています。	次のアクションを実行します。 処理スレッドの最大数に，現在の指定値よりも大きい値を指定します（HSDPCLI_COM_PARM_TBL の thd_max_num）。 処理スレッド数を減らします。
HSDPCLIER_OUT_OF_MEMORY	必要なメモリの割り当てに失敗しました。	次のアクションを実行します。

エラー内容	説明	対処方法
HSDPCLIER_OUT_OF_MEMORY	必要なメモリの割り当てに失敗しました。	割り当てるメモリの量を確認し，必要に応じて修正します。処理スレッドの最大数（HSDPCLI_COM_PARM_TBL の thd_max_num）と宛先の最大数（HSDPCLI_SEND_PARM_TBL の dst_max_num）に，現在の指定値よりも小さい値を指定します。
HSDPCLIER_FILE_OPEN	ファイルのオープンに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます）。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • open(2)
HSDPCLIER_FILE_TRUNCATE	ファイルの拡張に失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます）。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • write(2)
HSDPCLIER_FILE_MMAP	ファイルのメモリへのマッピングに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます）。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • mmap(2)
HSDPCLIER_FILE_CLOSE	ファイルのクローズに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます）。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • close(2)
HSDPCLIER_FILE_MADVISE	マッピングエリアの操作に失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます）。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • madvise(2)
HSDPCLIER_FILE_MSGGET	メッセージキューの作成に失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します

エラー内容	説明	対処方法
HSDPCLIER_FILE_MSGGET	メッセージキューの作成に失敗しました。	(errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • msgget(2)
HSDPCLIER_FILE_THREAD_CREATE	ファイル管理スレッドの生成に失敗しました。	次のアクションを実行します。 メモリ量を確認し，必要に応じて修正します。カーネルのスレッド数に関するシステム全体の制限値を確認し，必要に応じて修正します(/proc/sys/kernel/threads-max)。
HSDPCLIER_SYSTEM_FAULT	初期化に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル（HSDPCLI_ERR_INFO_TBL）のエラー内容（err_content）とエラーの原因（err_cause）を指します。

注意事項

- HSDP クライアントのスレッド終了（hsdp_cli_term_thd ()の呼び出し）後に，hsdp_cli_init_thd () を呼び出さないでください。

(3) hsdp_cli_reg_dst ()

形式

```
HSDPINT hsdp_cli_reg_dst (
    HSDPCLI_THD_ID    thd_id,
    HSDPCLI_ADDR_INFO_TBL *addr_inf_src_pt,
    HSDPCLI_ADDR_INFO_TBL *addr_inf_dst_pt,
    HSDPINT *dst_id_pt,
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，指定した宛先（TCP データ入力コネクター）を登録し，指定したソースを宛先に接続します。

前提条件

HSDP クライアントの処理スレッドを初期化する必要があります。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
thd_id	hsdp_cli_init_thd()によって返されるスレッド ID を指定します。	－	入力
addr_inf_src_pt	ソースアドレス情報が格納されている場所を指定します。addr_inf_src_pt パラメーターのip_addr メンバーに0 を指定すると，IP アドレスが自動的に割り当てられます。同様に，addr_inf_src_pt パラメーターの port_num メンバーに0 を指定すると，ポート番号が自動的に割り当てられます。	HSDPCLI_ADDR_INFO_TBL 詳細については，「11.6.2　構造体」の「(5) HSDPCLI_ADDR_INFO_TBL」を参照してください。	入力および出力
addr_inf_dst_pt	宛先のアドレス情報が格納されている場所を指定します。	HSDPCLI_ADDR_INFO_TBL 詳細については，「11.6.2　構造体」の「(5) HSDPCLI_ADDR_INFO_TBL」を参照してください。	入力
dst_id_pt	宛先 ID の格納に使用するポインター変数を指定します。宛先 ID は，登録した宛先を管理するため，HSDP クライアントによって使用されます。	－	出力
err_pt	エラー情報の格納に使用するポインター変数を指定します。**err_pt* の dst_id メンバーは，0 に設定されます。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2　構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLG を指定します。	HSDPCLI_NO_FLG	入力

（凡例）

－：該当しません。

注※

入力／出力は，入力パラメーター，出力パラメーター，またはその両方を示します。

- 入力
  変数の値を入力できますが，変更できません。
- 出力
  インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。

戻り値

値	意味
HSDP_OK	宛先が正常に登録されました。
HSDP_NG	登録中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_SEQUENCE	関数が呼び出されたシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_INVALID_ARGUMENT	1 つ以上の指定した引数が無効です。	正しい引数を指定します。
HSDPCLIER_OVER_DST_MAX	宛先数が，宛先の最大数を超えています。	次のアクションを実行します。 宛先の最大数に，現在の指定値よりも大きい値を指定します(HSDPCLI_SEND_PARM_TBL の dst_max_num)。 宛先の数を減らします。
HSDPCLIER_SEND_OPEN_FAIL	ソケットのオープンに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • socket(2) • fcntl(2)
HSDPCLIER_SEND_BIND_FAIL	アドレスのバインドに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • bind(2)
HSDPCLIER_SEND_CON_FAIL	接続の確立に失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します（errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • connect(2) • select(2) • getsockname(2)

エラー内容	説明	対処方法
HSDPCLIER_SEND_CON_TIMEOUT	接続確立要求がタイムアウトしました。	次のアクションを実行します： HSDP クライアントと TCP データ入力コネクター間の TCP/IP 通信が利用できることを確認します。 指定した IP アドレスとポート番号が正しいことを確認します。
HSDPCLIER_SYSTEM_FAULT	登録に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル（HSDPCLI_ERR_INFO_TBL）のエラー内容（err_content）とエラーの原因（err_cause）を指します。

注意事項

なし

(4) hsdp_cli_get_addr ()

形式

```
HSDPINT hsdp_cli_get_addr (
    HSDPCLI_THD_ID thd_id,
    void **return_addr_pt,
    HSDPINT data_sz,
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，データ出力のためのメモリアドレスを返します。このメモリアドレスは，データの割り当てに使用されます。

前提条件

HSDP クライアントの処理スレッドを初期化する必要があります。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
thd_id	hsdp_cli_init_thd()によって返されるスレッドIDを指定します。	hsdp_cli_init_thd()によって返されるスレッドID	入力
return_addr_pt	出力データのメモリアドレスを格納するための，ポインター変数を指定します。	－	出力
data_sz	出力データのサイズを指定します。	8～1,400の整数（バイト）	入力
err_pt	エラー情報の格納に使用するポインター変数を指定します。 **err_pt* のdst_idメンバーは，0に設定されます。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2 構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLGを指定します。	HSDPCLI_NO_FLG	入力

（凡例）
－：該当しません。
注※
入力／出力は，入力パラメーター，または出力パラメーターを示します。

- 入力
  変数の値を入力できますが，変更できません。
- 出力
  インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。

戻り値

値	意味
HSDP_OK	メモリアドレスが正常に取得されました。
HSDP_NG	メモリアドレスの取得中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_ SEQUENCE	関数が呼び出されたシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_INVALID_ ARGUMENT	指定した引数が無効です。	適切な引数を指定します。
HSDPCLIER_FILE_OPEN	ファイルのオープンに失敗しました。	HSDPCLI_ERR_INFO_TBLのerr_causeに従って，エラーの原因を解決します（errnoはシステムコールによって設定さ

エラー内容	説明	対処方法
HSDPCLIER_FILE_OPEN	ファイルのオープンに失敗しました。	れます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • open(2)
HSDPCLIER_FILE_ TRUNCATE	ファイルの拡張に失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します(errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • write(2)
HSDPCLIER_FILE_ MMAP	ファイルのメモリへのマッピングに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します(errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • mmap(2)
HSDPCLIER_FILE_ CLOSE	ファイルのクローズに失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します(errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • close(2)
HSDPCLIER_FILE_ MADVISE	マッピングエリアの操作に失敗しました。	HSDPCLI_ERR_INFO_TBL のerr_cause に従って，エラーの原因を解決します(errno はシステムコールによって設定されます)。システムコール名はメッセージログに出力されます。 原因となるシステムコール： • madvise(2)
HSDPCLIER_SYSTEM_ FAULT	取得に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル（HSDPCLI_ERR_INFO_TBL）のエラー内容（err_content）とエラーの原因（err_cause）を指します。

注意事項

なし

(5) hsdp_cli_send ()

形式

```
HSDPINT hsdp_cli_send (
    HSDPCLI_THD_ID thd_id,
    void *data_pt,
    HSDPINT data_sz,
    HSDPCLI_HASHKEY_TBL *hashkey,
    HSDPINT hashkey_num,
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，TCP データ入力コネクターにデータを送信します。hsdp_cli_init()で指定した負荷分散方式を使用してデータを送信します。

前提条件

HSDP クライアントの処理スレッドを初期化する必要があります。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
thd_id	hsdp_cli_init_thd()によって返されるスレッド ID を指定します。	hsdp_cli_init_thd()によって返されるスレッド ID	入力
data_pt	送信するデータが格納されている場所を指定します。マニュアル『Hitachi Streaming Data Platform 概説』に記載されているとおり，送信するデータは，TCP データの形式である必要があります。 hsdp_cli_get_addr()によって返されたアドレスを指定します。	既存メモリのアドレス	入力
data_sz	送信するデータのサイズを指定します。	1～1,400 の整数（バイト）	入力

パラメーター	説明	値の範囲	入力／出力※
hashkey	ハッシュキー配列の最初のアドレスを指定します。 負荷分散方式がラウンドロビンの場合は，NULL を指定します。	HSDPCLI_HASHKEY_TBL 詳細については，「11.6.2　構造体」の「(6) HSDPCLI_HASHKEY_TBL」を参照してください。	入力
hashkey_num	ハッシュキー番号を指定します。 負荷分散方式がラウンドロビンの場合は，0 を指定します。	1～16 の整数	入力
err_pt	エラー情報の格納に使用するポインター変数を指定します。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2　構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLG を指定します。	HSDPCLI_NO_FLG	入力

注※
入力／出力は，入力パラメーター，または出力パラメーターその両方を示します。

- 入力
  変数の値を入力できますが，変更できません。
- 出力
  インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。

戻り値

値	意味
HSDP_OK	データが HSDP に正常に送信されました。
HSDP_NG	データ送信中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_SEQUENCE	関数が呼び出されたシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_INVALID_ARGUMENT	1 つ以上の指定した引数が無効です。	正しい引数を指定します。
HSDPCLIER_SEND_UNABLE	データを送信できません。 ラウンドロビンの場合：すべての接続が切断されました。 ハッシングの場合：データ送信先への接続が切断されました。	次のアクションを実行します。 ラウンドロビンの場合：このエラーの前に発生したデータ送信エラーの原因を取り除きます。 ハッシングの場合：このエラーの前に発生したデータ送信エラーの原因を取り除

エラー内容	説明	対処方法
HSDPCLIER_SEND_UNABLE	データを送信できません。 ラウンドロビンの場合：すべての接続が切断されました。 ハッシングの場合：データ送信先への接続が切断されました。	きます。問題を解決できない場合は，データ送信を停止してください。
HSDPCLIER_SEND_FAIL	データの送信に失敗しました。	エラーの原因を解決する方法については，HSDPCLI_ERR_INFO_TBL の err_cause を参照してください（errno はシステムコールによって設定されます）。システムコール名はメッセージログに出力されます。 原因となるシステムコール： send(2)
HSDPCLIER_SEND_BUSY	接続がビジー状態のためデータを送信できません。送信バッファ内のデータが破棄されます。	SDP サーバと，HSDP クライアントおよび TCP データ入力コネクター間のネットワークのステータスを確認し，ビジー状態の接続の原因を取り除きます。
HSDPCLIER_SYSTEM_FAULT	送信に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル（HSDPCLI_ERR_INFO_TBL）のエラー内容（err_content）とエラーの原因（err_cause）のことを指します。

注意事項

なし

(6) hsdp_cli_term_thd ()

形式

```
HSDPINT hsdp_cli_term_thd (
    HSDPCLI_THD_ID thd_id,
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，HSDP クライアントのスレッドを終了します。スレッドの終了処理中に，この関数を 1 回呼び出します。

前提条件

HSDP クライアントの処理スレッドを初期化する必要があります。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
thd_id	hsdp_cli_init_thd()によって返されるスレッド ID を指定します。	hsdp_cli_init_thd()によって返されるスレッド ID	入力
err_pt	エラー情報の格納に使用するポインター変数を指定します。エラーがデータ送信機能に関連しない場合，**err_pt* の dst_id メンバーは0 に設定されます。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2 構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLG を指定します。	HSDPCLI_NO_FLG	入力

注※
入力／出力は，入力パラメーター，または出力パラメーターを示します。

- 入力
  変数の値を入力できますが，変更できません。
- 出力
  インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。

戻り値

値	意味
HSDP_OK	HSDP クライアントのスレッドが正常に終了されました。
HSDP_NG	終了中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_SEQUENCE	関数が呼び出されたシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_INVALID_ARGUMENT	1 つ以上の指定した引数が無効です。	正しい引数を指定します。
HSDPCLIER_SEND_DCON_FAIL	接続の解放に失敗しました。一部のデータが送信されなかったおそれがあります。	なし
HSDPCLIER_SEND_DCON_TIMEOUT	接続解放要求がタイムアウトしました。一部のデータが送信されなかったおそれがあります。	なし

エラー内容	説明	対処方法
HSDPCLIER_SYSTEM_FAULT	終了に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル（HSDPCLI_ERR_INFO_TBL）のエラー内容（err_content）とエラーの原因（err_cause）を指します。

注意事項

HSDP クライアントのスレッドの初期化後にエラーが発生した場合は，hsdp_cli_term_thd ()を呼び出してください。これをしないと，OS のリソースが解放されないことがあります。

(7) hsdp_cli_term ()

形式

```
HSDPINT hsdp_cli_term (
    HSDPCLI_ERR_INFO_TBL *err_pt,
    HSDPINT ex_flg
);
```

説明

この関数は，HSDP クライアントライブラリを終了します。HSDP クライアントの終了処理中に，この関数を 1 回呼び出します。

前提条件

HSDP クライアントの処理スレッドを終了する必要があります。

パラメーター

パラメーター	説明	値の範囲	入力／出力※
err_pt	エラー情報を格納するときに使用するポインター変数を指定します。*err_pt* のdst_id メンバーは，0 に設定されます。	HSDPCLI_ERR_INFO_TBL 詳細については，「11.6.2 構造体」の「(4) HSDPCLI_ERR_INFO_TBL」を参照してください。	出力
ex_flg	拡張フラグHSDPCLI_ NO_FLG を指定します。	HSDPCLI_NO_FLG	入力
注※			

パラメーター	説明	値の範囲	入力／出力※
入力／出力は，入力パラメーター，または出力パラメーターを示します。 • 入力 変数の値を入力できますが，変更できません。 • 出力 インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます。			

戻り値

値	意味
HSDP_OK	HSDP クライアントライブラリが正常に終了されました。
HSDP_NG	終了中にエラーが発生しました。

エラー

エラー内容	説明	対処方法
HSDPCLIER_INVALID_SEQUENCE	関数を呼び出したシーケンスが無効です。	関数を正しいシーケンスで呼び出します。
HSDPCLIER_SYSTEM_FAULT	終了に失敗しました。	システム管理者に連絡し，ログファイルを送信します。

メモ

「エラー」とは，エラー発生時にこの関数によって返された，エラー情報テーブル（HSDPCLI_ERR_INFO_TBL）のエラー内容（err_content）とエラーの原因（err_cause）のことを指します。

注意事項

- HSDP の初期化後にエラーが発生した場合は，この関数を呼び出す必要があります。

11.6.2 構造体

(1) HSDPCLI_COM_PARM_TBL

名前

HSDPCLI_COM_PARM_TBL

説明

HSDP クライアントライブラリの共通パラメーターです。

メンバー

メンバー	型	説明	値の範囲
thd_max_num	HSDPINT	HSDP クライアントのスレッドの最大数（hsdp_cli_init_thd()が呼び出された回数と同じ）	1～64 の整数
ts_offset	HSDPINT	レコードのタイムスタンプのオフセット	次のように，タイムスタンプのサイズによって値の範囲が異なります。 • タイムスタンプサイズが 8 の場合： 0～1,392 の整数（バイト） • タイムスタンプサイズが 12 の場合： 0～1,388 の整数（バイト）
ts_sz	HSDPINT	レコードのタイムスタンプのサイズ	8 または 12（バイト）
log_msg_out_dir	char*	メッセージログファイルが出力される絶対ディレクトリパスです。指定したパスの後ろに空文字を追加します。指定したディレクトリが見つからない場合は，ディレクトリが作成されます。	空文字を含む，1～512 バイトの長さの文字列
log_msg_file_max_cnt	HSDPINT	メッセージログファイルの最大数	2～16 の整数
log_msg_file_max_sz	HSDPINT	メッセージログファイルの最大サイズ	4,096～2,147,483,647 の整数（バイト）
log_trc_out_dir	char*	トレースログファイルが出力される絶対ディレクトリパスです。指定したパスの後ろに空文字を追加します。指定したディレクトリが見つからない場合は，ディレクトリが作成されます。	空文字を含む，1～512 バイトの長さの文字列
log_trc_file_max_cnt	HSDPINT	トレースログファイルの最大数	2～16 の整数
log_trc_file_max_sz	HSDPINT	トレースログファイルの最大サイズ	4,096～2,147,483,647 の整数（バイト）

(2) HSDPCLI_SEND_PARM_TBL

名前

HSDPCLI_SEND_PARM_TBL

説明

この構造体には，データ送信機能のパラメーターが含まれています。

メンバー

メンバー	型	説明	値の範囲
load_dspt_mtd	HSDPINT	負荷分散方式を指定します（ラウンドロビンまたはハッシング）。	次の値のうちの 1 つ。 • HSDPCLI_SEND_ROUND ラウンドロビンを使用します。 • HSDPCLI_SEND_HASH ハッシングを使用します。
dst_max_num	HSDPINT	宛先の最大数を指定します。	1～512 の整数
tcp_no_delay	HSDPINT	Nagle アルゴリズムを使用するかどうかを指定します。より高い性能を実現するため，Nagle アルゴリズムは使用しないでください。	次の値のうちの 1 つ。 • HSDPCLI_ON 遅延を有効にします（Nagle アルゴリズムを使用しません）。 • HSDPCLI_OFF 遅延を無効にします（Nagle アルゴリズムを使用します）。 • HSDPCLI_OSDEFAULT OS のデフォルト設定を適用します。
keep_alive	HSDPINT	キープアライブ機能を使用するかどうかを指定します。	次の値のうちの 1 つ。 • HSDPCLI_ON キープアライブを有効にします。 • HSDPCLI_OFF キープアライブを無効にします。 • HSDPCLI_OSDEFAULT OS のデフォルト設定を適用します。
send_buf_sz	HSDPINT	指定した値がwmem-max※の値より大きい場合は，wmem-max の値が使用されます。	1,024～1,073,741,823 の整数（バイト）または次の値。 • HSDPCLI_OSDEFAULT OS のデフォルト設定を適用します。

メンバー	型	説明	値の範囲
send_buf_tm	HSDPINT	送信バッファのオーバーフローによって生じる，データを送信するために掛かる待ち時間を監視します。	0～900,000 の整数（ミリ秒）または次の値。 • HSDPCLI_NO_MONITOR 監視しません。
con_est_tm	HSDPINT	接続を確立するために掛かる時間を監視します。	1～60,000 の整数（ミリ秒）または次の値。 • HSDPCLI_NO_MONITOR 監視しません。
con_rel_tm	HSDPINT	接続を解放するために掛かる時間を監視します。	1～60,000 の整数（ミリ秒）または次の値。 • HSDPCLI_NO_MONITOR 監視しません。
注※ /proc/sys/net/core/wmem_max で指定した値です。			

(3) HSDPCLI_FILE_PARM_TBL

名前

HSDPCLI_FILE_PARM_TBL

説明

この構造体には，ファイル出力機能のパラメーターが含まれています。

メンバー

メンバー	型	説明	値の範囲
out_dir	char*	バイナリファイルが出力される絶対ディレクトリパスです。指定したパスの後ろに空文字を追加します。指定したディレクトリが見つからない場合は，ディレクトリが作成されます。	空文字を含む，1～512 バイトの長さの文字列
bin_file_max_cnt	HSDPINT	バイナリファイルの最大数を指定します。	2～16 の整数
bin_file_max_sz	HSDPINT	バイナリファイルの最大サイズを指定します。	10,485,760～1,073,741,824 の整数（バイト）

(4) HSDPCLI_ERR_INFO_TBL

名前

HSDPCLI_ERR_INFO_TBL

説明

この構造体には，エラー情報が含まれています。この表のメンバーの値は，エラーの発生時に各関数によって設定されます。

メンバー

メンバー	型	説明	値の範囲
err_content	HSDPINT	エラー。	－
err_cause	HSDPINT	エラーの原因。errno の値はシステムコールによって設定されます。errno が0 の場合，原因はありません。	－
dst_id	HSDPINT	宛先 ID は1 から開始します。	－

