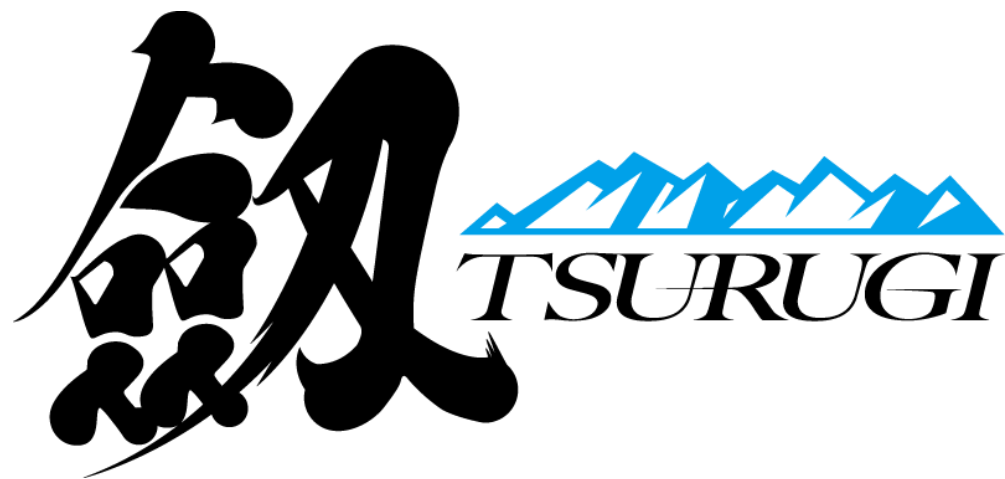


次世代RDB劔“Tsurugi” にアクセスするJavaライブラリー・ツール

Tsurugiを使う際の留意点



2024年6月16日

目次

1. 自己紹介
2. Tsurugiの概要
3. Tsubakuro (Java通信ライブラリー)
4. Iceaxe (Javaライブラリー)
5. PostgreSQL FDW (JDBC)
6. Tsurugi SQLコンソール (tgsql)
7. Embulkプラグイン
8. その他クライアントツール

自己紹介

- 株式会社ノーチラス・テクノロジーズ
 - 菱田 真人
- Tsurugi開発での担当
 - Iceaxe (Javaライブラリー)
 - Tsurugi SQLコンソール (tgsql)
 - 原価計算ベンチマーク
- 個人的に、Javaに関するウェブページを制作



Tsurugiの概要

Tsurugi (劔) とは

■ OSSのRDBMS

- <https://www.tsurugidb.com/>
- <https://github.com/project-tsurugi/tsurugidb>
- NEDOの支援を受けて、各大学研究機関・NEC・ノーチラス各社が協力して開発
- Apache License 2.0

■ インメモリーDB

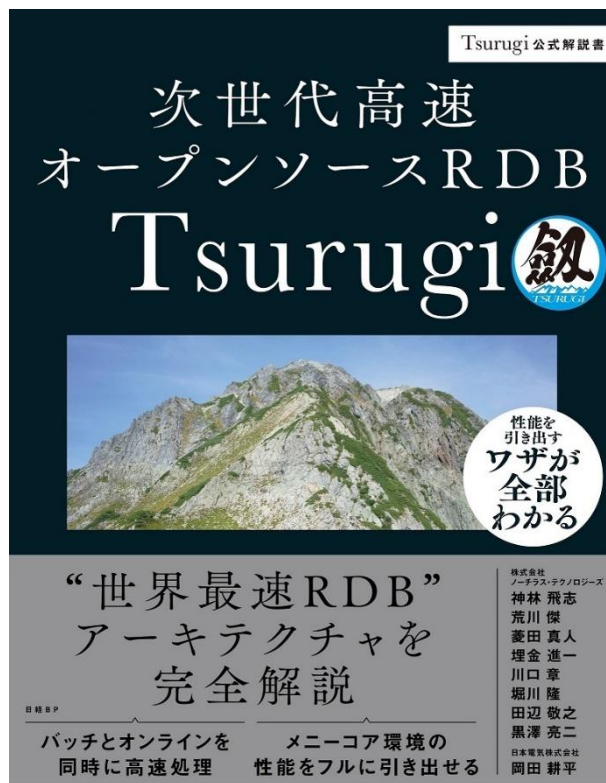
- 現代的なメニーコア・大容量メモリーのハードウェアが対象
- 現時点では単ノード。将来的にはレプリケーションも

■ トランザクション分離レベルはSERIALIZABLE

- Tsurugiを使う際に従来のRDBMSとは異なる考慮が必要

Tsurugiの書籍

- 『次世代高速オープンソースRDB Tsurugi』
 - 2023/10/5のTsurugi 1.0.0-BETA1公開と同時に、Tsurugiを解説した書籍が発売



