### クエリコマンドの実装

本節では、[SampleDriverQueryCommand.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleDriverQueryCommand.java) および [SampleSubnetGroupsCommand.java](https://github.com/logpresso/logpresso-app-examples/blob/main/logpresso-sample-app/src/main/java/com/logpresso/sonar/sample/query/SampleSubnetGroupsCommand.java) のコードを例に、クエリコマンドの実装方法について説明します。

#### ドライバーコマンド

SampleDriverQueryCommand クラスは DriverQueryCommand クラスを継承し、FieldOrdering インターフェースを実装しています。

public abstract class SampleDriverQueryCommand extends DriverQueryCommand implements FieldOrdering { protected SampleParams params; public SampleDriverQueryCommand(SampleParams params) { this.params = params; } protected abstract void run(ConnectProfile profile) throws IOException; @Override public void run() { for (ConnectProfile profile : params.getProfiles()) { if (isCancelRequested()) return; try { run(profile); } catch (Throwable t) { String msg = String.format("%s (profile %s) error - %s", getName(), profile.getName(), t.getMessage()); throw new IllegalStateException(msg, t); } } }

上記のように run() を実装し、スレッドを直接実行するコマンドを「ドライバーコマンド」と呼びます。1つのクエリ内に2つ以上のドライバーコマンドを含めることはできません。ドライバーコマンドは常に最初に配置され、パイプで接続された他のコマンドは、ドライバーコマンドのスレッドから onPush() が呼び出されて実行されます。join や union などのサブクエリでは、追加のドライバーコマンドを使用することが可能です。

クエリコマンドでは、キャンセル処理の実装漏れに注意が必要です。ループ処理内では isCancelRequested() を呼び出し、ユーザーまたはシステムによるキャンセル要求があるかを継続的に確認し、キャンセルされた場合は即座に処理を中断する必要があります。

コンストラクタの引数として渡される SampleParams オブジェクトは、ConnectProfileParams を継承したクラスであり、接続プロファイル（接続プロファイル）オブジェクトのリストを取得できます。

public String toString() { String s = getName(); if (params.getName() != null) s += Strings.doubleQuote(params.getName()); return s;}

すべてのクエリコマンド（QueryCommand）は、必ず toString() を実装しなければなりません。クエリコマンドの文字列表現は、クエリパーサーが再度パース可能な正規化された形式でコマンド構文を返す必要があります。これは、分散クエリ実行計画を生成する際に、クエリコマンドオブジェクトを再びクエリ文字列へ変換する処理が存在するためです。この実装が抜けていると、実行計画が正しく表示されず、分散クエリが失敗する可能性があります。

#### クエリコマンドの実装

SampleSubnetGroupsCommand は SampleDriverQueryCommand を継承し、以下のメソッドを実装します。

* getName(): クエリコマンドの名称
* getFieldOrder(): フィールド表示順。クエリ結果にフィールドが存在しない場合は、そのフィールドは無視されます。ここでは出力可能なすべてのフィールド名集合について完全なフィールド順序を定義すれば十分です。画面上では、クエリコマンドの順序上 FieldOrdering インターフェースを実装する最後のコマンドのフィールド出力順が適用されます。
* run(): ドライバーコマンドのスレッドで実行する機能を実装します。

サンプルコードでは RestApiClient クラスを利用してREST APIを呼び出し、pushPipe() メソッドを用いてコマンドの結果を出力します。pushPipe() の引数には Row クラスを使用します。これは Map インターフェースの実装であり、可変的なキー／値ペアを渡すことができます。ただし、値には以下の型のみが許容されます。

|  |  |  |
| --- | --- | --- |
| Java型 | ログプレッソ型 | 説明 |
| java.lang.String | string | 文字列 |
| java.lang.Short | short | 16ビット整数 |
| java.lang.Integer | int | 32ビット整数 |
| java.lang.Long | long | 64ビット整数 |
| java.lang.Float | float | 32ビット浮動小数点 |
| java.lang.Double | double | 64ビット浮動小数点 |
| java.lang.Boolean | bool | ブール値 |
| java.util.Date | date | 日付および時刻 |
| java.net.Inet4Address | ipv4 | IPv4アドレス |
| java.net.Inet6Address | ipv6 | IPv6アドレス |
| java.util.List | list | リスト |
| java.util.Map | map | マップ（キーは文字列型のみ許容） |
| byte | binary | バイト配列 |

1秒あたり数十万件以上の処理性能が求められる場合は、pushPipe(VectorizedRowBatch) メソッドを利用してデータを出力することを推奨します。VectorizedRowBatch の値には、以下のベクター型を使用できます。

|  |  |  |
| --- | --- | --- |
| ベクター型 | ログプレッソ型 | 説明 |
| org.araqne.log.api.ObjectVector | object | 文字列などすべての型を許容 |
| org.araqne.log.api.IntVector | int | 32ビット整数配列を最適化して処理 |
| org.araqne.log.api.LongVector | long | 64ビット整数配列を最適化して処理 |
| org.araqne.log.api.DoubleVector | double | 64ビット浮動小数点配列を最適化して処理 |
| org.araqne.log.api.BooleanVector | bool | ブール値配列を最適化して処理 |

約1000件単位でレコードをベクタライズしてバッチ処理することで、pushPipe() 関数呼び出しのオーバーヘッドが1000分の1に減少し、ロック競合によるパフォーマンス低下が最小化され、ガベージコレクションも減少するため、高いパフォーマンスを実現できます。