（凡例）
－：該当しません。

(5) HSDPCLI_ADDR_INFO_TBL

名前

HSDPCLI_ADDR_INFO_TBL

説明

この構造体には，アドレス情報が含まれています。

メンバー

メンバー	型	説明	値の範囲
ip_addr	HSDPUINT	IPv4 アドレス（ホストバイトオーダー）。例えば，0x7F000001 は 127.0.0.1 を表します。	0x00000000～0xFFFFFFFF の整数
port_num	HSDPUSHORT	ポート番号（ホストバイトオーダー）	0～65,535 の整数

(6) HSDPCLI_HASHKEY_TBL

名前

HSDPCLI_HASHKEY_TBL

説明

この構造体には，ハッシュキーが含まれています。

メンバー

メンバー	型	説明	値の範囲
keydata	void*	ハッシュキーが格納されている場所	既存のメモリアドレス
keydata_sz	HSDPINT	ハッシュキーのサイズ	1～1,400 の整数（バイト）

11.6.3 変換定義ファイル

説明

このファイルは，hsdpcliconv コマンドによるファイルフォーマット変換のためのパラメーターを定義します。

ファイル名

任意のファイル名を指定できます。

ファイルの格納先

任意のディレクトリ

形式

パラメーターはそれぞれ別の行で指定します。次の形式に従います。

キー= 値

コメントは，別の行にシャープ記号（#）の後ろに記述します。

キー	値	プリセット値
output_format	変換ファイルフォーマットを指定します。 次のフォーマットを指定します。	csv

キー	値	プリセット値
output_format	• csv	csv
output_file_name※1	変換したファイルの絶対パスを指定します。 ASCII 文字を1～512 の範囲で指定します。次の変換指定変数を使用できます。タイムスタンプは GMT で表します。 • %Y データソースのタイムスタンプの年を表す 4 桁 • %m データソースのタイムスタンプの月を表す 2 桁（01～12） • %d データソースのタイムスタンプの日を表す 2 桁（01～31） • %H データソースのタイムスタンプの時を表す 2 桁（00～23） • %M データソースのタイムスタンプの分を表す 2 桁（00～59） • %S データソースのタイムスタンプの秒を表す 2 桁（00～59） • %t バイナリファイルを作成したスレッドのスレッド ID（TID）	なし。
output_range	1 つの変換ファイルに出力されるデータの時間範囲を指定します。 1～86,400（秒）の整数を指定します※2。	1
record_format	バイナリファイルのレコードのフォーマットを指定します。 1～128 の文字を指定します。各データフィールドに，型とサイズ（バイト）をコロン（:）で区切って指定します。次のように，データフィールドをコンマ（,）で区切ります。 *型*: *サイズ*,*型*: *サイズ*, ... 次の型を指定できます。 • int • string • timestamp 変換したタイムスタンプ（GMT）のフォーマットは，*YYYY-mm-ddHH:MM:SS.ffffffff* となります。 指定した型によって，次のサイズをバイト単位で指定できます。 • int：1，2，4 または8 • string：1～1400 • timestamp：8 または12 8 を指定すると，データはミリ秒として解析されます。12 を指定すると，最初の8 バイトはミリ秒として解析され，次の4 バイトはナノ秒として解析されます。	なし。
record_timestamp	レコードのタイムスタンプのタイムスタンプ番号 レコードに2 つ以上のタイムスタンプがある場合，レコードの始めからのタイムスタンプ番号でレコードのタイムスタンプを指定します。例えば，レ	1

キー	値	プリセット値
record_timestamp	コードに5 つのタイムスタンプがあり，4 番目のタイムスタンプがレコードのタイムスタンプである場合は，4 を指定します。	1

注※1

右の列に示す変換指定変数を使用してファイル名を指定してください。これによって，変換されたファイルが，同じファイル名を使用したあとのデータで上書きされてしまい，データが正しく変換されなくなることを防ぎます。

注※2

大量のデータを扱う場合は，output_range に大きな値を指定しないでください。大きな値を指定すると，変換されたファイルのサイズが，100TB などの非常に大きな値となります。

例

```
# Conversion definition
# output_format=csv
output_file_name=/var/hsdpcli/conv/%Y%m%d/%H%M%S_%t.csv
# ourput_range=1
record_format=string:32,timestamp:12,string:64,string:64,int:2,int:2,int:2
# record_timestamp=1
```

11.7　HSDP クライアントライブラリのトラブルシューティング

11.7.1　エラー発生時のトラブルシューティング手順

操作手順

1. メッセージが出力されたかどうかを確認します。
   - C 関数でエラーが発生した場合：

     HSDP クライアントライブラリが提供する C 関数でエラーが発生している場合，関数はエラー情報を返し，メッセージログファイルにエラーメッセージが出力されます。
   - コマンドでエラーが発生した場合：

     コマンドでエラーが発生している場合，標準エラー出力にエラーメッセージが出力されます。
2. エラーの症状を確認し，エラーの原因を取り除きます。

   返されたエラー情報または出力されたメッセージを確認します。「15.4　関数の詳細」またはエラーメッセージに記載されている手順に従って，エラーの原因を取り除きます。必要な手順を実行しても問題が解消しない場合は，手順 3 に進みます。
3. ログを収集します。

   手順 2 を実行しても問題が解消しない場合は，「11.7.5　トラブルシュートのために収集されるファイルと情報」を参照して，エラーの原因究明に必要なログを収集します。

11.7.2　性能が予想より低い場合のトラブルシューティング手順

HSDP クライアントライブラリを使用するプログラムのファイル出力またはデータ送信のスループットが予想よりも低い場合には，次の手順を実行してください。

11.7.3　データ送信機能の性能が低い場合

操作手順

1. ネットワークインタフェースカード（NIC）のリンク速度をチェックします。

   HSDP クライアントホストの NIC のリンク速度をチェックして，NIC が最大限の性能を発揮しているかどうかを確認してください。`ethtool` コマンドを実行すると，リンク速度をチェックできます。

   `ethtool` コマンドを実行するには，次のパッケージのインストールと root 権限が必要です。
   - `ethtool`

コマンド実行例（リンク速度が 10 Gbps の場合）：

```
$ ethtool eth0
Settings for eth0:
        Supported ports: [ TP ]
        Supported link modes:   1000baseT/Full
                                10000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: No
        Advertised link modes:  Not reported
        Advertised pause frame use: No
        Advertised auto-negotiation: No
        Speed: 10000Mb/s
        Duplex: Full
        Port: Twisted Pair
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: off
        MDI-X: Unknown
        Supports Wake-on: uag
        Wake-on: d
        Link detected: yes
```

ifconfig コマンドを実行すると，ネットワーク装置名をチェックできます。

コマンド実行例：

```
$ ifconfig -a
eth0      Link encap:Ethernet  HWaddr 00:50:56:A3:47:0F
          inet addr:10.0.0.1  Bcast:172.17.127.255  Mask:255.255.128.0
          inet6 addr: fec0::44d:250:56ff:fea3:470f/64 Scope:Site
          inet6 addr: fd00::44d:250:56ff:fea3:470f/64 Scope:Global
          inet6 addr: fe80::250:56ff:fea3:470f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:208178120 errors:0 dropped:0 overruns:0 frame:0
          TX packets:22993317 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:61752988843 (57.5 GiB)  TX bytes:22588326909 (21.0 GiB)
```

上記の例で，IP アドレスが10.0.0.1 の装置の装置名はeth0 です。

データ送信のスループットがリンク速度に近い場合，性能低下の原因はハードウェア関連の性能制限です。この場合には，HSDP クライアントホストの NIC をより高速なものに交換してください。

また，次の点をチェックしてください。

- 接続先の NIC の速度が HSDP クライアントホストの NIC と同等以上であることを確認してください。
- HSDP クライアントによるデータ送信が行われるネットワーク（ハブ，ルータ，ネットワークケーブルを含む）をチェックして，このネットワークが HSDP クライアントホストの NIC のリンク速度に対応していることを確認してください。
- HSDP クライアントと接続先の間のネットワークに障害が発生していないことを確認してください。
- HSDP クライアントと接続先の間のネットワーク負荷が過度に高くなっていないことを確認してください。

2. CPU 利用率をチェックします。

手順 1.を実行しても原因がわからない場合は，プログラムの実行中にtop コマンドを実行して，CPU 利用率をチェックしてください。

コマンド実行例：

```
$ top
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
25116 user1     20   0  587m  24m 5732 S 102.1  1.2   0:10.50 hsdp_client
24980 user2     20   0  646m 1408  528 S 47.6  0.1   0:40.16 server
  :
  :
```

上記の例で，%CPU の値は CPU 利用率を示しています。HSDP クライアントの CPU 利用率が「*HSDP クライアントに割り当てられた CPU 番号* x 100%」または「*HSDP クライアントの処理スレッド数* x 100%」に近い場合，性能低下の原因は CPU 性能のボトルネックである可能性が高いです。CPU 利用率の高い HSDP クライアント処理があるかどうかをチェックしてください。

もし該当する処理がある場合，例えばHSDP クライアントの処理を修正するなどの対処をしてください。

3. バッファ内のデータの量をチェックします。

手順 2.を実行しても原因が分からない場合は，HSDP クライアントの稼働中にnetstat コマンドを実行して，OS の送信バッファに蓄積されたデータの量をチェックしてください。

コマンド実行例：

```
$ netstat -t -c
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address
State
tcp         0 85192 10.10.10.10:50000           10.10.10.11:60000
ESTABLISHED
tcp         0   356 10.10.10.10:50001           10.10.10.11:60001
ESTABLISHED
 :
 :
```

上記の例で，Send-Q の値は，OS の送信バッファに蓄積されたデータの量（バイト数）を示しています。Send-Q の値が増えている場合，またはこの値が高い状態（ライブラリ初期化中に指定された送信バッファサイズの約 2 倍）を維持している場合には，HSDP クライアントホストが過度のビジー状態になっていてデータを送信できなくなっている可能性が高いです。この場合には，次の対処をしてください。

- HSDP クライアントホストの送信バッファサイズを増やす

  ライブラリ初期化中に指定された送信バッファサイズ（HSDPCLI_SEND_PARM_TBL のsend_buf_sz）と HSDP クライアントホスト上の OS の送信バッファサイズ（/proc/sys/net/core/wmem_max）を増やしてください。

- 接続先の受信バッファサイズを増やす

  接続先の OS の受信バッファサイズ（/proc/sys/net/core/rmem_max）を増やしてください。

- Nagle アルゴリズムの使用状況をチェックし，必要に応じて見直す
  Nagle アルゴリズムを使用すると，HSDP クライアントホストが過度のビジー状態になってデータを送信できなくなることがあるので，Nagle アルゴリズムの ON/OFF の設定（HSDPCLI_SEND_PARM_TBL のtcp_no_delay）を変更して HSDP クライアントを再起動してください。
- 接続先をチェックする
  接続先で処理の遅延が発生していないことを確認してください。

11.7.4　ファイル出力機能の性能が低い場合

操作手順

1. プログラムの実行中にiostat コマンドを実行し，ディスクと CPU の使用状況を確認します。
   iostat コマンドを実行するには，次のパッケージのインストールが必要です。
   - sysstat

   コマンド実行例：

```
$ iostat -mxt 1
dd/mm/YY HH:MM:SS (実行日時が示されます)
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           0.00    0.00    3.07   33.74    0.00   63.19

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s avgrq-sz
avgqu-sz   await  svctm  %util
sda               0.00 10187.00   14.00   88.00     0.05    39.64   797.02
   18.48  196.42   9.79  99.90
dm-0              0.00     0.00   13.00 10275.00     0.05    40.14     8.00
  2199.05  229.96   0.10  99.90
dm-1              0.00     0.00    0.00    0.00     0.00     0.00     0.00
    0.00    0.00   0.00   0.00
dm-2              0.00     0.00    0.00    0.00     0.00     0.00     0.00
    0.00    0.00   0.00   0.00
```

   上記の例で，%util の値が 100%に近い状態を保っている場合には，ディスクがビジー状態であり，ディスク I/O 性能のボトルネックが原因で性能が低下している可能性が高いです。このような場合には，ディスクをより高速なものに交換してください。

なお，CPU の利用率は，avg-cpu の隣に出力されます。%util が 100%に近い値ではなくても，CPU 利用率 (%user と%system の合計値) が 100%に近い場合には，CPU 利用率の高い HSDP クライアント処理があるかどうかを確認してください。もし該当する処理がある場合は，例えば HSDP クライアントの処理を修正するなどの対処をしてください。

11.7.5　トラブルシュートのために収集されるファイルと情報

問題を解決できない場合は，次のファイルと情報を収集します。

ファイルと情報	C 関数でエラーが発生した場合	コマンドでエラーが発生した場合	性能が予想より低い場合
標準出力および標準エラー出力※1	×	○	×
メッセージログファイル	○	○	○
トレースログファイル	○	×	○
関数の引数※1	○	×	○
コマンドの引数※1	×	○	×
変換定義ファイル	×	○※2	×
関数によって返されたエラー情報 1	○	×	×
関数の戻り値※1	×	○	×
core ダンプ	○	○※3	×
hsdp_cli_init()に指定されたパラメーター	×	×	○
OS のステータス	×	×	○
OS の統計情報※1	×	×	○

（凡例）
○：このファイルまたは情報を収集してください。
×：このファイルまたは情報の収集は必要ありません。
注※1
HSDP クライアントを起動する前に，この情報を収集できるようにしてください。
注※2
エラーがhsdpcliconv コマンドの実行時に発生した場合にだけ，このファイルを収集します。
注※3
出力された場合にだけ，この情報を収集します。

標準出力および標準エラー出力

標準出力および標準エラー出力に出力されたメッセージを収集します。

メッセージログファイル

メッセージログの出力ディレクトリの配下にあるすべてのファイルを収集します。メッセージログの出力ディレクトリは，ライブラリの初期化時（hsdp_cli_init()の呼び出し時）に，HSDPCLI_COM_PARM_TBL のlog_msg_out_dir で指定されます。

トレースログファイル

トレースログの出力ディレクトリの配下で，問題発生時のタイムスロットに対応するファイルを収集します。トレースログの出力ディレクトリは，ライブラリの初期化時（hsdp_cli_init()の呼び出し時）に，HSDPCLI_COM_PARM_TBL のlog_trc_out_dir で指定されます。

関数の引数

エラーが発生した関数または性能が低い関数の引数を収集します。

コマンドの引数

エラーが発生したコマンドの引数を収集します。

変換定義ファイル

hsdpcliconv コマンドの実行中にエラーが発生した場合，コマンドを実行したときに指定した変換定義ファイルを取得します。

関数によって返されたエラー情報

エラーが発生した関数によって返されたエラー情報を収集します。

関数の戻り値

エラーが発生した関数によって返された戻り値を収集します。

core ダンプ

アクセス違反などによって出力された場合にだけ，core ダンプを収集します。また，HSDP クライアントの実行可能ファイルと，HSDP クライアントによって使用される共有ライブラリを収集します。

core ダンプを出力するには，ulimit -c unlimited を実行し，コアファイルの最大サイズを増やす必要があります（core ダンプは，デフォルトでは出力されません）。

hsdp_cli_init()に指定されたパラメーター

ライブラリの初期化時（hsdp_cli_init()の呼び出し時）に，HSDPCLI_COM_PARM_TBL，HSDPCLI_SEND_PARM_TBL，およびHSDPCLI_FILE_PARM_TBL で指定された値を収集します。

OS のステータス

OS のステータスについては，次の情報を収集します。

- 「11.7.3　データ送信機能の性能が低い場合」の「ネットワークインタフェースカード（NIC）のリンク速度をチェックします」で実行された，ethtool コマンドの実行結果。
- netstat -s の実行結果。

OS の統計情報

OS の統計情報については，次の情報を収集します。

- 60 秒以上の期間，1 秒間隔で（HSDP クライアントの実行中に）実行されたtop コマンドの実行結果

  top コマンドの実行例については，「11.7.3　データ送信機能の性能が低い場合」の「CPU 利用率をチェックします」を参照してください。

- 60 秒以上の期間，1 秒間隔で（HSDP クライアントの実行中に）実行されたnetstat -t-c の実行結果

  netstat コマンドの実行例については，「11.7.3　データ送信機能の性能が低い場合」の「バッファ内のデータの量をチェックします」を参照してください。

- sar -A の実行結果
- 60 秒以上の期間，1 秒間隔で（HSDP クライアントの実行中に）実行されたiostat コマンドの実行結果

iostat コマンドの実行例については，「11.7.4　ファイル出力機能の性能が低い場合」を参照してください。

11.7.6　メッセージログファイル

説明

項目	説明
ファイルサイズ	ファイルサイズは，ライブラリの初期化時（hsdp_cli_init()の呼び出し時）に，HSDPCLI_COM_PARM_TBL のlog_msg_file_max_sz で指定されます。
出力ファイルの切り替え時期	ファイルの切り替えは，現在のメッセージログファイルと，出力されるメッセージの合計サイズが，指定された値を超える場合に発生します。
切り替え時のファイルサイズ	ファイルの切り替えが発生すると，新規のメッセージログファイルのサイズは 0 になります（新規のメッセージログファイルの既存の行は削除されます）。
ファイル切り替え時の運用	ファイルの切り替えが発生すると，新規のメッセージログファイルの既存の行は削除され，先頭行から，新規ファイルに新しいメッセージが出力されます。
処理再開時の対象出力ファイル	新しいメッセージは，最も新しいタイムスタンプのメッセージログファイルに出力（追加）されます。

ファイル名

hsdpcli*N*.log

> **メモ**
>
> *N* は，メッセージログファイルのシリアル番号です（*N* の値の範囲は，1 からメッセージログファイルの最大数です）。メッセージログファイルの最大数は，ライブラリの初期化時（hsdp_cli_init()の呼び出し時）に，HSDPCLI_COM_PARM_TBL のlog_msg_file_max_sz で指定されます。

ファイルの格納先

メッセージログファイルは，ライブラリの初期化時（hsdp_cli_init()の呼び出し時）に，HSDPCLI_COM_PARM_TBL のlog_msg_out_dir で指定されたディレクトリに格納されます。

形式

メッセージログファイルは，次の形式で出力されます。

```
番号 日付 時刻 アプリケーション名 プロセスID スレッドID メッセージID　メッセージ
CRLF
```

項目	詳細
番号	メッセージに割り当てられたシリアル番号（0000～9999）
日付	メッセージの収集日（*yyyy/mm/dd*）
時刻	ローカルタイムでの，メッセージの収集時刻（*hh:mm:ss.sss*）
アプリケーション名	HSDPCLILIB
プロセス ID	メッセージを出力するプロセスのプロセス ID
スレッド ID	メッセージを出力するスレッドのスレッド ID
メッセージ ID	メッセージ ID（KFHD80100～KFHD80199）
メッセージ	メッセージ本文
CRLF	改行コード