TsurugiはインメモリーDB

■ インメモリーDB

- データは全てメモリー上に置く
 - 例外的に、BLOB（未実装）はファイルとしてディスクに置く予定
 - スピルアウト機能は（まだ）無い
- 永続化としてディスクに書く
 - Tsurugiサーバー再起動時に永続化されたデータをディスクから読むが、それ以外に稼働中にディスクから読むことは無い

■ ハードウェア環境

- メニーコア（30コア以上、できれば100コア以上）
 - 少ないコア数（10コア以下）では、PostgreSQLの方が速そう
 - ただしPostgreSQLはバッチ処理とオンライン処理の混在は無理
- 大容量メモリー（500GB以上）
- 少量データでちょっと試すだけなら、普通のサーバーでも動く
 - Dockerイメージも提供されている

TsurugiはRDBMS

■ SQL

- 基本的にANSI準拠・PostgreSQL似
 - 現時点ではまだ未実装の機能（構文・関数）は多い
 - <https://github.com/project-tsurugi/tsurugidb/blob/master/docs/sql-features.md>

■ ACIDトランザクション

- トランザクション分離レベルはSERIALIZABLE
- Tsurugi独自のトランザクション理論（epoch・MVCCベース）
 - 書籍の第3部

■ 内部はトランザクション機能付きKVS

- 実体はMasstree（キー順に並べるツリー構造）
 - JavaのTreeMap<Pk, Record>のようなイメージ
- 他のRDBMSだと、プライマリキーに対して暗黙にユニークインデックスが作られたりするが、Tsurugiでは自動的にプライマリキーで並ぶ

(KVS関連) Tsurugiのinsert文

■ insert

```
insert into tb values(1, 11)
```

- KVSのget（一意制約違反の判定）+put

■ insert or replace（いわゆるupsert）

```
insert or replace into tb values(1, 11)
```

- 既存データの有無に関わらず書く
- KVSのput（のみ）に相当するため、単なるinsertより速い

■ insert if not exists

```
insert if not exists into tb values(1, 11)
```

- データが無いときだけ書く
 - データが有る場合は何もしない（一意制約違反にならない）
 - JavaのMap#putIfAbsent()のようなイメージ

トランザクション分離レベル

- <https://ja.wikipedia.org/wiki/トランザクション分離レベル> によると、「どれほどの一貫性、正確性で実行するかを4段階で定義したもの」
 1. READ UNCOMMITTED
 - PostgreSQL, Oracle, SQL Serverのデフォルト
 2. READ COMMITTED
 - MySQLのデフォルト
 3. REPEATABLE READ
 - Tsurugi
 4. SERIALIZABLE

SERIALIZABLEとは

■ SERIALIZABLE

- 2つのトランザクションT1とT2を同時に（並列に）実行した時、順番に（T1→T2あるいはT2→T1の順で）実行したのと同じ結果になる
- TsurugiはSERIALIZABLE（のみ）

■ 例えばREAD COMMITTEDは

- コミットされたデータを読む
- トランザクション実行中に他トランザクションがデータを更新してコミットしたら、そのデータが読めてしまう

SERIALIZABLE用のプログラミング

- 同時に実行されている複数のトランザクションがSERIALIZABLEの条件を満たせない場合、（SQL実行時や）コミット時にシリアライゼーションエラーが発生する（アボートする）
- シリアライゼーションエラーが発生したら、トランザクション（トランザクション内で実行されたSQL）を再実行（リトライ）する
 - リトライは（DBMSが行うのではなく、）アプリケーション側で行う必要がある
 - いわば楽観ロック

順番に実行する例(T1→T2の順)

■ 初期データとしてinsert into tb (v) values(10)

	T1	T2
1	begin;	
2	select v from tb; → 10	
3	select v from tb; → 10	
4	commit;	
5		begin;
6		update tb set v = 20;
7		commit;

- T1のselectは、2回とも同じ値を読んでいる

順番に実行する例(T2→T1の順)

■ 初期データとしてinsert into tb (v) values(10)

	T1	T2
1		begin;
2		update tb set v = 20;
3		commit;
4	begin;	
5	select v from tb; → 20	
6	select v from tb; → 20	
7	commit;	

— T1のselectは、2回とも同じ値を読んでいる

READ COMMITTEDの例

■ 初期データとしてinsert into tb (v) values(10)

	T1	T2
1	begin;	
2	select v from tb; → 10	
3		begin;
4		update tb set v = 20;
5		commit;
6	select v from tb; → 20	
7	commit;	