11.8 HSDP クライアントプログラムのサンプルプログラム

以下に，HSDP クライアントのサンプルの C プログラム（ラウンドロビンの場合）を示します。

```
/*
 * All Rights Reserved.Copyright (C) 2014, Hitachi, Ltd.
 */

/*
 * include files
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <endian.h>
#include <unistd.h>
#include <pthread.h>

#include "hsdpcli.h"

/*
 * macro definition
 */
/* Library common parameters */
#define THD_MAX_NUM     64
#define TS_OFFSET       32
#define TS_SZ           12
#define LOG_MSG_DIR     "/var/hsdpcli/logs"
#define LOG_MSG_CNT     4
#define LOG_MSG_SZ      1048576
#define LOG_TRC_DIR     "/var/hsdpcli/trc"
#define LOG_TRC_CNT     4
#define LOG_TRC_SZ      1048576

/* Data sender function parameters */
#define LOAD_DSPT_MTD   HSDPCLI_SEND_ROUND
#define DST_MAX_NUM     512
#define TCP_NO_DELAY    HSDPCLI_OFF
#define KEEP_ALIVE      HSDPCLI_OFF
#define SEND_BUF_SZ     HSDPCLI_OSDEFAULT
#define SEND_BUF_TM     HSDPCLI_NO_MONITOR
#define CON_EST_TM      HSDPCLI_NO_MONITOR
#define CON_REL_TM      HSDPCLI_NO_MONITOR

/* File output function parameters */
#define FILE_OUT_DIR    "/var/hsdpcli/out"
#define FILE_MAX_CNT    10
#define FILE_MAX_SZ     10485760

/* Processing parameters */
#define PROC_THD_NUM    1
#define PROC_THDDST_NUM 2
#define SEND_ERR_MAX    30

/* Generating data parameters */
```

```
#define DATA_NUM        10
#define DATA_SZ         178

/* Generating data structure */
typedef struct hsdp_data {
    char data_id[32];
    uint64_t msec;
    unsigned int nsec;
    char src_ipaddr[64];
    char dst_ipaddr[64];
    unsigned short src_port;
    unsigned short dst_port;
    unsigned short value;
    char reserve[2];
} HSDP_DATA;

/* Sources and destinations */
typedef struct addr_infos {
    HSDPCLI_ADDR_INFO_TBL src_addr_info;
    HSDPCLI_ADDR_INFO_TBL dst_addr_info;
} ADDR_INFOS;

/*
 * prototype definition
 */
void* proc_thread(void* arg);

/*
 * main function
 */
int main(int argc, char *argv[]) {
    /* variable definition */
    HSDPINT func_rc;
    int i;

    char log_msg_dir[512];
    char log_trc_dir[512];
    char file_out_dir[512];

    pthread_t thd[PROC_THD_NUM];
    HSDPCLI_COM_PARM_TBL com_parm;
    HSDPCLI_SEND_PARM_TBL send_parm;
    HSDPCLI_FILE_PARM_TBL file_parm;
    HSDPCLI_ERR_INFO_TBL err_info;

    ADDR_INFOS addr_info_parm[PROC_THD_NUM][PROC_THDDST_NUM];

    /* set parameters for initialization */
    com_parm.thd_max_num = THD_MAX_NUM;
    com_parm.ts_offset = TS_OFFSET;
    com_parm.ts_sz = TS_SZ;
    strcpy(log_msg_dir, LOG_MSG_DIR);
    com_parm.log_msg_out_dir = log_msg_dir;
    com_parm.log_msg_file_max_cnt = LOG_MSG_CNT;
    com_parm.log_msg_file_max_sz = LOG_MSG_SZ;
    strcpy(log_trc_dir, LOG_TRC_DIR);
    com_parm.log_trc_out_dir = log_trc_dir;
    com_parm.log_trc_file_max_cnt = LOG_TRC_CNT;
```

```
com_parm.log_trc_file_max_sz = LOG_TRC_SZ;

send_parm.load_dspt_mtd = LOAD_DSPT_MTD;
send_parm.dst_max_num = DST_MAX_NUM;
send_parm.tcp_no_delay = TCP_NO_DELAY;
send_parm.keep_alive = KEEP_ALIVE;
send_parm.send_buf_sz = SEND_BUF_SZ;
send_parm.send_buf_tm = SEND_BUF_TM;
send_parm.con_est_tm = CON_EST_TM;
send_parm.con_rel_tm = CON_REL_TM;

strcpy(file_out_dir, FILE_OUT_DIR);
file_parm.out_dir = file_out_dir;
file_parm.bin_file_max_cnt = FILE_MAX_CNT;
file_parm.bin_file_max_sz = FILE_MAX_SZ;

/* initialize library */
func_rc = hsdp_cli_init(&com_parm, &send_parm, &file_parm,
        &err_info, HSDPCLI_NO_FLG);
if (func_rc != HSDP_OK) {
    printf("[main]initialization error.(err=%d, cause=%d, dst_id=%d)¥n",
            err_info.err_content, err_info.err_cause, err_info.dst_id);
    exit(-1);
}

/* set address information */
addr_info_parm[0][0].src_addr_info.ip_addr = 0;
addr_info_parm[0][0].src_addr_info.port_num = 0;
addr_info_parm[0][0].dst_addr_info.ip_addr = 0x7F000001;
addr_info_parm[0][0].dst_addr_info.port_num = 20000;

addr_info_parm[0][1].src_addr_info.ip_addr = 0;
addr_info_parm[0][1].src_addr_info.port_num = 0;
addr_info_parm[0][1].dst_addr_info.ip_addr = 0x7F000001;
addr_info_parm[0][1].dst_addr_info.port_num = 20001;

/* create processing thread */
for (i = 0; i < PROC_THD_NUM; i++) {
    func_rc = pthread_create(&thd[i], NULL,
            proc_thread, (void*)&addr_info_parm[i]);
    if (func_rc != 0) {
        printf("[main]thread creation error.¥n");
        exit(-1);
    }
}

/* wait thread process */
for (i = 0; i < PROC_THD_NUM; i++) {
    pthread_join(thd[i], NULL);
}

/* terminate library */
func_rc = hsdp_cli_term(&err_info, HSDPCLI_NO_FLG);
if (func_rc != HSDP_OK) {
    printf("[main]termination error.(err=%d, cause=%d, dst_id=%d)¥n",
            err_info.err_content, err_info.err_cause, err_info.dst_id);
}
```

```
    return (0);
}

/*
 * processing thread
 */
void* proc_thread(void* arg) {
    /* variable definition */
    HSDPINT func_rc;
    int i;

    HSDPCLI_THD_ID thd_id;
    HSDPCLI_ERR_INFO_TBL err_info;
    ADDR_INFOS* addr_info_parms;
    int dst_id[PROC_THDDST_NUM];

    char* data_buf;
    HSDP_DATA data;

    int send_err_cnt;

    /* initialize thread */
    func_rc = hsdp_cli_init_thd(&thd_id, &err_info, HSDPCLI_NO_FLG);
    if (func_rc != HSDP_OK) {
        printf("[processing thread]thread initialization error."
                "(err=%d, cause=%d, dst_id=%d)¥n",
                err_info.err_content, err_info.err_cause, err_info.dst_id);
        exit(-1);
    }

    /* register destination */
    addr_info_parms = (ADDR_INFOS*)arg;
    for (i = 0; i < PROC_THDDST_NUM; i++) {
        func_rc = hsdp_cli_reg_dst(thd_id,
                &(addr_info_parms[i].src_addr_info),
                &(addr_info_parms[i].dst_addr_info),
                &dst_id[i], &err_info, HSDPCLI_NO_FLG);
        if (func_rc != HSDP_OK) {
            printf("[processing thread]destination registration error."
                    "(err=%d, cause=%d, dst_id=%d)¥n",
                err_info.err_content, err_info.err_cause, err_info.dst_id);

            /* terminate thread */
            func_rc = hsdp_cli_term_thd(thd_id, &err_info, HSDPCLI_NO_FLG);
            if (func_rc != HSDP_OK) {
                printf("[processing thread]thread termination error."
                        "(err=%d, cause=%d, dst_id=%d)¥n",
                err_info.err_content, err_info.err_cause, err_info.dst_id);
            }
            exit(-1);
        }
    }

    for (i = 0; i < DATA_NUM; i++) {
        /* generate data */
        sprintf(data.data_id, "data_id%d", i);
        data.msec = htobe64((uint64_t)i);
        data.nsec = htobe32(0);
```

```
strcpy(data.src_ipaddr, "10.0.0.1");
strcpy(data.dst_ipaddr, "20.0.0.1");
data.src_port = htobe16(30000 + i);
data.dst_port = htobe16(40000 + i);
data.value = htobe16(100 + i);

/* get address for file output */
func_rc = hsdp_cli_get_addr(thd_id, (void**)&data_buf, DATA_SZ,
        &err_info, HSDPCLI_NO_FLG);
if (func_rc != HSDP_OK) {
    printf("[processing thread]get address error."
            "(err=%d, cause=%d, dst_id=%d)¥n",
        err_info.err_content, err_info.err_cause, err_info.dst_id);

    /* terminate thread */
    func_rc = hsdp_cli_term_thd(thd_id, &err_info, HSDPCLI_NO_FLG);
    if (func_rc != HSDP_OK) {
        printf("[processing thread]thread termination error."
                "(err=%d, cause=%d, dst_id=%d)¥n",
        err_info.err_content, err_info.err_cause, err_info.dst_id);
    }
    exit(-1);
}

/* write data */
memcpy(data_buf, &data, DATA_SZ);
printf("[processing thread]data:(%s,%ld,%d,%s,%s,%d,%d,%d)¥n",
        data.data_id,
        be64toh(data.msec),
        be32toh(data.nsec),
        data.src_ipaddr,
        data.dst_ipaddr,
        be16toh(data.src_port),
        be16toh(data.dst_port),
        be16toh(data.value));

/* send data */
send_err_cnt = 0;
do {
    func_rc = hsdp_cli_send(thd_id, data_buf, DATA_SZ, NULL, 0,
            &err_info, HSDPCLI_NO_FLG);
    if (func_rc != HSDP_OK) {
        if ((err_info.err_content == HSDPCLIER_SEND_FAIL ||
                err_info.err_content == HSDPCLIER_SEND_BUSY) &&
                send_err_cnt < SEND_ERR_MAX) {
            printf("[processing thread]data send error.Continue."
                    "(err=%d, cause=%d, dst_id=%d)¥n",
                    err_info.err_content, err_info.err_cause,
                    err_info.dst_id);
            send_err_cnt++;
            sleep(1);

        } else {
            printf("[processing thread]data send error.Exit."
                    "(err=%d, cause=%d, dst_id=%d)¥n",
                    err_info.err_content, err_info.err_cause,
                    err_info.dst_id);
```

```
                    /* terminate thread */
                    func_rc = hsdp_cli_term_thd(thd_id, &err_info,
                            HSDPCLI_NO_FLG);
                    if (func_rc != HSDP_OK) {
                        printf("[processing thread]thread termination error."
                                "(err=%d, cause=%d, dst_id=%d)¥n",
                                err_info.err_content, err_info.err_cause,
                                err_info.dst_id);
                    }
                    exit(-1);
                }
            }
        } while (func_rc != HSDP_OK);
    }

    /* terminate thread */
    func_rc = hsdp_cli_term_thd(thd_id, &err_info, HSDPCLI_NO_FLG);
    if (func_rc != HSDP_OK) {
        printf("[processing thread]thread termination error."
                "(err=%d, cause=%d, dst_id=%d)¥n",
                err_info.err_content, err_info.err_cause, err_info.dst_id);
    }

    return (NULL);
```

12 外部定義関数の作成（Java）

外部定義集合関数，外部定義スカラ関数，外部定義ストリーム間演算関数を使用する場合は，CQLでこれらの関数を定義するほかに，外部定義関数の作成が必要です。この章では，外部定義関数の作成方法について説明します。

12.1　外部定義関数の概要（Java）

外部定義関数は，ユーザーが Java の API などを使用して作成する関数です。外部定義関数の処理ロジックを，ユーザーが Java で記述して作成するクラスファイルにメソッドとして実装することで，任意の処理を実行できます。

外部定義集合関数，外部定義スカラ関数，外部定義ストリーム間演算関数を使用する場合は，CQL でこれらの関数を定義するほかに，外部定義関数を作成します。外部定義関数の作成手順を次に示します。

1. 関数の実装

   外部定義関数の処理ロジックを，Java で記述して作成するクラスファイルにメソッドとして実装します。関数の実装方法については，「12.2　外部定義関数の実装（Java）」を参照してください。

2. 定義ファイルでの関数の記述

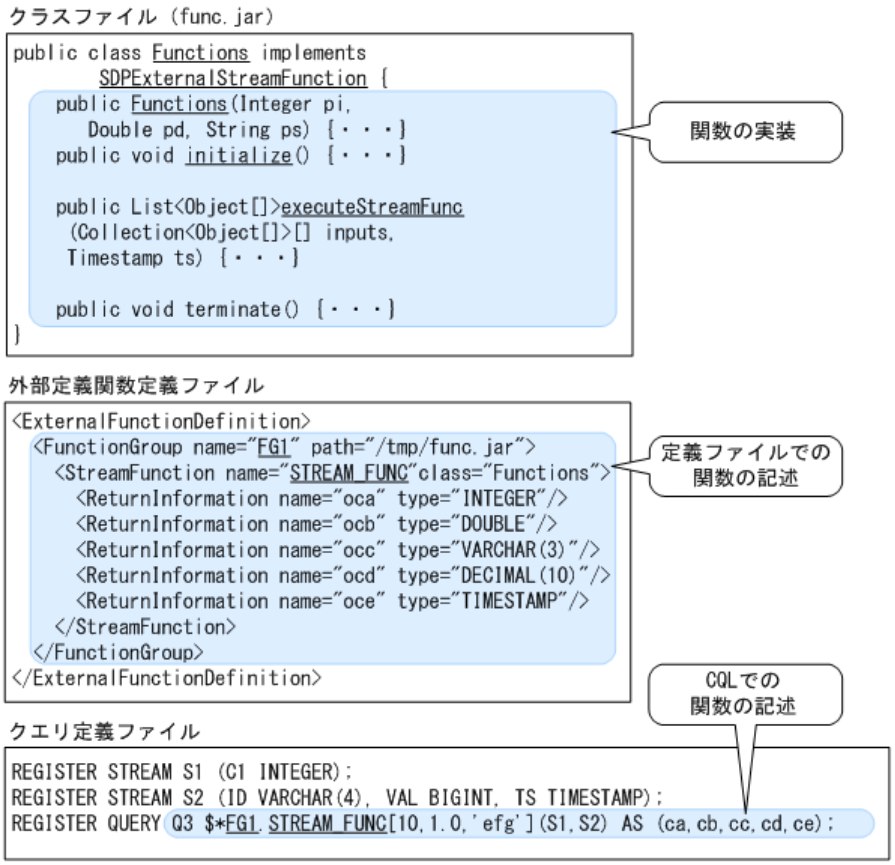
   外部定義関数を実装したクラスとの関連づけを外部定義関数定義ファイルに記述して，SDP サーバに外部定義関数を認識させます。外部定義関数定義ファイルについては，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

3. CQL での関数の記述

   外部定義関数定義ファイルに記述した外部定義関数をクエリ定義ファイルの CQL に記述します。外部定義集合関数の CQL への記述方法については，「4.4.23　外部定義集合関数」を参照してください。外部定義スカラ関数の CQL への記述方法については，「4.4.24　外部定義スカラ関数」を参照してください。また，外部定義ストリーム間演算関数の CQL への記述方法については，「4.4.25　外部定義ストリーム間演算関数」を参照してください。

外部定義関数で使用するファイルの関連を次の図に示します。

図 12-1　外部定義関数で使用するファイルの関連



12.2　外部定義関数の実装（Java）

外部定義関数の実装では，外部定義関数の処理ロジックを，Java で記述して作成するクラスファイルにメソッドとして実装します。ここでは，外部定義関数のクラスとメソッドの実装方法について説明します。また，外部定義関数の変更について説明します。なお，外部定義関数インタフェースの詳細については，「14.　外部定義関数インタフェース（Java）」を参照してください。

12.2.1　外部定義関数のクラスの実装（Java）

外部定義関数のクラスを実装する場合の作成規則，および注意事項を説明します。

作成規則

外部定義関数を実装するクラスファイルは，次の規則に従って作成してください。

使用できるフィールド

外部定義関数を実装するクラスで，static フィールドを使用できます。

static フィールドの値は，同一のクエリグループ内で同じクラスファイルを使用する，すべての外部定義関数の使用個所で共有されます。ただし，クエリグループが異なる場合は，同じクラスファイルを使用してもstatic フィールドの値は個別に管理されます。

パッケージ名，およびクラス名

パッケージ名，およびクラス名には，任意の名前を使用できます。

ただし，jp.co.Hitachi.soft.sdp から始まるパッケージ名は指定できません。

コンストラクターの指定

外部定義関数を実装するクラスは，クエリ内で外部定義関数の使用個所ごとに，CQL の定義内容に一致するコンストラクターを使用して，クエリグループの登録・削除時にインスタンスを作成・破棄します。このため，次の条件を満たすように作成してください。

- クラスは，修飾子に必ずpublic を指定して作成してください。インスタンス化できない抽象クラス（abstract）や内部クラスは使用できません。
- コンストラクターは，修飾子に必ずpublic を指定し，CQL で指定する初期化パラメーターの数およびデータ型が一致するように作成してください。ただし，プリミティブ型を引数とするコンストラクターは使用できません。なお，CQL で初期化パラメーターを省略する場合には，コンストラクターを作成しないでデフォルトコンストラクターを使用するか，または引数を持たないコンストラクターを作成してください。

これらの条件を満たしていない場合，外部定義関数を実装するクラスのインスタンスを生成できないため，クエリグループの登録でエラーになります。

実装するインタフェース（外部定義ストリーム間演算関数）

外部定義ストリーム間演算関数を実装するクラスは，SDPExternalStreamFunction インタフェースを実装してください。SDPExternalStreamFunction インタフェースは，次のメソッドを実装する必要があります。

- executeStreamFunc メソッド

  このメソッドには，外部定義ストリーム間演算関数の処理を実装してください。

- initialize メソッド

  このメソッドには，外部定義ストリーム間演算関数を実装するクラスが持つフィールドの初期設定など，関数実行前の初期化処理を実装してください。

- terminate メソッド

  このメソッドには，外部定義ストリーム間演算関数を実装するクラスが持つフィールドのクリアなど，関数実行後の終了処理を実装してください。

SDPExternalStreamFunction インタフェースの詳細については，「14.4　SDPExternalStreamFunction インタフェース」を参照してください。

実装するインタフェース（外部定義集合関数または外部定義スカラ関数）

外部定義集合関数または外部定義スカラ関数を実装するクラスは，SDPExternalFunction インタフェースを実装してください。SDPExternalFunction インタフェースは，次のメソッドを実装する必要があります。

- refresh メソッド

  このメソッドには，外部定義集合関数または外部定義スカラ関数の実装クラスが持つフィールドの初期化処理を実装してください。

SDPExternalFunction インタフェースの詳細については，「14.3　SDPExternalFunction インタフェース」を参照してください。

注意事項

外部定義関数を実装するクラスファイルの注意事項を次に示します。

- 外部定義関数を実装するクラスファイルのパッケージ名を含むクラス名は，jvm_options.cfg の SDP_CLASS_PATH パラメーターに指定したクラスパスのクラスファイルと重複しないようにしてください。

  重複した場合は，SDP_CLASS_PATH パラメーターに指定したクラスパスのクラスが優先して読み込まれるため，次に示すようなエラーが発生して外部定義関数が正しく動作しなくなるおそれがあります。

  （例）

  - インタフェースの未実装，またはメソッドが存在しないなどの要因で，クエリグループの登録でエラーになる。
  - SDP_CLASS_PATH パラメーターに指定したクラスパスのクラスのメソッドが外部定義関数として実行されてしまう。

- 外部定義関数を実装するクラスの中で，自身のクラスパス以外のクラスパスにあるクラスを参照している場合は，jvm_options.cfg のSDP_CLASS_PATH パラメーターにそのクラスパスを指定してください。

  指定しない場合は，参照個所がフィールドやコンストラクターのときは，インスタンスの生成ができないため，クエリグループの登録でエラーになります。参照個所がメソッドのときは，そのメソッドの実行時に例外java.lang.NoClassDefFoundError が発生してクエリグループが閉塞します。

  特に，次の場合には，クエリグループが閉塞し，クエリグループを削除しないと再開始できなくなるため注意してください。

  - クエリグループの停止・閉塞や，SDPExternalFunction インタフェースのrefresh メソッド，またはSDPExternalStreamFunction インタフェースのterminate メソッドで例外が発生した。
  - クエリグループの開始や，SDPExternalStreamFunction インタフェースのinitialize メソッドで例外が発生した。

  なお，jvm_options.cfg は，SDP サーバの起動後には変更できません。SDP サーバの起動前にjvm_options.cfg を編集してください。

- 外部定義関数を実装するクラスファイルのクラスパスは，jvm_options.cfg のSDP_CLASS_PATH パラメーターには指定しないでください。指定した場合は，次のようになるため注意してください。
  - SDP サーバの起動中に，クラスやメソッドの変更を反映できなくなります。
  - 外部定義関数を実装するクラスのstatic フィールドの値は，すべてのクエリグループで共有されます。

- 外部定義関数を実装するクラスの中で，標準出力および標準エラー出力への出力を実行した場合，SDP サーバを実行している JavaVM プロセスに出力します。また，相対パスでファイル参照などを実行した場合は，運用ディレクトリからの相対パスとして扱います。

- 外部定義関数はクエリ実行中に呼び出されるため，外部定義関数の実行時間が長いほど，ストリームデータ処理システムの処理性能が低下します。また，単位時間当たりに処理できるタプル数が減り，入力ストリームキューでキューあふれが発生する要因にもなります。このため，最長でも 10 ミリ秒程度で外部定義関数の処理が完了するように実装することをお勧めします。

- 外部定義関数で JNI を使用する場合，外部定義関数の実装クラスとは別にネイティブライブラリのロード専用の Java クラスを作成し，そのクラスパスをjvm_options.cfg のSDP_CLASS_PATH パラメーターに指定してください。外部定義関数実装クラス内でネイティブライブラリをロードすると，クエリの登録に失敗するおそれがあります。

12.2.2　外部定義関数のメソッドの実装（Java）

この節では，外部定義関数のメソッドの実装について説明します。なお，外部定義関数インタフェースの詳細については，「14.　外部定義関数インタフェース（Java）」を参照してください。

(1) 外部定義集合関数と外部定義スカラ関数で実装するメソッド

外部定義スカラ関数と外部定義集合関数のメソッドを実装する場合の作成規則，および注意事項を説明します。

共通の作成規則

外部定義スカラ関数と外部定義集合関数で共通のメソッド作成規則について，次に示します。

- 必ずrefresh()メソッドを実装してください。このメソッドには，外部定義関数を実装するクラスが持つフィールドの初期化処理を実装してください。このメソッドは，クエリ内で外部定義関数を使用するフィールドごとに，次の契機で実行します。
  - クエリグループを停止または閉塞したとき
  - クエリグループのすべての入力ストリームにStreamInput#putEnd()メソッドを実行したとき
  - 処理対象のタプル集合からタプルがなくなったとき（RANGE ウィンドウの生存期間を超過した場合など）

SDPExternalFunction インタフェースの詳細は「14.3　SDPExternalFunction インタフェース」を参照してください。

- 外部定義関数に使用するメソッドは必ずpublic を指定して作成してください。public を指定していない場合，メソッドの読み込みに失敗してクエリグループの登録時にエラーとなります。

  次の規則で外部定義関数に使用するメソッド名を作成してください。
  - 半角英数字と下線（_）から成る，1～100 文字の名称（先頭は半角英字）を指定します。
  - CQL の予約語は使用できません。
  - ストリーム名，列名，クエリ名と重複する文字列を使用できます。
- 外部定義関数に使用するメソッドの戻り値のデータ型として，CQL のデータ型に対応する Java のデータ型をすべて使用できます。また，これらのデータ型でプリミティブ型と対応するラッパークラスを持つデータ型（int など）は，プリミティブ型とラッパークラスのどちらも使用できます。それ以外のデータ型，またはvoid を指定した場合はクエリグループの登録時にエラーとなります。
- 同一クラス内に，外部定義関数に使用する複数のメソッドを作成できます。ただし，1 つのクラスから外部定義関数定義ファイルに指定できるメソッドの数は 64 個までです。外部定義関数定義ファイルの詳細は「12.3　外部定義関数の作成例（Java）」を参照してください。

外部定義スカラ関数固有の作成規則

外部定義スカラ関数固有の作成規則を次に示します。

- 外部定義スカラ関数に使用するメソッドの引数は，1 つのObject 配列とします。メソッドの実行時，この引数にはクエリ定義ファイルで引数に指定したデータが指定順に格納された配列が渡されます。

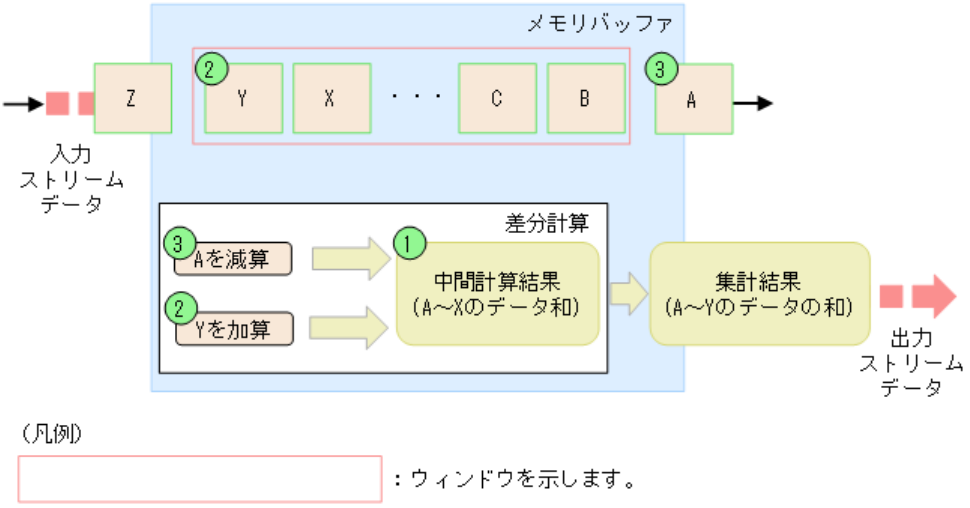
外部定義集合関数固有の処理方式と作成規則

外部定義集合関数固有の処理方式と作成規則を次に示します。

外部定義集合関数を実装するには，メソッドの処理内容を HSDP での集合関数の処理方式に合わせて作成する必要があります。HSDP では，高速データ処理を実行する仕組みとして中間計算結果をメモリ上に保存し，対象の集合にデータが追加，削除された契機でそのデータの差分計算だけをするように実装しています。

例えば，25 個（アルファベットの A～Y）のデータの合計値を求める場合，集合関数SUM は次の処理ロジックで合計値を計算しています。

図 12-2　集合関数SUM の処理ロジック



1. メモリ上に現在の合計値を中間計算結果として保存する領域の値を指定します。この領域の初期値は0です。
2. ウィンドウにデータ（Y）が追加された場合，そのデータ（Y）を中間計算結果に加算した合計値が集計結果として出力されます。
3. ウィンドウからデータ（A）が削除された場合，そのデータ（A）を中間計算結果から減算した合計値が集計結果として出力されます。

外部定義集合関数を実装するメソッドの引数は，次の 2 つです。

- 第 1 引数：Object 配列

  メソッドの実行時，この引数にはクエリ定義ファイルで引数に指定したデータが指定順に格納された配列が渡されます。

  第 2 引数：boolean

  メソッドの実行時，この引数には第 1 引数のデータが集合に追加されたデータか削除されたデータかを示す値が渡されます。

  集合にデータを追加した場合の実行時：true

  集合からデータを削除した場合の実行時：false

- HSDP の集合関数の処理方式で外部定義集合関数を実装するメソッドを作成する場合，次の 3 つを作成してください。
  - 中間計算結果を保存するフィールド
  - 対象の集合にデータが追加された場合の中間計算結果更新，結果返却処理
  - 対象の集合からデータが削除された場合の中間計算結果更新，結果返却処理

  追加時の処理と削除時の処理の振り分けは，第 2 引数の値で実行してください。

共通の注意事項

外部定義スカラ関数と外部定義集合関数で共通の注意事項を次に示します。

- 外部定義関数に使用するメソッドでは，次の処理を実行しないでください。
  - SDP サーバのプロセスを終了する処理（System#exit()メソッド，Runtime#halt()メソッドなど）
  - Thread#currentThread()メソッドで取得した SDP サーバのスレッドに対するメソッドの実行
  - 無限ループなど，メソッドが終了しない処理

  これらの処理を実行する場合，SDP サーバの動作は保証されません。
- 外部定義関数に使用するメソッドでは，エラーが発生した場合に戻り値を返さないで例外をスローしてください。正常な計算結果以外を返してクエリを続行すると，それ以降のクエリ処理で SDP サーバが予期しないエラーを発生する原因となります。

  例外をスローすると，メッセージKFSP42401-E が出力され，クエリグループが閉塞されます。
- 外部定義関数に使用するメソッドでは，戻り値としてnull を返さないでください。null を返した場合，メッセージKFSP42400-E が出力され，クエリグループが閉塞されます。
- クエリ定義ファイルで外部定義関数の引数に指定した値式の数やデータ型が，実装したメソッドの処理内容に対応しているかどうかは SDP サーバではチェックされません。引数のチェック処理が必要な場合は，メソッド内でチェック処理を実装してください。
- 外部定義関数定義ファイルでは，同じグループの中に，名前が重複するメソッドを複数指定できません（大文字小文字を区別しない）。このため，外部定義関数として使用するメソッドを複数用意する場合は，大文字小文字だけが異なる名前も含めて，同じ名前のメソッドを作成しないでください。外部定義関数定義ファイルの詳細は「12.3　外部定義関数の作成例（Java）」を参照してください。

外部定義スカラ関数固有の注意事項

外部定義スカラ関数固有の注意事項を次に示します。

- 外部定義スカラ関数に使用するメソッドは，同じ値の引数を渡した場合に結果を変更しないでください。

```
private int data = 0;
public int method(Object[] arg) {
int a = ((Integer)arg[0]).intValue();
int result = a + data;
data = a;
```

```
return result;  // 引数が同じ値でも前回の引数が違えば結果が異なる
}
```

同じ引数でも条件によって結果を変更する場合，以降のクエリ処理が実行されません。また，SDP サーバが予期しないエラーを発生する原因となります。

外部定義集合関数固有の注意事項

外部定義集合関数固有の注意事項を次に示します。

- クエリがGROUP BY 句などで入力データを複数グループに分けた場合，グループごとに外部定義集合関数の実装クラスのインスタンスを作成して，中間計算結果をグループ別に管理します。そのため，次の 2 つに注意する必要があります。
  - 外部定義集合関数の中間計算結果を管理するフィールドはstatic にしないでください。static にすると中間計算結果を入力データのグループ別に管理できなくなるため，集合関数の結果を正しく出力されません。
  - 入力データのグループ別にインスタンスが作成されるため，クエリ処理の実行中にインスタンスが追加されることがあります。そのため，外部定義集合関数を実装するクラスに処理時間の長いコンストラクターを作成すると処理時間が長くなります。

    外部定義集合関数を実装するクラスにコンストラクターを作成する場合，処理時間が長くならないようにしてください。

(2) 外部定義ストリーム間演算関数で実装するメソッド

外部定義ストリーム間演算関数のメソッドを実装する場合の作成規則，および注意事項を説明します。

作成規則

外部定義関数を実装するメソッドは，次の規則に従って作成してください。

修飾子とメソッド名

メソッドは，修飾子に必ずpublic を指定し，名前は「executeStreamFunc」で作成してください。

メソッドの引数

外部定義ストリーム間演算関数に使用するメソッドの引数は，次の 2 つです。

- 第 1 引数：Collection<Object[]>[]型

  メソッドの実行時，この引数には，クエリ定義ファイルで指定した入力ストリームごとに，同時刻に発生したタプルが格納されたデータオブジェクト配列を渡します。
- 第 2 引数：Timestamp 型

  メソッドの実行時，この引数には，第 1 引数で渡されるタプルの時刻情報を渡します。

  同時刻のタイムスタンプを持つタプルを複数回に分けてメソッドに渡す場合があるため，第 2 引数の時刻情報には，前回のメソッド実行時と同時刻のタイムスタンプが渡されることがあります。

メソッドの戻り値

外部定義関数に使用するメソッドの戻り値は，List<Object[]>型とします。

- メソッドの実行時，出力ストリームのタプルが格納されたデータオブジェクト配列のリストを返してください。出力対象のタプルが存在しない場合は，空のリストを返してください。
- メソッドの戻り値（List<Object[]>型）のObject[]に格納する各要素（タプル）のデータ型は，CQLのデータ型に対応するJavaのデータ型をすべて使用できます。しかし，外部定義関数定義ファイルで指定する出力ストリームのカラムのデータ型（CQLのデータ型）に対応する型で実装してください。

  また，これらのデータ型でプリミティブ型と対応するラッパークラスを持つデータ型（intなど）は，プリミティブ型とラッパークラスのどちらも使用できます。

  それ以外のデータ型を指定した場合は，クエリ実行時にエラーになります。

メソッドの戻り値の検証

外部定義関数を使用するクエリの実行キーは，外部定義関数定義ファイルに指定した戻り値の情報を基に生成されます。

クエリ実行時の外部定義関数の戻り値が，外部定義関数定義ファイルの戻り値の情報と一致していない場合，結果の不正や例外の発生などの意図しない動作が発生するおそれがあります。これらの現象を回避するため，双方の戻り値の情報が一致しているかどうかを検証できます。

戻り値の情報の一致を検証するかどうかは，system_config.propertiesのengine.externalStreamFuncVerifyModeパラメーターで指定します。

このパラメーターにtrueを指定し，検証結果が不正だった場合は，エラーメッセージが出力されて，クエリグループが閉塞されます。この場合は，実装メソッドの処理および外部定義関数定義ファイルを見直してください。

外部定義関数のクラスファイルまたは外部定義関数定義ファイルを修正する場合は，クエリグループの削除と再登録が必要です。外部定義関数の変更，またはクエリグループの変更については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

注意事項

外部定義関数を実装するメソッドの注意事項を次に示します。

- メソッドの実行時に，1つの入力ストリームに対して複数のタプルが入力された場合，そのタプル間の順序は不定となります。メソッドでは，引数で渡されるタプルの順序に依存しない処理を実装する必要があります。
- メソッドでは，次の処理を実行しないでください。
  - SDPサーバのプロセスを終了する処理（System#exit()，Runtime#halt()など）
  - Thread#currentThread()などで取得したSDPサーバのスレッドに対するメソッドの実行
  - 無限ループなど，メソッドが終了しない処理
  - SDPサーバに対して，直接API（put()など）を発行する処理や，運用コマンドを実行する処理

  これらの処理を外部定義関数で実行した場合，SDPサーバの動作は保証されません。

- メソッドでは，新たにスレッドを起動する場合には，スレッドを停止させる処理も実装してください。
- メソッドでは，次に示すような事象の発生でクエリを続行できないと判断する場合には，戻り値を返さないで例外をスローしてください。
  - エラーの発生によって，次回の関数の実行でもエラーが発生する確率が高く，関数が正しい戻り値を返せない場合。
  - 関数内での演算によって NaN が発生したなど，以降の関数の戻り値が NaN になって正しい戻り値を返せない場合。

  例外をスローすると，メッセージKFSP42401-E を出力して，クエリグループを閉塞します。

  なお，今回は一時的な要因で正しい戻り値を返せなくても，次回の関数の実行では正常に動作可能と判断できる場合には，例外をスローしないで空のリストを返すことで，クエリ処理を継続できます。
- メソッドの戻り値，および戻り値の各要素としてnull を返さないでください。null を返した場合，メッセージKFSP42400-E またはKFSP42403-E を出力して，クエリグループを閉塞します。
- メソッドでは，長さを指定できるデータ型を返すときに，定義値を超えるデータを返さないでください。返した場合には，メッセージを出力して，クエリグループを閉塞します。

  ただし，system_config.properties のengine.externalStreamFuncVerifyMode パラメーターにfalse を指定した場合には，外部定義関数が返したデータをクエリ実行結果としてそのまま利用するため，メソッドを実装するときに注意してください。
- クエリ定義ファイルで外部定義関数の引数に指定した入力ストリームの数が，実装したメソッドの処理内容に対応しているかは SDP サーバではチェックしません。引数のチェック処理が必要な場合は，メソッド内でチェック処理を実装してください。

12.2.3 外部定義関数の変更（Java）

実装したメソッドの処理内容が原因で障害が発生した場合などに，クラスやメソッドを変更するときは，クエリグループの削除と再登録が必要です。

外部定義関数定義ファイルに指定した外部定義関数の実装クラスとメソッドは，クエリグループの登録時に読み込まれます。このため，外部定義関数の実装クラスとメソッドの変更は，クエリグループを削除して再登録することで反映されます。再登録をしていないクエリグループには，メソッドの変更は反映されません。

外部定義関数の変更，またはクエリグループの変更については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

12.3　外部定義関数の作成例（Java）

この節では，外部定義関数の作成例について説明します。なお，外部定義関数インタフェースの詳細については，「14.　外部定義関数インタフェース（Java）」を参照してください。

12.3.1　外部定義集合関数と外部定義スカラ関数の作成例

外部定義集合関数または外部定義スカラ関数を使用する場合の，外部定義関数の作成例を次に示します。

外部定義関数の実装クラスファイルの作成例

```
package samples;

import java.util.HashMap;
import jp.co.Hitachi.soft.sdp.plugin.SDPExternalFunction;

public class Functions implements SDPExternalFunction {
//  フィールド（外部定義関数の中間計算結果）
private HashMap<Integer, Integer> data = new HashMap<Integer, Integer>();

//  外部定義スカラ関数用メソッド（INTEGER型の剰余計算）
public int Mod(Object[] args) {
int a = ((Integer)args[0]).intValue();
int b = ((Integer)args[1]).intValue();
return (a % b);
}
//  外部定義集合関数用メソッド（集合から，指定した値と一致する個数を返す）
public int Countif(Object[] args, boolean isInput) {
Integer a = (Integer)args[0]; //集合の追加/削除データ
Integer b = (Integer)args[1]; //一致条件
if(isInput) {
// 集合にデータが追加された場合の処理
// 中間計算結果に，args[0]のデータ個数を1加算
if (data.containsKey(a)) {
data.put(a, new Integer((data.get(a)).intValue()+1));
} else {
data.put(a, new Integer(1));
}
} else {
// 集合からデータが削除された場合の処理
// 中間計算結果から，args[0]のデータ個数を1減算
if (data.containsKey(a)) {
data.put(a, new Integer((data.get(a)).intValue()-1));
}
}
// 現在のargs[1]の値のデータ個数を返却
if(data.containsKey(b)) {
return (data.get(b)).intValue();
} else {
return 0;
}
}
```

```
// 初期化メソッド
public void refresh() {
data.clear(); //フィールドの初期化処理を実装する。
}
}
```

外部定義関数定義ファイルの作成例

```
<?xml version="1.0" encoding="UTF-8"?>
<root:ExternalFunctionDefinition
xmlns:root="http://www.hitachi.co.jp/soft/xml/sdp/function"
xmlns:group="http://www.hitachi.co.jp/soft/xml/sdp/function/functiongroup">
<!-- 外部定義関数のグループ定義  -->
<group:FunctionGroup name="fg1" path="/tmp/func.jar">
<!-- クラス定義 -->
<group:ClassInfo name="samples.Functions">  <!-- クラス名はパッケージから記述 -->
<!-- 関数定義 -->
<group:func name="Mod"/>    <!-- 関数名は実装メソッドと大文字小文字をそろえる -->
<group:func name="Countif"/>
</group:ClassInfo>
</group:FunctionGroup>
</root:ExternalFunctionDefinition>
```

クエリ定義ファイルの作成例

```
REGISTER STREAM S1 (C1 INTEGER);

// 入力ストリームS1から，カラムC1が3で割り切れる値のタプルだけ出力するクエリ
REGISTER QUERY FILTER
ISTREAM ( SELECT * FROM S1[ROWS 10] WHERE $$FG1.MOD(S1.C1, 3) = 0 );

// 入力ストリームS1から，カラムC1が10のタプル個数を出力するクエリ
REGISTER QUERY COUNT10
ISTREAM ( SELECT $#FG1.COUNTIF(S1.C1, 10) FROM S1[ROWS 10]);
```

12.3.2　外部定義ストリーム間演算関数の作成例

外部定義ストリーム間演算関数を使用する場合の，外部定義関数の作成例を次に示します。

外部定義関数のクラスファイルの作成例

```
package samples;

import java.sql.Timestamp;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import jp.co.Hitachi.soft.sdp.plugin.function.stream.SDPExternalStreamFunction;
```

```
public class ExternalStreamFunction implements SDPExternalStreamFunction {
    // 合計値格納マップ
    Map<String, Long> tempMap = new HashMap<String, Long>();
    // 乗数
    int mul;
    // 実行回数
    int loopCount;

    // 外部定義関数の実装クラスのコンストラクター。
    // mul:乗数。クエリに記述した外部定義関数の初期化パラメーターに指定した値。
    public ExternalStreamFunction(Integer mul) {
        this.mul = mul;
        loopCount = 0;
    }

    // 外部定義関数の処理を実装するメソッド。
    // IDごとにVALの合計値を保持し，乗数で乗算した結果を出力します。
    // inputs:タプルの集合。入力ストリームごとにタプルが格納されています。
    // ts    :inputsに格納されているタプルの時刻(タイムスタンプ)。
    public List<Object[]> executeStreamFunc(Collection<Object[]>[] inputs, Timestamp ts) {
        // 戻り値格納リスト
        List<Object[]> res = new ArrayList<Object[]>();
        // 入力ストリーム数分ループ
        for (int i = 0; i < inputs.length; i++) {
            // 入力ストリームを取得します
            Collection<Object[]> values = inputs[i];
            // タプル数分ループ
            for (Object[] value : values) {
                // IDを取得します
                String id = (String) value[0];
                // VALを取得します
                Long val = (Long) value[1];
                // 合計値をIDごとに合計値格納マップに保存します
                if (tempMap.containsKey(id)) {
                    // 既出IDの場合，加算します
                    tempMap.put(id, tempMap.get(id) + val);
                } else {
                    // 新規IDの場合，新規に登録します
                    tempMap.put(id, val);
                }
            }
        }
        // IDの数分ループ
        for (String key : tempMap.keySet()) {
            // 戻り値格納リストに出力タプルを格納します
            res.add(new Object[] { loopCount, key, tempMap.get(key) * mul, ts });
        }
        loopCount++;
        return res;
    }
    // 初期化メソッド
    public void initialize() {
        tempMap.clear();
        loopCount = 0;
    }

    // 終了メソッド
```

```
    public void terminate() {
        // 処理なし
    }
}
```

外部定義関数定義ファイルの作成例

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- All Rights Reserved. Copyright (C) 2013, Hitachi, Ltd. -->
<root:ExternalFunctionDefinition
xmlns:root="http://www.hitachi.co.jp/soft/xml/sdp/function"
xmlns:group="http://www.hitachi.co.jp/soft/xml/sdp/function/functiongroup">
    <!-- 外部定義関数グループ定義 -->
    <group:FunctionGroup name="FG1" path="samples/external/src">
        <!-- 関数定義 -->
        <group:StreamFunction name="FUNC1"
         class="samples.ExternalStreamFunction">
            <!-- 戻り値定義 -->
            <group:ReturnInformation name="R1" type="INT" />
            <group:ReturnInformation name="R2" type="VARCHAR(10)" />
            <group:ReturnInformation name="R3" type="BIGINT" />
            <group:ReturnInformation name="R4" type="TIMESTAMP(9)" />
        </group:StreamFunction>
    </group:FunctionGroup>
</root:ExternalFunctionDefinition>
```

クエリ定義ファイルの作成例

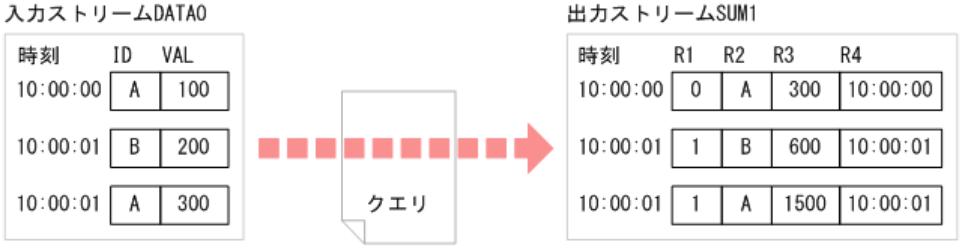
```
REGISTER STREAM DATA0(ID VARCHAR(10), VAL BIGINT);

//入力ストリームDATA0から，カラムIDごとに最新のタプルを保持するクエリ
REGISTER QUERY Q1 SELECT ID, VAL FROM DATA0[PARTITION BY ID ROWS 1];
//最新のタプルを出力するクエリ
REGISTER QUERY Q2 ISTREAM (SELECT * FROM Q1);
//直前のタプルを出力するクエリ
REGISTER QUERY Q3 DSTREAM (SELECT * FROM Q1);
//入力タプルのカラムIDごとにカラムVALの値を合計し，乗数3で乗算した結果を出力するクエリ
REGISTER QUERY SUM1 $*FG1.FUNC1[3](Q2,Q3);
```

入力ストリーム DATA0 と出力ストリーム SUM1 の関係

上記の作成例での入力ストリームDATA0 と出力ストリームSUM1 の関係を，一例として次の図に示します。

図 12-3　入力ストリーム DATA0 と出力ストリーム SUM1 の関係



図中のクエリでは，次の 4 つの列を持つ出力ストリームSUM1 を出力します。

- R1（1 列目）：外部定義関数の呼び出し回数-1
- R2（2 列目）：カラム ID
- R3（3 列目）：カラム ID ごとに VAL の値を合計して 3 で乗算した値
- R4（4 列目）：時刻（タプルのタイムスタンプ）

図に示した一例でのexecuteStreamFunc メソッドの引数を次に示します。

外部定義関数の呼び出し回数	第 1 引数				第 2 引数
	Collection<Object[]>[0] (出力ストリーム Q2)		Collection<Object[]>[1] (出力ストリーム Q3)		Timestamp (タプルのタイムスタンプ)
	Object[0] (ID)	Object[1] (VAL)	Object[0] (ID)	Object[1] (VAL)	
1 回目	A	100	なし	なし	10:00:00
2 回目	B	200	A	100	10:00:01
	A	300	なし	なし	

13 外部定義関数の作成（C）

この章では，C の外部定義関数を作成する方法を説明します。

13.1　外部定義関数の概要（C）

外部定義関数を作成する手順は次のとおりです。

操作手順

1. C で関数を実装します。

   外部定義関数のロジックは，C のソースファイルで C 関数として定義されます。関数の実装方法については，「13.2　外部定義関数の実装（C）」を参照してください。

2. 定義ファイルで関数を定義します。

   SDP に追加される C コードと外部定義関数の関係性は，外部定義関数定義ファイルで定義され，SDP サーバは外部定義関数を認識します。外部定義関数定義ファイルの詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

3. CQL で関数を定義します。

   （外部定義関数定義ファイルで定義した）外部定義関数に対するコールを（クエリ定義ファイルが記述される）CQL で定義します。CQL で外部定義関数を定義する方法については，「3.1　CQL の記述規則」を参照してください。

使用例

次に，外部定義関数に使用されるファイルの関係性を示します。

図 13-1　ソースファイル（libfunc.so にコンパイル，アーカイブされる）

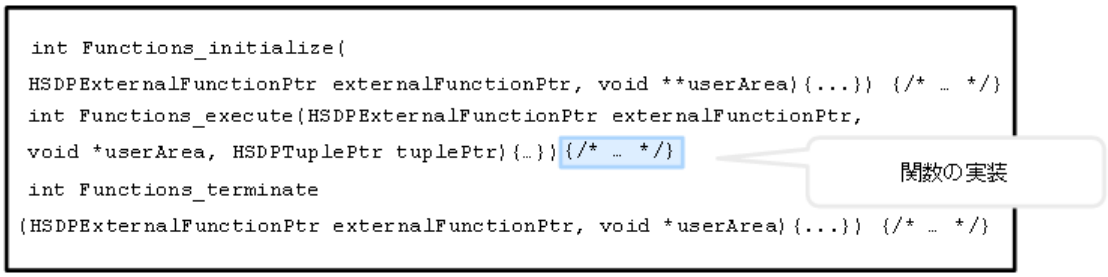
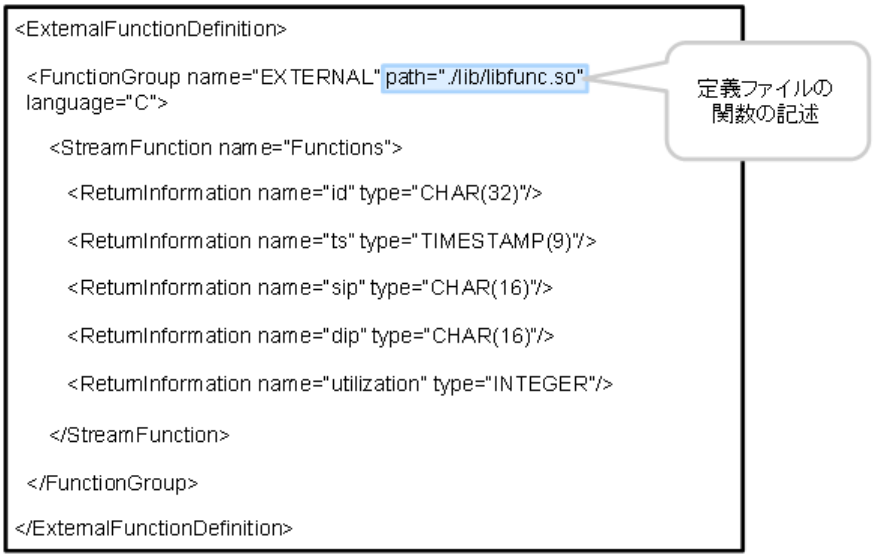
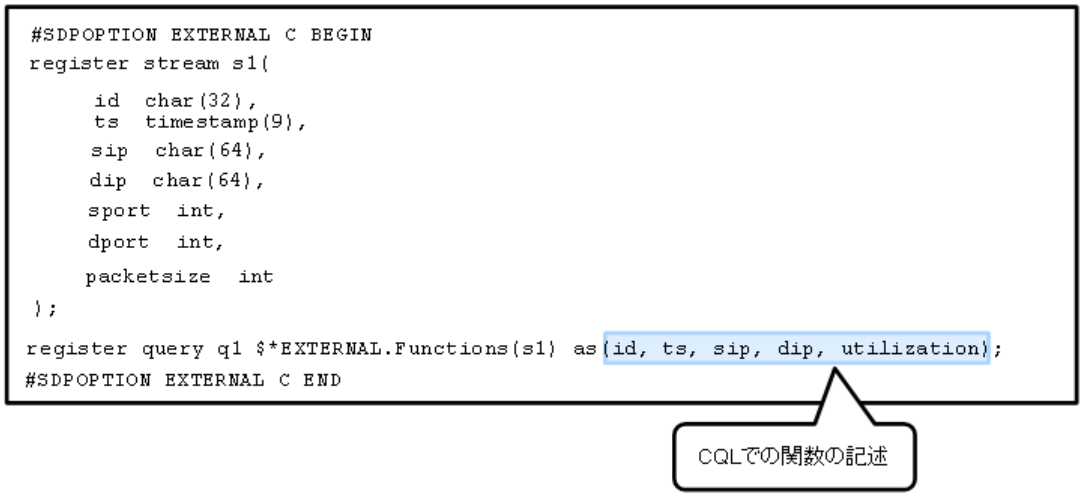


図 13-2　外部定義関数定義ファイル




```
<ExternalFunctionDefinition>
 <FunctionGroup name="EXTERNAL" path="./lib/libfunc.so"
 language="C">
   <StreamFunction name="Functions">
     <ReturnInformation name="id" type="CHAR(32)"/>
     <ReturnInformation name="ts" type="TIMESTAMP(9)"/>
     <ReturnInformation name="sip" type="CHAR(16)"/>
     <ReturnInformation name="dip" type="CHAR(16)"/>
     <ReturnInformation name="utilization" type="INTEGER"/>
   </StreamFunction>
 </FunctionGroup>
</ExternalFunctionDefinition>
```

図 13-3　クエリ定義ファイル




```
#SDPOPTION EXTERNAL C BEGIN
register stream s1(
     id  char(32),
     ts  timestamp(9),
     sip  char(64),
     dip  char(64),
     sport  int,
     dport  int,
     packetsize  int
);
register query q1 $*EXTERNAL.Functions(s1) as(id, ts, sip, dip, utilization);
#SDPOPTION EXTERNAL C END
```

13.1.1　コンパイラーの要件

HSDP クライアントライブラリを使用して C または C++で HSDP クライアントを開発するか，HSDP を使用して C で外部定義関数を開発するには，GCC 4.4.6 以降が必要です。

13.2　外部定義関数の実装（C）

外部定義関数の実装では，外部定義関数の処理ロジックが C で作成されたソースファイルの関数として実装されます。ここでは，C のソースファイルと外部定義関数の実装方法と，外部定義関数に対する変更について説明します。

13.2.1　外部定義関数のための C 関数の実装（C）

外部定義関数のための C 関数を実装する場合の作成規則，および注意事項を説明します。

作成規則

次の規則に従って，外部定義関数を実装するソースファイルを作成します。

- ソースファイル名

  任意の名前を使用できます。

- ヘッダー

  外部定義関数を実装する C ソースファイルを作成する場合は，次のヘッダーを含めます。

  ヘッダー：hsdpedf.h

  場所：/opt/hitachi/hsdp/sdk/lib/include/

  関数を使用する場合は，「15.2　外部定義関数の C の API」を参照してください。

- 実装する C 関数

  外部定義関数を実装する C コードで，次の 3 つの関数を実装する必要があります。*関数名* は，外部定義関数の名前を表します。

  *関数名*_initialize ()関数

  　この関数で，外部定義関数の初期化処理を実装します。

  *関数名*_execute ()関数

  　この関数で，外部定義関数の処理を実装します。

  *関数名*_terminate ()関数

  　この関数で，外部定義関数の終了処理を実装します。

  このインタフェースの詳細については，「14.2　外部定義関数インタフェースの一覧」を参照してください。

- 初期化関数

  外部定義関数を実装する C コードは，外部定義関数の各定義に一致する，クエリ定義ファイルで利用可能な初期化関数を実装し，クエリグループの登録，削除時にインスタンスを作成，破棄する必要があります。したがって，作成時には次の条件を満たす必要があります。

  初期化関数は，CQL で指定した初期化パラメーターの数とデータ型が常に一致するよう作成します。ただし，初期化関数には，引数としてプリミティブ型を設定することはできません。

- 初期化パラメーター

  初期化パラメーターを使用する場合は，データ型に対応する C の API 関数を使用します。提供されている関数についての詳細は，「15.4.38　HSDPExternalFunction_getIntegerInitParameter ()」から「15.4.41　HSDPExternalFunction_getStringInitParameter」を参照してください。

- 外部定義関数のコンパイル

  コンパイラーを使用して外部定義関数のソースファイルをコンパイルし，共有ライブラリを作成します。共有ライブラリの名前は必ず，libライブラリ名.so とします。

  *ライブラリ名*には任意の名前を指定できます。

  コンパイルには，GCC コンパイラーを使用します。

  コンパイル例を次に示します。

```
gcc -c sample.c -I/opt/hitachi/hsdp/sdk/lib/include -fPIC
gcc -shared -o libsample.so sample.o -L/opt/hitachi/hsdp/system/sdp/lib
-lhsdpcfw
```

注意事項

- 外部定義関数を実装する C コードから標準出力および標準エラー出力が出力される場合は，SDP サーバを実行する Java VM プロセスに出力されます。

  ファイルが相対パスで参照される場合は，運用ディレクトリからの相対パスとして扱われます。

- 外部定義関数はクエリの実行中に呼び出されるため，外部定義関数の実行時間が長くなると，ストリームデータ処理システムの処理性能が低下するおそれがあります。また，単位時間当たりに処理できるタプル数が減り，入力ストリームキューでキューあふれが発生する要因にもなります。

- 作成した共有ライブラリの名前が既存ライブラリと同じで，競合する場合は，ExternalDefinitionFunction.xml でライブラリの絶対パスを指定します。ライブラリパスを相対パスとしても指定できますが，運用ディレクトリからの相対パスとなるように指定してください。

13.2.2　外部定義関数メソッドの実装に関する注意事項（C）

外部定義関数を実装する関数についての注意事項を次に示します。

- 関数で次の処理を実行しないでください。次の処理が外部定義関数で実行されると，SDP サーバの動作は保証されません。
  - SDP サーバの処理を完了する処理（exit()など）
  - 関数が終了しない処理（無限ループなど）
  - SDP サーバに対し，直接 API を発行する処理と，運用コマンドを実行する処理
  - 子プロセスを作成する処理（fork()など）
- 関数で新たにスレッドを起動する場合は，スレッドを停止する処理も実装します。

- 次に示す問題が原因でクエリを続行できないと判断される場合は，戻り値としてエラーコードを返します。
  - エラーの発生によって，関数の次の実行時にエラーが発生する可能性が高く，関数が正しい戻り値を返さない場合。
  - 後続の関数の戻り値が NaN のため，正しい戻り値を返せない場合（関数内での演算によって NaN が発生した場合など）。

  エラーコードが返されると，KFSP42401-E メッセージが出力され，クエリグループが保留されます。また，一時的な理由によって正しい戻り値が返されない場合でも，通常の運用によって次の関数を実行できると判断できるときには，戻り値コード0 によってクエリ処理を継続できます。
- クエリ実行時に，外部定義関数定義ファイルの出力ストリームタプルの情報が，外部定義関数の戻り値と一致しない場合は，意図しない操作（不正な結果など）が実行されている可能性があります。

13.2.3 外部定義関数の変更（C）

実装したメソッドの処理内容が原因でエラーが発生した場合にソースファイルを変更するときは，クエリグループの削除と再登録が必要です。

外部定義関数定義ファイルに指定した外部定義関数の実装は，クエリグループの登録時に読み込まれます。そのため，外部定義関数の実装の変更は，クエリグループを削除して再登録することによって反映されます。メソッドの変更は，再登録していないクエリグループには反映されません。

外部定義関数とクエリグループに対する変更の詳細については，マニュアル『Hitachi Streaming Data Platform システム構築ガイド』を参照してください。

13.3 外部定義関数の作成例（C）

```
#include <stdlib.h>
#include <stdio.h>
#include <hsdpedf.h>

#define MALLOC_ERROR        (1)

typedef struct {
    int                     intValue;
    long                    longValue;
    double                  doubleValue;
    char *                  charValue;
}Context, *ContextPtr;


int ExternalFunction_initialize(
        HSDPExternalFunctionPtr         externalFunctionPtr,
        void**                          userArea)
{
    ContextPtr  initializeArea;
    int         retValue;

    /* create user area */
    initializeArea  = malloc(sizeof(Context));
    if(initializeArea == NULL){
        return (MALLOC_ERROR);
    }

    /* get initial value of intValue */
    retValue = HSDPExternalFunction_getIntegerInitParameter(
                    externalFunctionPtr, 0, &(initializeArea->intValue));
    if(retValue != HSDP_OK){
        return (retValue);
    }

    /* get initial value of longValue */
    retValue = HSDPExternalFunction_getLongInitParameter(
                    externalFunctionPtr, 1, &(initializeArea->longValue));
    if(retValue != HSDP_OK){
        return (retValue);
    }

    /* get initial value of doubleValue */
    retValue = HSDPExternalFunction_getDoubleInitParameter(
                    externalFunctionPtr, 2, &(initializeArea->doubleValue));
    if(retValue != HSDP_OK){
        return (retValue);
    }

    /* get initial value of charValue */
    retValue = HSDPExternalFunction_getStringInitParameter(
                    externalFunctionPtr, 3, &(initializeArea->charValue));
    if(retValue != HSDP_OK){
        return (retValue);
    }
```

```
    *userArea = initializeArea;
    return (HSDP_OK);
}

int ExternalFunction_execute(
            HSDPExternalFunctionPtr         externalFunctionPtr,
            void*                           userArea,
            HSDPTuplePtr                    tuplePtr)
{
    ContextPtr          context = (ContextPtr)userArea;
    int                 retValue = HSDP_OK;
    HSDPSchemaPtr       inputSchemaPtr;
    HSDPSchemaPtr       outputSchemaPtr;
    HSDPTuplePtr        tupleArray[1];
    int                 allocedSize;
    int                 addedSize;

    HSDPCharType            charTest;
    HSDPByteType            byteTest;
    HSDPShortType           shortTest;
    HSDPIntType             intTest;
    HSDPLongType            longTest;
    HSDPFloatType           floatTest;
    HSDPDoubleType          doubleTest;
    HSDPVarCharTypePtr      varcharTest;
    HSDPTimestampTypePtr    timeTest;

    /* get input stream schema */
    retValue = HSDPExternalFunction_getInputTupleSchema(
                    externalFunctionPtr, &inputSchemaPtr);
    if(retValue != HSDP_OK){
        goto FUNC_END;
    }

    /* get output stream schema */
    retValue = HSDPExternalFunction_getOutputTupleSchema(
                    externalFunctionPtr, &outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto FUNC_END;
    }

    /* allocate output tuple */
    retValue = HSDPExternalFunction_allocOutputTuple(
                        externalFunctionPtr, tupleArray,
                        1, &allocedSize);
    if(retValue != HSDP_OK){
        goto FUNC_END;
    }


    /* get value from input tuple & set value to output tuple */
    /* Char */
    retValue = HSDPTuple_getChar(tuplePtr, 0, inputSchemaPtr, &charTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_setChar(tupleArray[0], 0, charTest, outputSchemaPtr);
```

```
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* Byte */
    retValue = HSDPTuple_getByte(tuplePtr, 1, inputSchemaPtr, &byteTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_setByte(tupleArray[0], 1, byteTest, outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* Short */
    retValue = HSDPTuple_getShort(tuplePtr, 2, inputSchemaPtr, &shortTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_setShort(tupleArray[0], 2, shortTest, outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* Int */
    retValue = HSDPTuple_getInt(tuplePtr, 3, inputSchemaPtr, &intTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_setInt(tupleArray[0], 3, intTest, outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* Long */
    retValue = HSDPTuple_getLong(tuplePtr, 4, inputSchemaPtr, &longTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_setLong(tupleArray[0], 4, longTest, outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* Float */
    retValue = HSDPTuple_getFloat(tuplePtr, 5, inputSchemaPtr, &floatTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_setFloat(tupleArray[0], 5, floatTest, outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* Double */
    retValue = HSDPTuple_getDouble(tuplePtr, 6, inputSchemaPtr, &doubleTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
```

```
    }
    retValue = HSDPTuple_setDouble(tupleArray[0], 6,
                                   doubleTest, outputSchemaPtr);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* VarChar */
    retValue = HSDPVarCharType_init("", &varcharTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_getVarChar(tuplePtr, 7, inputSchemaPtr, varcharTest);
    if(retValue != HSDP_OK){
        HSDPVarCharType_term(varcharTest);
        goto ERR_END;
    }
    retValue = HSDPTuple_setVarChar(tupleArray[0], 7, varcharTest,
                                    outputSchemaPtr);
    if(retValue != HSDP_OK){
        HSDPVarCharType_term(varcharTest);
        goto ERR_END;
    }
    HSDPVarCharType_term(varcharTest);

    /* Timestamp */
    retValue = HSDPTimestampType_init(0,0, &timeTest);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }
    retValue = HSDPTuple_getTimestamp(tuplePtr, 8, inputSchemaPtr, timeTest);
    if(retValue != HSDP_OK){
        HSDPTimestampType_term(timeTest);
        goto ERR_END;
    }
    retValue = HSDPTuple_setTimestamp(tupleArray[0], 8, timeTest,
                                      outputSchemaPtr);
    if(retValue != HSDP_OK){
        HSDPTimestampType_term(timeTest);
        goto ERR_END;
    }
    HSDPTimestampType_term(timeTest);

    /* output tuple */
    retValue = HSDPExternalFunction_addTuple(
                    externalFunctionPtr, tupleArray, 1, &addedSize);
    if(retValue != HSDP_OK){
        goto ERR_END;
    }

    /* deallocate input tuple */
    HSDPExternalFunction_deallocInputTuple(externalFunctionPtr, &tuplePtr, 1);
    return (HSDP_OK);

ERR_END:
    /* deallocate output tuple */
    HSDPExternalFunction_deallocOutputTuple(externalFunctionPtr, tupleArray, 1);
```

```
FUNC_END:
    /* deallocate input tuple */
    HSDPExternalFunction_deallocInputTuple(externalFunctionPtr, &tuplePtr, 1);
    return (retValue);
}

int ExternalFunction_terminate(
            HSDPExternalFunctionPtr             externalFunctionPtr,
            void*                               userArea)
{
    ContextPtr context = (ContextPtr)userArea;
    free(context->charValue);
    free(context);
    return (HSDP_OK);
}
```

14 外部定義関数インタフェース（Java）

この章では，外部定義関数の作成で使用する外部定義関数インタフェースの文法について説明します。

14.1　外部定義関数インタフェースの記述形式

ここでは，外部定義関数インタフェースの記述形式について説明します。

各インタフェースで説明する項目は次のとおりです。ただし，インタフェースによっては説明しない項目もあります。

形式
　インタフェースの記述形式を示します。

説明
　インタフェースの機能について説明します。

パラメーター
　インタフェースのパラメーターについて説明します。

例外
　インタフェースを利用する際に発生する例外について説明します。

戻り値
　インタフェースの戻り値について説明します。

注意事項
　インタフェースを利用する上での注意事項について説明します。

14.2　外部定義関数インタフェースの一覧

外部定義関数インタフェースの一覧を次に示します。

表 14-1　外部定義関数インタフェースの一覧

項番	パッケージ名	インタフェース名	説明
1	jp.co.Hitachi.soft.sdp.plugin	SDPExternalFunction	外部定義集合関数，および外部定義スカラ関数を実装するクラスが継承するインタフェースです。クラスの初期化処理の実装を提供します。
2	jp.co.Hitachi.soft.sdp.plugin.function.stream	SDPExternalStreamFunction	外部定義ストリーム間演算関数を実装するクラスが継承するインタフェースです。

14.3 SDPExternalFunction インタフェース

説明

外部定義集合関数および外部定義スカラ関数を実装するクラスが継承するインタフェースです。外部定義集合関数および外部定義スカラ関数のクラスの初期化処理を実装します。

メソッド

SDPExternalFunction インタフェースのメソッド一覧を次に示します。

戻り値	メソッド名	説明
void	refresh()	外部定義集合関数または外部定義スカラ関数の実装クラスが持つフィールドの初期化処理を実装します。

14.3.1 refresh()メソッド

形式

```
public void refresh()
```

説明

refresh()メソッドは，外部定義関数の実装クラスが持つフィールドの初期化処理を実装する抽象メソッドです。

SDP サーバは，外部定義関数の使用個所ごとに実装クラスのインスタンスを作成し，次の契機でこのメソッドを実行します。

- クエリグループを停止または閉塞したとき
- 外部定義集合関数の使用個所で，クエリグループのすべての入力ストリームにStreamInput#putEnd()を実行したとき
- 処理対象のタプル集合からタプルがなくなったとき（RANGE ウィンドウの生存期間を超過した場合など）

パラメーター

なし。

例外

なし。

戻り値

なし。

注意事項

このメソッドで初期化しないフィールドは，値の変更がクエリグループを削除するまで維持されます。

14.4 SDPExternalStreamFunction インタフェース

説明

外部定義ストリーム間演算関数を実装するクラスが継承するインタフェースです。ストリーム間演算関数の処理の抽象メソッドを実装します。

メソッド

SDPExternalStreamFunction インタフェースのメソッド一覧を次に示します。

戻り値	メソッド名	説明
List<Object[]>	executeStreamFunc(Collection<Object[]>[] *ObjectArrayCollection*,Timestamp *timestamp*)	外部定義ストリーム間演算関数の処理を実装します。
void	initialize()	外部定義ストリーム間演算関数の実装クラスが持つフィールドなどの初期化処理を実装します。
void	terminate()	外部定義ストリーム間演算関数の実装クラスが持つフィールドのクリアなど，終了処理を実装します。

14.4.1 executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection,Timestamp timestamp)メソッド

形式

```
abstract List<Object[]> executeStreamFunc(Collection<Object[]>[] ObjectArrayCollection,Timestamp timestamp)
```

説明

外部定義ストリーム間演算関数の処理を実装する抽象メソッドです。

SDP サーバは，クエリ定義で指定した外部定義関数の入力ストリームにタプルが到着したときに，このメソッドを実行します。

パラメーター

ObjectArrayCollection

Collection<Object[]>[]型で，入力ストリーム別に，同時刻に発生したタプルが格納されたデータオブジェクト配列を指定します。

timestamp

Timestamp 型で，ストリームデータのタプルの時刻情報を指定します。

例外

なし。

戻り値

出力ストリームのタプルが格納されたデータオブジェクト配列のリスト(List <Object[]>型オブジェクト)。

注意事項

1 つの入力ストリームに対して複数のタプルが入力された場合，そのタプル間の順序は不定となります。メソッドでは，引数で渡されるタプルの順序に依存しない処理を実装する必要があります。

14.4.2 initialize()メソッド

形式

```
abstract void initialize()
```

説明

initialize()メソッドは，外部定義関数の実装クラスが持つフィールドなどの初期化処理を実装する抽象メソッドです。

SDP サーバは，外部定義関数の使用個所ごとに実装クラスのインスタンスを作成し，次の契機でこのメソッドを実行します。

- クエリグループを開始したとき
- StreamInput インタフェースのputEnd()メソッドを実行したときのterminate()メソッド実行後

パラメーター

なし。

例外

なし。

戻り値

なし。

注意事項

なし。

14.4.3 terminate()メソッド

形式

```
abstract void terminate()
```

説明

外部定義関数の実装クラスが持つフィールドのクリアなど，終了処理を実装する抽象メソッドです。

SDP サーバは，外部定義関数の使用個所ごとに実装クラスのインスタンスを作成し，次の契機でこのメソッドを実行します。

- クエリグループを停止または閉塞したとき
- クエリグループのすべての入力ストリームにStreamInput インタフェースのputEnd()メソッドを実行したとき

パラメーター

なし。

例外

なし。

戻り値

なし。

注意事項

このメソッドでクリアしないフィールドは，クエリグループを削除するまで値の変更が保持されます。

15 外部定義関数インタフェース（C）

この章では，C の外部定義関数について説明します。

15.1　関数

関数の説明で使用する形式について説明します。*関数名*はユーザーが指定する外部定義関数名を表します。この章のすべての表の入力／出力列は，関数でパラメーターがどのように処理されるかを示します（アドレスによって渡されたパラメーターに対する処理を示します）。「入力」は，パラメーターで指定された値が関数に入力されることを表します（変数の値を入力できますが，変更できません）。「出力」は，パラメーターで指定された値が関数によって設定されて，呼び出し元の関数に返されることを表します（インタフェース内で変数の値を設定し，メインプログラムに返します。初期値はすべて無視されます）。なお，表内の「－」は，該当しないことを示します。

次の３つのＣ関数を作成する必要があります。

- 初期化
- 実行
- 終了

15.1.1　初期化

形式

```
int FUNC-NAME_initialize(
HSDPExternalFunctionPtr externalFunctionPtr,
void ** userArea
);
```

説明

この関数は，外部定義関数の初期化を実装します。

SDP サーバは，それぞれの外部定義関数が使用するエリアに実装関数のインスタンスを作成します。この関数は，クエリグループが起動されると実行されます。

前提条件

なし

パラメーター

パラメーター	パラメーターの説明	値の範囲	入力／出力
externalFunctionPtr	HSDPExternalFunction 構造体に対するポインターを指定します。	HSDPExternalFunctionPtr 詳細については，「15.2　外部定義関数のＣの API」を参照してください。	入力

パラメーター	パラメーターの説明	値の範囲	入力／出力
userArea	この関数で割り当てられた，新しいエリアに対するポインターを指定します。外部定義関数は，このパラメーターで指定されたエリアに格納されます。	この関数で割り当てられたエリアのアドレス。	出力

戻り値

値	意味
HSDP_OK	外部定義関数が正常に初期化されました。
HSDP_OK 以外	初期化中にエラーが発生しました。

エラー

なし

注意事項

ユーザーが独自に定義したエリア（領域）を使用する場合は，*userArea* の構造体に定義してください。

15.1.2 実行

形式

```
int FUNC-NAME_execute(
HSDPExternalFunctionPtr externalFunctionPtr,
void * userArea,
HSDPTuplePtr tuplePtr
);
```

説明

この関数は，外部定義関数の処理を実装します。

SDP サーバは，クエリ定義内で指定されている外部定義関数の入力ストリームにタプルが到達すると，この関数を実行します。

前提条件

なし

パラメーター

パラメーター	パラメーターの説明	値の範囲	入力／出力
externalFunctionPtr	HSDPExternalFunction 構造体に対するポインターを指定します。	HSDPExternalFunctionPtr。 詳細については，「15.2　外部定義関数の C の API」を参照してください。	入力
userArea	初期化関数で割り当てられたエリアに対するポインターを指定します。	－	出力
tuplePtr	入力ストリーム内のタプルに対するポインターを指定します。	HSDPTuplePtr 構造体に対するポインター。	入力

戻り値

値	意味
HSDP_OK	外部定義関数が正常に実行されました。
HSDP_OK 以外	実行時にエラーが発生しました。

エラー

なし

注意事項

なし

15.1.3　終了

形式

```
int FUNC-NAME_terminate
( HSDPExternalFunctionPtr externalFunctionPtr,
void * userArea
);
```

説明

この関数は，外部定義関数の終了処理（クリアなど）を実装します。

SDP サーバは，それぞれの外部定義関数が使用するエリアに実装関数のインスタンスを作成します。この関数は，クエリグループが停止または保留されると実行されます。

前提条件

なし

パラメーター

パラメーター	パラメーターの説明	値の範囲	入力／出力
externalFunctionPtr	HSDPExternalFunction 構造体に対するポインターを指定します。	HSDPExternalFunctionPtr。 詳細については，「15.2　外部定義関数の C の API」を参照してください。	入力
userArea	初期化関数で割り当てられたエリアに対するポインターを指定します。	－	出力

戻り値

値	意味
HSDP_OK	外部定義関数が正常に終了されました。
HSDP_OK 以外	終了中にエラーが発生しました。

エラー

なし

注意事項

この関数でクリアされない構造体のメンバー変数については，クエリグループが削除されるまで，変更された値が維持されます。

15.2 外部定義関数の C の API

<hsdpedf.h>ヘッダーを含めて，次に示す基本的なデータ型，構造体，および関数を使用します。ヘッダーは，/opt/hitachi/hsdp/sdk/lib/include/にあります。

次に，定義済みの基本的なデータ型，構造体，および関数を示します。

型	説明
HSDPIntType	int
HSDPShortType	short
HSDPByteType	char
HSDPLongType	long long
HSDPFloatType	float
HSDPDoubleType	double
HSDPCharType	char * データ形式は，固定長の文字列です。
HSDPTimestampType	タイムスタンプ情報です。
HSDPVarCharType	データ形式は，可変長の文字列です。

構造体	説明
HSDPExternalFunction	外部定義関数のパラメーター
HSDPFieldInfo	フィールド情報
HSDPSchema	タプルのスキーマのパラメーター
HSDPTimestampType	タイムスタンプ情報
HSDPTuple	タプルのパラメーター
HSDPVarCharType	可変長文字列

これらの構造体には，SDP サーバの内部で使用されるメンバーも含まれています。ただし，内部で使用されるメンバーについては，このマニュアルでは説明していません。

関数	説明
HSDPTuple_set ()	指定したスキーマに対応するタプルを作成します。
HSDPTuple_getInt ()	タプルからint 型として値を取得します。
HSDPTuple_getShort ()	タプルからshort 型として値を取得します。
HSDPTuple_getByte ()	タプルからchar 型として値を取得します。
HSDPTuple_getLong ()	タプルからlong long 型として値を取得します。

関数	説明
HSDPTuple_getFloat ()	タプルからfloat 型として値を取得します。
HSDPTuple_getDouble ()	タプルからdouble 型として値を取得します。
HSDPTuple_getChar ()	タプルからchar *型として値を取得します。
HSDPTuple_getVarChar ()	タプルからchar *型として値を取得します。
HSDPTuple_getTimestamp ()	タプルからtimestamp 型として値を取得します。
HSDPTuple_setInt ()	指定した値をint 型としてタプルに設定します。
HSDPTuple_setShort ()	指定した値をshort 型としてタプルに設定します。
HSDPTuple_setByte ()	指定した値をchar 型としてタプルに設定します。
HSDPTuple_setLong ()	指定した値をlong long 型としてタプルに設定します。
HSDPTuple_setFloat ()	指定した値をfloat 型としてタプルに設定します。
HSDPTuple_setDouble ()	指定した値をdouble 型としてタプルに設定します。
HSDPTuple_setChar ()	指定した値をchar *型としてタプルに設定します。
HSDPTuple_setVarChar ()	指定した値をchar *型としてタプルに設定します。
HSDPTuple_setTimestamp ()	指定した値をtimestamp 型としてタプルに設定します。
HSDPTuple_getSystemTimestamp ()	タプルからシステムタイムスタンプを取得します。
HSDPSchema_init ()	HSDPSchema の構造体を初期化します。
HSDPSchema_term ()	HSDPSchema の構造体を終了します。
HSDPSchema_getFieldInfo ()	HSDPSchema の構造体からHSDPFieldInfo の構造体を取得します。
HSDPSchema_getSize ()	スキーマのデータサイズを取得します。
HSDPSchema_getNumField ()	フィールド数を取得します。
HSDPFieldInfo_init ()	HSDPFieldInfo の構造体を初期化します。
HSDPFieldInfo_term ()	HSDPFieldInfo の構造体を終了します。
HSDPFieldInfo_getName ()	スキーマフィールドの名前を取得します。
HSDPFieldInfo_getType ()	スキーマフィールドの型を取得します。
HSDPFieldInfo_getSize ()	スキーマフィールドのサイズを取得します。
HSDPFieldInfo_getOffset ()	スキーマフィールドのオフセットを取得します。
HSDPFieldInfo_setName ()	スキーマフィールドの名前を設定します。
HSDPFieldInfo_setType ()	スキーマフィールドの型を設定します。
HSDPFieldInfo_setSize ()	スキーマフィールドのサイズを設定します。
HSDPFieldInfo_setOffset ()	タプルの先頭からのオフセットを設定します。

関数	説明
HSDPVarCharType_init ()	HSDPVarCharType の構造体を初期化します。
HSDPVarCharType_term ()	HSDPVarCharType の構造体を終了します。
HSDPVarCharType_getSize ()	HSDPVarCharType のサイズを取得します。
HSDPVarCharType_getString ()	HSDPVarCharType の文字列を取得します。
HSDPVarCharType_setString ()	HSDPVarCharType に文字列を設定します。
HSDPTimestampType_init ()	HSDPTimestampType の構造体を初期化します。
HSDPTimestampType_term ()	HSDPTimestampType の構造体を終了します。
HSDPTimestampType_getMsec ()	ミリ秒の情報を取得します。
HSDPTimestampType_getNano ()	ナノ秒の情報を取得します。
HSDPTimestampType_setMsec ()	ミリ秒の情報を設定します。
HSDPTimestampType_setNano ()	ナノ秒の情報を設定します。
HSDPExternalFunction_allocInputTuple ()	入力ストリームキューからのタプルを割り当てます。
HSDPExternalFunction_deallocInputTuple ()	入力ストリームキューからのタプルの割り当てを解除します。
HSDPExternalFunction_allocOutputTuple ()	出力ストリームキューからのタプルを割り当てます。
HSDPExternalFunction_deallocOutputTuple ()	出力ストリームキューからのタプルの割り当てを解除します。
HSDPExternalFunction_addTuple ()	出力ストリームキューにタプルを追加します。
HSDPExternalFunction_getInputTupleSchema ()	入力タプルのスキーマを取得します。
HSDPExternalFunction_getOutputTupleSchema ()	出力タプルのスキーマを取得します。
HSDPExternalFunction_getIntegerInitParameter ()	整数型の初期パラメーターを取得します。
HSDPExternalFunction_getLongInitParameter ()	long 型の初期パラメーターを取得します。
HSDPExternalFunction_getDoubleInitParameter ()	double 型の初期パラメーターを取得します。
HSDPExternalFunction_getStringInitParameter ()	日付データ，時刻データ，時刻印データ，または文字列型の初期パラメーターを取得します。

15.3　構造体

15.3.1　HSDPExternalFunction 構造体

名前

HSDPExternalFunction

説明

この構造体には，外部定義関数用のパラメーターが含まれています。

15.3.2　HSDPFieldInfo 構造体

名前

HSDPFieldInfo

説明

この構造体には，フィールド情報が含まれています。

HSDPFieldInfoPtr は，HSDPFieldInfo 構造体のポインター変数です。

15.3.3　HSDPSchema 構造体

名前

HSDPSchema

説明

この構造体には，タプルのスキーマが含まれています。

HSDPSchemaPtr は，HSDPSchema 構造体のポインター変数です。

15.3.4　HSDPTimestampType 構造体

名前

HSDPTimestampType

説明

この構造体には，タイムスタンプ情報が含まれています。小数部分を含むタイムスタンプ情報を指定する場合は，必ずナノ秒単位まで指定してください。

15.3.5　HSDPVarCharType 構造体

名前

HSDPVarCharType

説明

この構造体には，可変長文字列のパラメーターが含まれています。

メンバー

メンバー	型	説明	値の範囲
size	unsigned short	文字列の数。	1～32,767 の整数。
string	char *	文字列。	1～32,767 の半角文字。

15.3.6　HSDPTuple 構造体

名前

HSDPTuple

説明

この構造体には，タプル用のパラメーターが含まれています。

HSDPTuplePtr は，HSDPTuple 構造体のポインター変数です。

メンバー

メンバー	型	説明	値の範囲
ts	HSDPTimestampType	タプルのシステム時刻。	－
dataPtr	char *	データの格納先。	－

15.4　関数の詳細

15.4.1　HSDPTuple_set ()

形式

```
int HSDPTuple_set( HSDPTuplePtr _this, HSDPSchemaPtr schemaPtr,... );
```

説明

この関数は，指定したスキーマに対応するタプルを作成します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	タプルが格納されている場所を指定します。	－	入力
schemaPtr	スキーマが格納されている場所を指定します。	－	入力
...	タプルに設定する引数の変数です。スキーマの順に値を指定します。	－	入力

戻り値

値	意味
HSDP_OK	タプルが正常に作成されました。
HSDP_NULLPOINTER	タプルの作成中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.2 HSDPTuple_get*Type* ()

形式

```
int HSDPTuple_getTYPE( HSDPTuplePtr _this, int index, HSDPSchemaPtr schemaPtr, HSDPTYPETypeP
tr getValue );
```

次の表に，*TYPE* として指定できる値を示します。

TYPE	C のデータ型
Int	int
Short	short
Byte	char
Long	long long
Float	float
Double	double
Char	char *
VarChar	HSDPVarCharType
Timestamp	HSDPTimestampType

説明

この関数は，*TYPE* 型としてタプルから指定された値を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	タプルが格納されている場所を指定します。	－	入力
index	HSDPschemaPtr にある HSDPFieldInfoPtr のインデックスです。	－	入力
schemaPtr	スキーマが格納されている場所を指定します。	－	入力
getValue	値が格納されている場所を指定します。	－	出力

戻り値

値	意味
HSDP_OK	指定した値がタプルから取得されました。
HSDP_NULLPOINTER	タプルから値を取得中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。
HSDP_ERR_MEMORY_ALLOCATE※	メモリの割り当てに失敗しました。
※：HSDPTuple_getVarChar ()は，値だけ返します。	

注意事項

関数HSDPTuple_getChar ()によって提供されたデータは，null 終端バイト文字列ではありません。

15.4.3 HSDPTuple_set*Type* ()

形式

```
int HSDPTuple_setTYPE( HSDPTuplePtr _this, int index, HSDPTYPEType setValue, HSDPSchemaPtr schemaPtr );
```

次の値を *TYPE* として指定できます。

TYPE	C のデータ型
Int	int
Short	short
Byte	char
Long	long long
Float	float
Double	double
Char	char *
VarChar	HSDPVarCharType
Timestamp	HSDPTimestampType

説明

この関数は，指定した値を *TYPE* 型としてタプルに設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	タプルが格納されている場所を指定します。	－	入力
index	HSDPschemaPtr にある HSDPFieldInfoPtr のインデックスです。	－	入力
setValue	タプルに設定する値を指定します。	－	入力
schemaPtr	スキーマが格納されている場所を指定します。	－	入力

戻り値

値	意味
HSDP_OK	指定した値がタプルに設定されました。
HSDP_NULLPOINTER	値をタプルに設定中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.4 HSDPTuple_getSystemTimestamp ()

形式

```
int HSDPTuple_getSystemTimestamp( HSDPTuplePtr _this, HSDPTimestampType * getValue );
```

説明

この関数は，タプルからシステムタイムスタンプを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	タプルが格納されている場所を指定します。	－	入力
getValue	値が格納されている場所のシステムタイムスタンプを指定します。	－	出力

戻り値

値	意味
HSDP_OK	タプルからシステムタイムスタンプが取得されました。
HSDP_NULLPOINTER	タプルからシステムタイムスタンプを取得中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.5 HSDPSchema_init ()

形式

```
int HSDPSchema_init( int numField, HSDPFieldInfoPtr * fieldInfoPtrs, HSDPSchemaPtr * schemaPtr );
```

説明

この関数は，HSDPSchema の構造体を初期化します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
numField	フィールドの数。	1～3,000 の整数。	入力
fieldInfoPtrs	フィールド情報リストが格納されている場所。	－	入力

パラメーター	説明	値の範囲	入力／出力
schemaPtr	スキーマが格納されている場所を指定します。	－	出力

戻り値

値	意味
HSDP_OK	HSDPSchema が正常に初期化されました。
HSDP_NULLPOINTER	HSDPSchema の初期化中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。
HSDP_ERR_MEMORY_ALLOCATE	メモリの割り当てに失敗しました。

注意事項

なし。

15.4.6 HSDPSchema_term ()

形式

```
int HSDPSchema_term ( HSDPSchemaPtr schemaPtr);
```

説明

この関数は，HSDPSchema の構造体を終了します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
schemaPtr	HSDPSchema が格納されている場所を指定します。	－	入力

戻り値

値	意味
HSDP_OK	HSDPSchema が正常に終了されました。

値	意味
HSDP_NULLPOINTER	HSDPSchema の終了中にエラーが発生しました。

注意事項

なし。

15.4.7 HSDPSchema_getFieldInfo ()

形式

```
int HSDPSchema_getFieldInfo ( HSDPSchemaPtr _this, int index, HSDPFieldInfoPtr * fieldInfo )
;
```

説明

この関数は，HSDPSchema の構造体からHSDPFieldInfo の構造体を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPSchema が格納されている場所を指定します。	－	入力
index	HSDPschemaPtr にある HSDPFieldInfoPtr のインデックスです。	*numField* より小さい整数。	入力
fieldInfo	フィールド情報が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	構造体が正常に取得されました。
HSDP_NULLPOINTER	構造体の取得中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.8 HSDPSchema_getSize ()

形式

```
int HSDPSchema_getSize( HSDPSchemaPtr _this, int * size );
```

説明

この関数は，スキーマのデータサイズを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPSchema が格納されている場所を指定します。	－	入力
size	スキーマのサイズ。	1～768,000 の整数。	出力

戻り値

値	意味
HSDP_OK	データサイズが正常に取得されました。
HSDP_NULLPOINTER	データサイズの取得中にエラーが発生しました。

注意事項

なし。

15.4.9 HSDPSchema_getNumField ()

形式

```
int HSDPSchema_getNumField( HSDPSchemaPtr _this, int * numField );
```

説明

この関数は，フィールドの数を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPSchema が格納されている場所を指定します。	−	入力
numField	フィールドの数。	1～3,000 の整数。	出力

戻り値

値	意味
HSDP_OK	フィールド数が正常に取得されました。
HSDP_NULLPOINTER	フィールド数の取得中にエラーが発生しました。

注意事項

なし。

15.4.10　HSDPFieldInfo_init ()

形式

```
int HSDPFieldInfo_init( char * name, HSDPFieldType type, int size, HSDPFieldInfoPtr * fieldInfoPtr );
```

説明

この関数は，HSDPFieldInfo の構造体を初期化します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
name	フィールド名。	このメンバーを指定する場合，次の規則が適用されます。 • 名前には，半角英数字とアンダースコア（_）だけを使用できます。 • 名前は半角英字で開始し，数字やアンダースコア（_）で開始はできません。 • 名前は，大文字小文字を区別しません。文字はすべて半角大文字として扱われます。 • 名前にスペースを含めることはできません。 • CQL の予約語を名前に使用できません（予約語ではないキーワードと同じ名前にはできます）。 • 名前を二重引用符（”）では囲めません。 • 名前に指定できる文字数は 1～100 文字です。	入力
type	フィールドの型。	次の値のうちの 1 つです。 • HSDPFieldType_Int int • HSDPFieldType_Short short • HSDPFieldType_Byte char • HSDPFieldType_Long long long • HSDPFieldType_Float float • _Double double • HSDPFieldType_Char char * • HSDPFieldType_Varchar HSDPVarCharType • HSDPFieldType_Timestamp HSDPTimestampType	入力
size	フィールドのサイズ。	1～32,767 の整数。	入力

パラメーター	説明	値の範囲	入力／出力
fieldInfoPtr	フィールド情報が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	HSDPFieldInfo の構造体が正常に初期化されました。
HSDP_NULLPOINTER	HSDPFieldInfo の構造体の初期化中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。
HSDP_ERR_MEMORY_ALLOCATE	メモリの割り当てに失敗しました。

注意事項

なし。

15.4.11　HSDPFieldInfo_term ()

形式

```
int HSDPFieldInfo_term( HSDPFieldInfoPtr fieldInfoPtr );
```

説明

この関数は，HSDPFieldInfo の構造体を終了します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
fieldInfoPtr	フィールド情報が格納されている場所。	－	入力

戻り値

値	意味
HSDP_OK	HSDPFieldInfo が正常に終了しました。

値	意味
HSDP_NULLPOINTER	HSDPFieldInfo の終了中にエラーが発生しました。

注意事項

なし。

15.4.12 HSDPFieldInfo_getName ()

形式

```
int HSDPFieldInfo_getName( HSDPFieldInfoPtr _this, char ** name );
```

説明

この関数は，スキーマフィールドの名前を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	－	入力
name	フィールド名。	このメンバーを指定する場合，次の規則が適用されます。 • 名前には，半角英数字とアンダースコア（_）だけを使用できます。 • 名前は半角英字で開始し，数字やアンダースコア（_）で開始はできません。 • 名前は，大文字小文字を区別しません。文字はすべて半角大文字として扱われます。 • 名前にスペースを含めることはできません。 • CQL の予約語を名前に使用できません（予約語ではないキーワードと同じ名前にはできます）。	出力

パラメーター	説明	値の範囲	入力／出力
name	フィールド名。	• 名前を二重引用符（”）では囲めません。 • 名前に指定できる文字数は1～100文字です。	出力

戻り値

値	意味
HSDP_OK	スキーマフィールドの名前が正常に取得されました。
HSDP_NULLPOINTER	スキーマフィールドの名前の取得中にエラーが発生しました。

注意事項

なし。

15.4.13 HSDPFieldInfo_getType ()

形式

```
int HSDPFieldInfo_getType( HSDPFieldInfoPtr _this, HSDPFieldType * type );
```

説明

この関数は，スキーマフィールドの型を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	－	入力
type	フィールドの型。	次の値のうちの1つ。 • HSDPFieldType_Int int • HSDPFieldType_Short short • HSDPFieldType_Byte char	出力

パラメーター	説明	値の範囲	入力／出力
type	フィールドの型。	• HSDPFieldType_Long long long • HSDPFieldType_Float float • _Double double • HSDPFieldType_Char char * • HSDPFieldType_Varchar HSDPVarCharType • HSDPFieldType_Timestamp HSDPTimestampType	出力

戻り値

値	意味
HSDP_OK	スキーマフィールドの型が正常に取得されました。
HSDP_NULLPOINTER	スキーマフィールドの型の取得中にエラーが発生しました。

注意事項

なし。

15.4.14 HSDPFieldInfo_getSize ()

形式

```
int HSDPFieldInfo_getSize( HSDPFieldInfoPtr _this, int * size );
```

説明

この関数は，スキーマフィールドのサイズを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	−	入力
size	フィールドのサイズ。	1～32,767 の整数。	出力

戻り値

値	意味
HSDP_OK	スキーマフィールドのサイズが正常に取得されました。
HSDP_NULLPOINTER	スキーマフィールドのサイズの取得中にエラーが発生しました。

注意事項

なし。

15.4.15　HSDPFieldInfo_getOffset ()

形式

```
int HSDPFieldInfo_getOffset( HSDPFieldInfoPtr _this, int * offset );
```

説明

この関数は，タプルの先頭からのオフセットを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	−	入力
offset	タプルの先頭からのオフセット。	−	出力

戻り値

値	意味
HSDP_OK	オフセットが正常に取得されました。
HSDP_NULLPOINTER	オフセットの取得中にエラーが発生しました。

注意事項

なし。

15.4.16　HSDPFieldInfo_setName ()

形式

```
int HSDPFieldInfo_setName( HSDPFieldInfoPtr _this, char * name );
```

説明

この関数は，スキーマフィールドの名前を設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	－	出力
name	フィールド名	このメンバーを指定する場合，次の規則が適用されます。 • 名前には，半角英数字とアンダースコア（_）だけ使用できます。 • 名前の先頭文字には，半角英字を指定します。数字やアンダースコア（_）を名前の先頭文字には指定できません。 • 名前は，大文字小文字を区別しません。文字はすべて半角大文字として扱われます。	入力

パラメーター	説明	値の範囲	入力／出力
name	フィールド名	• 名前にスペースを含めることはできません。 • CQL の予約語は名前に使用できません（予約語ではないキーワードと同じ名前は使用できます）。 • 名前を二重引用符（”）で囲めません。 • 名前に指定できる文字数は 1～100 文字です。	入力

戻り値

値	意味
HSDP_OK	スキーマフィールドの名前が正常に設定されました。
HSDP_NULLPOINTER	スキーマフィールドの名前を設定中にエラーが発生しました。

注意事項

なし。

15.4.17 HSDPFieldInfo_setType ()

形式

```
int HSDPFieldInfo_setType( HSDPFieldInfoPtr _this, HSDPFieldType type );
```

説明

この関数は，スキーマフィールドの型を設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	－	出力
type	フィールドの型。	次の値のうちの 1 つ。	入力

パラメーター	説明	値の範囲	入力／出力
type	フィールドの型。	• HSDPFieldType_Int int • HSDPFieldType_Short short • HSDPFieldType_Byte char • HSDPFieldType_Long long long • HSDPFieldType_Float float • HSDPFieldType_Double double • HSDPFieldType_Char char * • HSDPFieldType_Varchar HSDPVarCharType • HSDPFieldType_Timestamp HSDPTimestampType	入力

戻り値

値	意味
HSDP_OK	スキーマフィールドの型が正常に設定されました。
HSDP_NULLPOINTER	スキーマフィールドの型を設定中にエラーが発生しました。

注意事項

なし。

15.4.18 HSDPFieldInfo_setSize ()

形式

```
int HSDPFieldInfo_setSize( HSDPFieldInfoPtr _this, int size );
```

説明

この関数は，スキーマフィールドのサイズを設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	−	出力
size	スキーマフィールドのサイズ。	1～32,767 の整数。	入力

戻り値

値	意味
HSDP_OK	スキーマフィールドのサイズが正常に設定されました。
HSDP_NULLPOINTER	スキーマフィールドのサイズを設定中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.19　HSDPFieldInfo_setOffset ()

形式

```
int HSDPFieldInfo_setOffset( HSDPFieldInfoPtr _this, int offset );
```

説明

この関数は，タプルの先頭からのオフセットを設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	フィールド情報が格納されている場所。	−	出力

パラメーター	説明	値の範囲	入力／出力
offset	タプルの先頭からのオフセット。	－	入力

戻り値

値	意味
HSDP_OK	タプルの先頭からのオフセットが正常に設定されました。
HSDP_NULLPOINTER	タプルの先頭からのオフセットの設定中にエラーが発生しました。

注意事項

なし。

15.4.20　HSDPVarCharType_init ()

形式

```
int HSDPVarCharType_init( char * string, HSDPVarCharTypePtr * varCharStringPtr );
```

説明

この関数は，HSDPVarCharType の構造体を初期化します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
string	文字列。	1～32,767 の半角文字。	入力
varCharStringPtr	HSDPVarCharTypePtr が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	HSDPVarCharType の構造体が正常に初期化されました。
HSDP_NULLPOINTER	HSDPVarCharType の構造体の初期化中にエラーが発生しました。

値	意味
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.21 HSDPVarCharType_term ()

形式

```
int HSDPVarCharType_term(HSDPVarCharTypePtr varCharStringPtr);
```

説明

この関数は，HSDPVarCharType の構造体を初期化します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
varCharStringPtr	HSDPVarCharTypePtr が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	HSDPVarCharType の構造体が正常に終了されました。
HSDP_NULLPOINTER	HSDPVarCharType の構造体の終了中にエラーが発生しました。

注意事項

なし。

15.4.22　HSDPVarCharType_getSize ()

形式

```
int HSDPVarCharType_getSize(HSDPVarCharTypePtr _this, unsigned short * size);
```

説明

この関数は，HSDPVarCharType のサイズを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPVarCharTypePtr が格納されている場所。	－	入力
size	文字列の数。	1～32,767 の整数。	出力

戻り値

値	意味
HSDP_OK	HSDPVarCharType のサイズが正常に取得されました。
HSDP_NULLPOINTER	HSDPVarCharType のサイズの取得中にエラーが発生しました。

注意事項

なし。

15.4.23　HSDPVarCharType_getString ()

形式

```
int HSDPVarCharType_getString( HSDPVarCharTypePtr _this, char ** string );
```

説明

この関数は，HSDPVarCharType の文字列を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPVarCharTypePtr が格納されている場所。	－	入力
string	文字列。	1～32,767 の半角文字。	出力

戻り値

値	意味
HSDP_OK	HSDPVarCharType の文字列が正常に取得されました。
HSDP_NULLPOINTER	HSDPVarCharType の文字列の取得中にエラーが発生しました。

注意事項

なし。

15.4.24　HSDPVarCharType_setString ()

形式

```
int HSDPVarCharType_setString( HSDPVarCharTypePtr _this, char * string );
```

説明

この関数は，HSDPVarCharType の文字列を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPVarCharTypePtr が格納されている場所。	－	出力
string	文字列。	1～32,767 の半角文字。	入力

戻り値

値	意味
HSDP_OK	HSDPVarCharType の文字列が正常に取得されました。
HSDP_NULLPOINTER	HSDPVarCharType の文字列の設定中にエラーが発生しました。

注意事項

なし。

15.4.25 HSDPTimestampType_init ()

形式

```
int HSDPTimestampType_init( long msec, int nano, HSDPTimestampTypePtr * timestampPtr );
```

説明

この関数は，HSDPTimestampType の構造体を初期化します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
msec	この日を表す 1970 年 1 月 1 日，00:00:00 GMT からの経過ミリ秒数。	-2,147,483,648～2,147,483,647 の範囲。	入力
nano	小数部分の秒を表す新しいコンポーネント	0～999,999,999 の整数。	入力
timestampPtr	HSDPTimestampTypePtr が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	HSDPTimestampType の構造体が正常に初期化されました。
HSDP_NULLPOINTER	HSDPTimestampType の構造体の初期化中にエラーが発生しました。

値	意味
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。
HSDP_ERR_MEMORY_ALLOCATE	メモリの割り当てに失敗しました。

注意事項

なし。

15.4.26 HSDPTimestampType_term ()

形式

```
int HSDPTimestampType_term( HSDPTimestampTypePtr timestampPtr );
```

説明

この関数は，HSDPTimestampType の構造体を終了します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
timestampPtr	HSDPTimestampTypePtr が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	HSDPTimestampType の構造体が正常に終了されました。
HSDP_NULLPOINTER	HSDPTimestampType の構造体の終了中にエラーが発生しました。

注意事項

なし。

15.4.27　HSDPTimestampType_getMsec ()

形式

```
int HSDPTimestampType_getMsec( HSDPTimestampTypePtr _this, long * msec );
```

説明

この関数は，ミリ秒の情報を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPTimestampTypePtr が格納されている場所。	－	入力
msec	この日を表す 1970 年 1 月 1 日，00:00:00 GMT からの経過ミリ秒数。	-2,147,483,648～2,147,483,647 の範囲。	出力

戻り値

値	意味
HSDP_OK	ミリ秒の情報が正常に取得されました。
HSDP_NULLPOINTER	ミリ秒の情報取得中にエラーが発生しました。

注意事項

なし。

15.4.28　HSDPTimestampType_getNano ()

形式

```
int HSDPTimestampType_getNano( HSDPTimestampTypePtr _this, int * nano );
```

説明

この関数は，ナノ秒の情報を取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPTimestampTypePtr が格納されている場所。	－	入力
nano	小数部分の秒を表す新しいコンポーネント	0～999,999,999 の整数。	出力

戻り値

値	意味
HSDP_OK	ナノ秒の情報が正常に取得されました。
HSDP_NULLPOINTER	ナノ秒の情報取得中にエラーが発生しました。

注意事項

なし。

15.4.29　HSDPTimestampType_setMsec ()

形式

```
int HSDPTimestampType_setMsec( HSDPTimestampTypePtr _this, long msec );
```

説明

この関数は，ミリ秒の情報を設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPTimestampTypePtr が格納されている場所。	－	出力

パラメーター	説明	値の範囲	入力／出力
msec	この日を表す 1970 年 1 月 1 日，00:00:00 GMT からの経過ミリ秒数。	-2,147,483,648～2,147,483,647 の範囲。	入力

戻り値

値	意味
HSDP_OK	ミリ秒の情報が正常に設定されました。
HSDP_NULLPOINTER	ミリ秒の情報を設定中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.30 HSDPTimestampType_setNano ()

形式

```
int HSDPTimestampType_setNano( HSDPTimestampTypePtr _this, int nano );
```

説明

この関数は，ナノ秒の情報を設定します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPTimestampTypePtr が格納されている場所。	－	出力
nano	小数部分の秒を表す新しいコンポーネント	0～999,999,999 の整数。	入力

戻り値

値	意味
HSDP_OK	ナノ秒の情報が正常に設定されました。
HSDP_NULLPOINTER	ナノ秒の情報を設定中にエラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限を超えています。

注意事項

なし。

15.4.31 HSDPExternalFunction_allocInputTuple ()

形式

```
int HSDPExternalFunction_allocInputTuple( HSDPExternalFunctionPtr _this, HSDPTuplePtr * tupleArray, int allocSize, int * allocedSize );
```

説明

この関数は，入力ストリームキューからのタプルを割り当てます。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
tupleArray	タプルのアレイ。	－	出力
allocSize	取得するタプルのサイズ。	0 より大きい整数	入力
allocedSize	取得したタプルのサイズ。	－	出力

戻り値

値	意味
HSDP_OK	入力ストリームキューからのタプルが正常に割り当てられました。

値	意味
HSDP_NULLPOINTER	入力ストリームキューからのタプルの割り当て中に，エラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限（負の値）を超えています。
HSDP_ERR_INTERNAL	入力ストリームキューからのタプルの割り当てに失敗しました。

注意事項

なし。

15.4.32 HSDPExternalFunction_deallocInputTuple()

形式

```
int HSDPExternalFunction_deallocInputTuple( HSDPExternalFunctionPtr _this, HSDPTuplePtr * tupleArray, int deallocSize );
```

説明

この関数は，入力ストリームキューからのタプルの割り当てを解除し，タプルを戻します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
tupleArray	タプルのアレイ。	－	出力
deallocSize	割り当てを解除するタプルのサイズ。	0 より大きい整数	入力

戻り値

値	意味
HSDP_OK	入力ストリームキューからのタプルが正常に割り当て解除されました。
HSDP_NULLPOINTER	入力ストリームキューからのタプルの割り当て解除中に，エラーが発生しました。

値	意味
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限（負の値）を超えています。
HSDP_ERR_INTERNAL	入力ストリームキューからのタプルの割り当て解除に失敗しました。

注意事項

なし。

15.4.33 HSDPExternalFunction_allocOutputTuple ()

形式

```
int HSDPExternalFunction_allocOutputTuple( HSDPExternalFunctionPtr _this, HSDPTuplePtr * tupleArray, int allocSize, int * allocedSize );
```

説明

この関数は，出力ストリームキューからのタプルを割り当てます。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
tupleArray	タプルのアレイ。	－	出力
allocSize	取得するタプルのサイズ。	0 より大きい整数	入力
allocedSize	取得したタプルのサイズ。	－	出力

戻り値

値	意味
HSDP_OK	出力ストリームキューからのタプルが正常に割り当てられました。
HSDP_NULLPOINTER	出力ストリームキューからのタプルの割り当て中に，エラーが発生しました。

値	意味
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限（負の値）を超えています。
HSDP_ERR_INTERNAL	出力ストリームキューからのタプルの割り当てに失敗しました。

注意事項

なし。

15.4.34 HSDPExternalFunction_deallocOutputTuple ()

形式

```
int HSDPExternalFunction_deallocOutputTuple( HSDPExternalFunctionPtr _this, HSDPTuplePtr * tupleArray, int deallocSize );
```

説明

この関数は，出力ストリームキューからのタプルの割り当てを解除して，タプルを戻します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
tupleArray	タプルのアレイ。	－	出力
deallocSize	割り当てを解除するタプルのサイズ。	0 より大きい整数	入力

戻り値

値	意味
HSDP_OK	出力ストリームキューからのタプルが正常に割り当てを解除されました。
HSDP_NULLPOINTER	出力ストリームキューからのタプルの割り当ての解除中に，エラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限（負の値）を超えています。

値	意味
HSDP_ERR_INTERNAL	出力ストリームキューからのタプルの割り当て解除に失敗しました。

注意事項

なし。

15.4.35 HSDPExternalFunction_addTuple ()

形式

```
int HSDPExternalFunction_addTuple( HSDPExternalFunctionPtr _this, HSDPTuplePtr * tupleArray,
 int addSize, int * addedSize );
```

説明

この関数は，出力ストリームキューにタプルを追加します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
tupleArray	タプルのアレイ。	－	出力
addSize	追加するタプルのサイズ。	0 より大きい整数	入力
addedSize	追加したタプルのサイズ。	－	出力

戻り値

値	意味
HSDP_OK	タプルが，出力ストリームキューに正常に追加されました。
HSDP_NULLPOINTER	出力ストリームキューへのタプルの追加中に，エラーが発生しました。
HSDP_ERR_OUT_OF_RANGE	数値の引数が制限（負の値）を超えています。
HSDP_ERR_INTERNAL	出力ストリームキューへのタプルの追加に失敗しました。

注意事項

なし。

15.4.36 HSDPExternalFunction_getInputTupleSchema ()

形式

```
int HSDPExternalFunction_getInputTupleSchema( HSDPExternalFunctionPtr _this, HSDPSchemaPtr *
 schemaPtr );
```

説明

この関数は，入力タプルのスキーマを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
schemaPtr	HSDPSchemaPtr が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	入力タプルのスキーマが正常に取得されました。
HSDP_NULLPOINTER	入力タプルのスキーマの取得中にエラーが発生しました。
HSDP_ERR_INTERNAL	入力タプルのスキーマの取得に失敗しました。

注意事項

なし。

15.4.37　HSDPExternalFunction_getOutputTupleSchema ()

形式

```
int HSDPExternalFunction_getOutputTupleSchema( HSDPExternalFunctionPtr _this, HSDPSchemaPtr*
 schemaPtr );
```

説明

この関数は，出力タプルのスキーマを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
schemaPtr	HSDPSchemaPtr が格納されている場所。	－	出力

戻り値

値	意味
HSDP_OK	出力タプルのスキーマが正常に取得されました。
HSDP_NULLPOINTER	出力タプルのスキーマの取得中にエラーが発生しました。
HSDP_ERR_INTERNAL	出力タプルのスキーマの取得に失敗しました。

注意事項

なし。

15.4.38　HSDPExternalFunction_getIntegerInitParameter ()

形式

```
int HSDPExternalFunction_getIntegerInitParameter( HSDPExternalFunctionPtr _this, int index,
int * value );
```

説明

この関数は，整数型の初期パラメーターを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
index	クエリ定義ファイルで定義される初期値パラメーターの列番号。	0～15 の整数	入力
value	整数型の初期パラメーターの値。	－	出力

戻り値

値	意味
HSDP_OK	整数型の初期パラメーターが正常に取得されました。
HSDP_NULLPOINTER	整数型の初期パラメーターの取得中にエラーが発生しました。
HSDP_ERR_INTERNAL	整数型の初期パラメーターの取得に失敗しました。
HSDP_ERR_INVALID_DATA_TYPE	指定されたインデックスの初期パラメーターが整数型ではありません。

注意事項

なし。

15.4.39 HSDPExternalFunction_getLongInitParameter ()

形式

```
int HSDPExternalFunction_getLongInitParameter( HSDPExternalFunctionPtr _this, int index, long * value );
```

説明

この関数は，long 型の初期パラメーターを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
index	クエリ定義ファイルで定義される初期値パラメーターの列番号。	0～15 の整数	入力
value	long 型の初期パラメーターの値。	－	出力

戻り値

値	意味
HSDP_OK	long 型の初期パラメーターが正常に取得されました。
HSDP_NULLPOINTER	long 型の初期パラメーターの取得中にエラーが発生しました。
HSDP_ERR_INTERNAL	long 型の初期パラメーターの取得に失敗しました。
HSDP_ERR_INVALID_DATA_TYPE	指定したインデックスの初期パラメーターが long 型ではありません。

注意事項

なし。

15.4.40 HSDPExternalFunction_getDoubleInitParameter ()

形式

```
int HSDPExternalFunction_getDoubleInitParameter( HSDPExternalFunctionPtr _this, int index, double * value );
```

説明

この関数は，double 型の初期パラメーターを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
index	クエリ定義ファイルで定義される初期値パラメーターの列番号。	0～15 の整数	入力
value	double 型の初期パラメーターの値。	－	出力

戻り値

値	意味
HSDP_OK	double 型の初期パラメーターが正常に取得されました。
HSDP_NULLPOINTER	double 型の初期パラメーターの取得中にエラーが発生しました。
HSDP_ERR_INTERNAL	double 型の初期パラメーターの取得に失敗しました。
HSDP_ERR_INVALID_DATA_TYPE	指定したインデックスの初期パラメーターが double 型ではありません。

注意事項

なし。

15.4.41　HSDPExternalFunction_getStringInitParameter

形式

```
int HSDPExternalFunction_getStringInitParameter( HSDPExternalFunctionPtr _this, int index, char ** value );
```

説明

この関数は，日付データ，時刻データ，時刻印データまたは文字列型の初期パラメーターを取得します。

前提条件

なし。

パラメーター

パラメーター	説明	値の範囲	入力／出力
_this	HSDPExternalFunctionPtr が格納されている場所。	－	入力
index	クエリ定義ファイルで定義される初期値パラメーターの列番号。	0～15 の整数	入力
value	日付データ，時刻データ，時刻印データ，および文字列の初期パラメーターの文字列。	－	出力

戻り値

値	意味
HSDP_OK	日付データ，時刻データ，時刻印データまたは文字列型の初期パラメーターが正常に取得されました。
HSDP_NULLPOINTER	日付データ，時刻データ，時刻印データまたは文字列型の初期パラメーターの取得中にエラーが発生しました。
HSDP_ERR_INTERNAL	日付データ，時刻データ，時刻印データまたは文字列型の初期パラメーターの取得に失敗しました。
HSDP_ERR_INVALID_DATA_TYPE	指定したインデックスの初期パラメーターが日付データ，時刻データ，時刻印データまたは文字列型ではありません。

注意事項

なし。

16 コマンド

この章では，コマンドについて説明します。

16.1 hsdpcliconv

説明

このコマンドは，HSDP クライアントライブラリによって出力されたバイナリファイルのレコードを，CSV 形式に変換します。

形式

```
hsdpcliconv [-s 変換対象の開始時間] [-e 変換対象の終了時間] [-r] -d 変換定義ファイル バイナリファイルが格納されているディレクトリ
```

実行権限

HSDP クライアント運用ユーザー

コマンドを実行するための前提条件

次のファイルとディレクトリがあること。

- 変換定義ファイル
- バイナリファイルが格納されているディレクトリ

格納先ディレクトリ

hsdpcli.tar.gz が展開されたディレクトリ。

引数

-s *変換対象の開始時間*

変換するバイナリファイルのレコードの開始時間を *YYYYmmddHHMMSS*（GMT）形式で指定します。hsdpcliconv コマンドは，データソースのタイムスタンプが開始時間以降のレコードと終了時間より前のレコードを変換します。

-s オプションを省略すると，データソースのタイムスタンプが終了時間より前のすべてのレコードが変換されます。

-e *変換対象の終了時間*

変換するバイナリファイルのレコードの終了時間を *YYYYmmddHHMMSS*（GMT）形式で指定します。hsdpcliconv コマンドは，データソースのタイムスタンプが開始時間以降のレコードと終了時間より前のレコードを変換します。

-e オプションを省略すると，データソースのタイムスタンプが終了時間より後のすべてのレコードが変換されます。

-s 20140610100000 -e 20140610100100 と指定すると，コマンドは，2014/06/10 10:00:00.000 から 2014/06/10 10:00:59.999 までのレコードをすべて変換します。

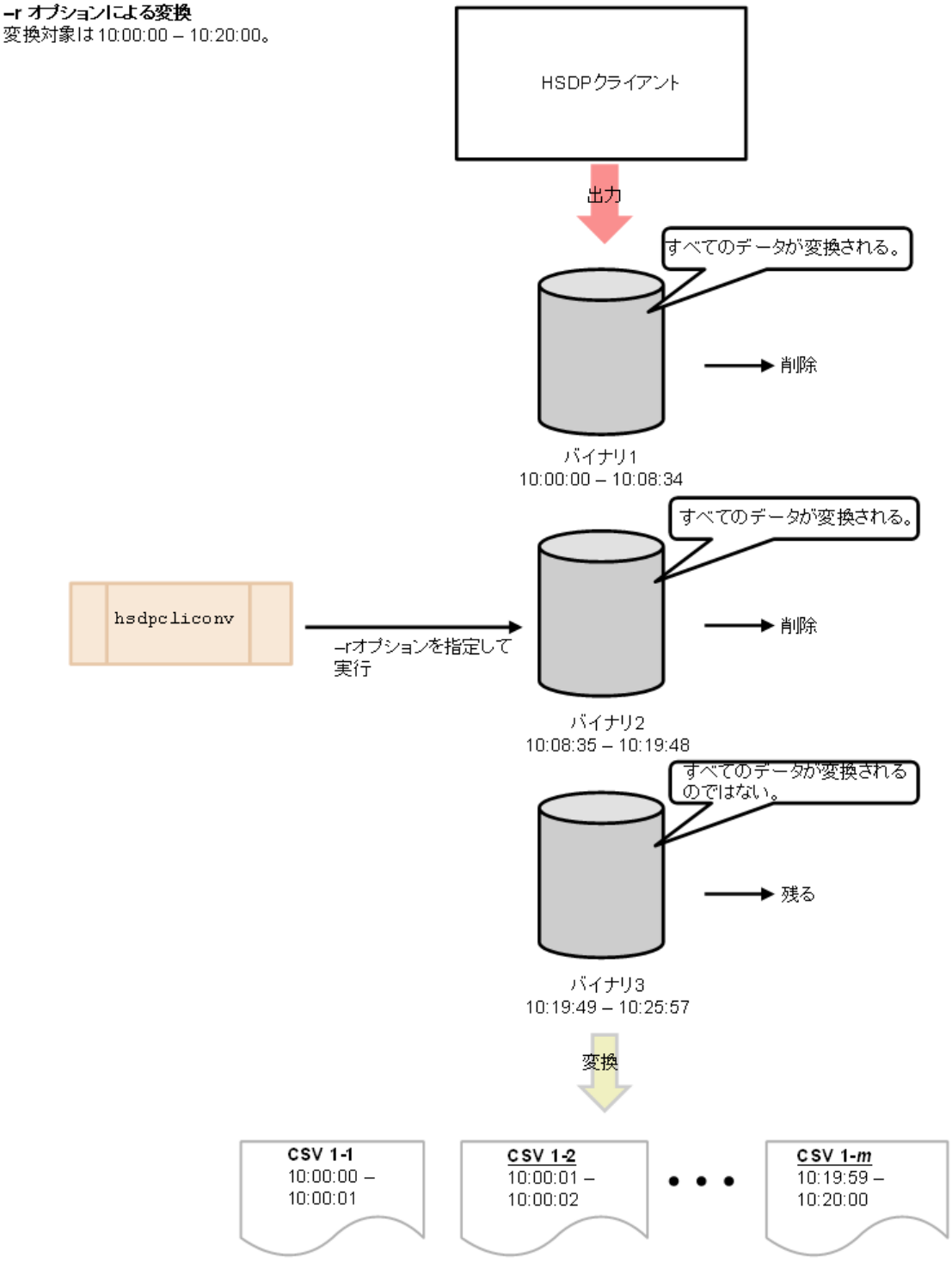
-s オプションと-e オプションを省略すると，指定したディレクトリのすべてのレコードが変換されます。

-r

ファイルの変換後，バイナリファイルを削除します。

特定のバイナリファイルのすべてのデータが変換されないように，-s オプションまたは-e オプションを指定すると，そのバイナリファイルは削除されません。このようなファイルを削除するには，hsdpclirm コマンドを使用します。次の図に，-r オプションを指定した変換例を示します。

図 16-1　-r オプションを指定した変換例



-d *変換定義ファイル*

変換定義ファイルの絶対パスを指定するか，コマンド実行時のカレントディレクトリからの相対パスを指定します。1 から512 バイトでパスを指定します。変換定義ファイルは，ユーザーが作成するファイルで，ファイルフォーマット変換のパラメーターを定義します。

変換定義ファイルの詳細については，「11.6.3　変換定義ファイル」を参照してください。

バイナリファイルが格納されているディレクトリ

変換されるバイナリファイルが含まれているディレクトリの絶対パスを指定するか，コマンド実行時のカレントディレクトリからの相対パスを指定します。1 から512 バイトを使用してパスを指定します。

注意事項

- hsdpcliconv とhsdpclirm コマンドを同時に実行しないでください。ファイルフォーマット変換でエラーが発生するおそれがあります。
- HSDP クライアントが出力するバイナリファイルは変換されません。

戻り値

値	意味	対処方法
0	バイナリファイルが正常に変換されました。	対処は不要です。
1	1 つ以上の引数が無効です。	正しい引数を指定してください。
2	変換定義ファイルのオープンに失敗しました。	変換定義ファイルのアクセス権を確認してください。 アクセス権が正しい場合は，システム管理者に連絡してください。
3	変換定義ファイルの構文が無効です。	変換定義ファイルの構文を訂正してください。
4	バイナリファイルまたは出力ファイルのオープンに失敗しました。	出力ディレクトリまたは出力ファイルのアクセス権を確認してください。
5	バイナリファイルのステータス取得に失敗しました。	システム管理者に連絡してください。
6	バイナリファイルのメモリへのマッピングに失敗しました。	システムのメモリ容量または開いているファイルの数を確認してください。
7	バイナリファイルのクローズに失敗しました。	出力ディレクトリまたは出力ファイルのアクセス権を確認してください。
8	バイナリファイルを格納しているディレクトリへのアクセスに失敗しました。	指定したディレクトリが存在するか確認してください。または，指定したディレクトリのアクセス権を確認してください。
10	メモリの割り当てに失敗しました。	システムのメモリ容量を確認してください。
11	バイナリファイルのファイルフォーマットが無効です。	システム管理者に連絡してください。

値	意味	対処方法
12	出力ファイルへの書き込みに失敗しました。	次のアクションを実行してください。 • 出力ディレクトリまたは出力ファイルのアクセス権を確認する。 • ディスク容量を確認する。
13	コマンドの処理時にエラーが発生しました。	システム管理者に連絡してください。

16.2 hsdpclirm

説明

hsdpclirm コマンドは，HSDP クライアントライブラリによって出力されたバイナリファイルを削除します。このコマンドは，hsdpcliconv コマンドによって変換されたファイルだけを削除します。hsdpcliconv コマンドの詳細については，「16.1 hsdpcliconv」を参照してください。

なお，削除条件には，次の 2 種類を指定できます。

- 時間に基づく条件
  指定した時間より前に作成されたバイナリファイルを削除します。
- 容量に基づく条件
  バイナリファイルの合計サイズが指定したサイズより小さくなるまで，バイナリファイルを削除します。

形式

```
hsdpclirm [-f] {-s 合計サイズ|-t 時間} バイナリファイルが格納されているディレクトリ
```

実行権限

HSDP クライアントの運用ユーザー

コマンドを実行するための前提条件

バイナリファイルが格納されているディレクトリがあること。

格納先ディレクトリ

hsdpcli.tar.gz が展開されたディレクトリ。

引数

-f

一部のレコードだけ変換されたバイナリファイルを含め，バイナリファイルを強制的に削除します。このオプションを省略すると，hsdpcliconv コマンドで変換されたバイナリファイルだけが削除されます。

-s *合計サイズ*

容量に基づく条件の場合に，格納されるバイナリファイルの最大サイズをバイトで指定します。バイナリファイルは，データソースのタイムスタンプが古いファイルから順に削除されます。

パラメーターに，K または k（× 1,024），M または m（× $1{,}024^2$），G または g（× $1{,}024^3$），T または t（× $1{,}024^4$）を指定できます。値の範囲は，0 から 10TB（10,995,116,277,760 バイト）です。

`-t` *時間*

時間に基づく条件の場合に，*YYYYmmddHHMMSS* の形式で時間を指定します。データソースのタイムスタンプが指定した時間より前のバイナリファイルが削除されます。

バイナリファイルが格納されているディレクトリ

変換されるバイナリファイルが含まれているディレクトリの絶対パスを指定するか，コマンド実行時の現在のパスからの相対パスを指定します。1 から512 バイトを使用してパスを指定します。

注意事項

- hsdpcliconv コマンドとhsdpclirm コマンドを同時に実行しないでください。ファイルフォーマット変換でエラーが発生するおそれがあります。

戻り値

値	意味	対処方法
0	バイナリファイルが正常に変換されました。	対処は不要です。
1	1 つ以上の引数が無効です。	正しい引数を指定してください。
4	バイナリファイルのオープンに失敗しました。	出力ディレクトリまたは出力ファイルのアクセス権を確認してください。
5	バイナリファイルのステータス取得に失敗しました。	システム管理者に連絡してください。
7	バイナリファイルのクローズに失敗しました。	出力ディレクトリまたは出力ファイルのアクセス権を確認してください。
8	バイナリファイルを格納しているディレクトリへのアクセスに失敗しました。	指定したディレクトリが存在するか確認してください。または，指定したディレクトリのアクセス権を確認してください。
10	メモリの割り当てに失敗しました。	システムのメモリ容量を確認してください。

16.3 hsdpcqldebug

説明

hsdpcqldebug コマンドは，クエリの検証に使用します。

形式

```
hsdpcqldebug
   {クエリグループ名 -i 入力ディレクトリ -o 出力ディレクトリ
   | -help}
```

実行権限

運用ユーザー

コマンドを実行するための前提条件

- 開発者は次のファイルを準備しておくこと。

  運用ディレクトリ/conf/system_config.properties

  運用ディレクトリ/conf/jvm_options.cfg

- コマンドで使用するポート番号は，システムコンフィグプロパティファイル（system_config.properties）で指定すること。システムコンフィグプロパティファイル（system_config.properties）でポート番号を指定する方法は次のとおり。

  rmi.serverPort=*ポート番号*

  > **メモ**
  >
  > ポート番号を指定しない場合は，デフォルトの20400 が使用されます。

- 対象の SDP サーバは，事前に停止しておくこと。

格納先ディレクトリ

運用ディレクトリ/bin/

引数

クエリグループ名

登録するグループの名前を指定します。

-i *入力ディレクトリ*

入力データのファイルがあるディレクトリのパスを指定します。パスは，絶対パスで指定するか，運用ディレクトリからの相対パスで指定します。

-o *出力ディレクトリ*

出力結果が保存されているファイルのディレクトリのパスを指定します。パスは，絶対パスで指定するか，運用ディレクトリからの相対パスで指定します。

-help

hsdpcqldebug コマンドの使用方法についての情報を表示します。

戻り値

値	意味
0	クエリの検証が正常に完了しました。
1	クエリの検証時にエラーが発生しました。

16.4 hsdpsdkversion

説明

このコマンドは，現在インストールされている Hitachi Streaming Data Platform software development kit の製品バージョンを*V1V2.R1.R2-MM* の形式で表示します。

- *V1V2*：バージョン番号（*V1* が0 の場合，*V1* は省略されます）。
- *R1*：リリース番号。
- *R2*：改訂番号。
- *MM*：メンテナンスレベルまたはサービスパック番号（*MM* は省略できません）。

形式

```
hsdpsdkversion [-help]
```

実行権限

運用ユーザー

コマンドを実行するための前提条件

なし。

格納先ディレクトリ

/opt/hitachi/hsdp/sdk/util/

引数

-help

コマンドの使用方法についての情報を表示します。このオプションを指定すると，コマンドは，現在インストールされているバージョンを表示しないで終了します。

注意事項

なし。

戻り値

値	意味
0	コマンドは正常に終了しました。
1	コマンドはエラーで終了しました。

株式会社 日立製作所
〒 100-8280　東京都千代田区丸の内一丁目 6 番 6 号