- T1とT2を並列に実行すると、このような状態が起こり得る。
- T1でselectした値が1回目と2回目で異なる。

READ COMMITTED・select for updateの例

■ 初期データとしてinsert into tb (v) values(10)

	T1	T2
1	begin;	
2	select v from tb for update; → 10	
3		begin;
4		update tb set v = 20; → 待つ
5	select v from tb; → 10	
6	commit;	
7		commit;

— ロックをとることで、T1のselectは2回とも同じ値を読んでいる

SERIALIZABLE (TsurugiのOCC) の例

	T ₁ (OCC)	T ₂ (OCC)
1	begin transaction;	
2	select v from tb; → 10	
3		begin transaction;
4		update tb set v = 20;
5		commit;
6	select v from tb; → 20	
7	commit; → シリアライゼーションエラー	
8	begin transaction; リトライ	
9	select v from tb; → 20	
10	select v from tb; → 20	
11	commit;	

コミット時にも
(通信エラー以外の) エラーが
発生しうる

selectのみのトランザクションで
もシリアライゼーションエラー
が発生することがある

シリアライゼーションエラーが
発生したら、トランザクション
全体を再実行する

SERIALIZABLE (TsurugiのLTX) の例

	T ₁ (LTX)	T ₂ (LTX)
1	begin long transaction;	
2	select v from tb; → 10	
3		begin long transaction write preserve tb;
4		update tb set v = 20;
5		commit;
6	select v from tb; → 10	
7	commit;	

- LTXではトランザクション開始時点の値を読む仕様なので、T₁のselectは2回とも同じ値（T₂の更新前）を読んでいる

Tsurugiのトランザクションの特徴

■ OCC・LTX

- TsurugiではSERIALIZABLEを実現する為の内部プロトコルが2種類あり、どれを使うかをトランザクション開始時に指定する。
 - すなわち、トランザクションを意識的に開始する必要がある。

■ 排他方法（ロックのコスト）

- シリアライゼーションエラー発生時にリトライする楽観ロック
 - 一方、select for updateは悲観ロック。
 - 悲観ロックはコストが大きいため、トランザクションの競合が少ないなら、毎回ロックするよりエラー時にリトライする方が総合的なコストは低い。
 - リトライする方式は、select for updateと異なり、デッドロックしない。

■ コミット

- selectのみのトランザクションでも必ずコミットして、シリアライゼーションエラーが発生しないことを確認する必要がある。

Tsurugiのトランザクション種別（1）

- TsurugiでSERIALIZABLEを実現する為の内部プロトコルはOCC・LTXの2種類
- Tsurugiでトランザクション開始時にユーザーが指定するトランザクション種別はOCC・LTX・RTXの3種類
 - トランザクション毎に指定する（異なる種類を指定可能）
 - DB全体の設定で一括して切り替えるようなものではない

Tsurugiのトランザクション種別（2）

- OCC（short transaction・optimistic concurrency control）
 - 実行時間が短いトランザクション（数十ミリ秒・いわゆるオンライン処理向け）
 - SQL実行時点の最新データを読む（READ COMMITTEDと同様）
 - 他トランザクションとの競合時にシリアライゼーションエラーになりやすい。OCC同士では先にコミットした方が優先される
- LTX（long transaction）
 - 実行時間が長いトランザクション（いわゆるバッチ処理向け）
 - トランザクション開始時点のデータを読む
 - LTX同士では先に始まった方が優先度が高い。一番最初に実行を開始したLTXはシリアライゼーションエラーにならない
- RTX（read only transaction）
 - 読み取り専用トランザクション（内部プロトコルではLTXの一種）
 - トランザクション開始以前のデータを読む
 - 他トランザクション終了直後にそのデータが読めるとは限らない（タイムラグがある）
 - 他トランザクションと競合しない

TsurugiのLTX同士の競合の例

	T ₁ (LTX)	T ₂ (LTX)
1	begin long transaction write preserve tb;	
2	update tb set v = 11;	
3		begin long transaction write preserve tb;
4		update tb set v = 20;
5		commit; →待ちに入る
6	commit;	
7		commit終了→シリアライゼーションエラー→リトライ

- T2が先にコミットに入っても、優先度が高いLTX（T1）が同じデータを更新するかもしれないので、T1が終わるのを待つ

TsurugiのRTXの例

	T ₁ (RTX)	T ₂ (LTX)
1	begin read only;	
2	select v from tb; → 10	
3		begin long transaction write preserve tb;
4		update tb set v = 20;
5		commit;
6	select v from tb; → 10	
7	commit;	

- T₁のselectは、2回とも同じ値（T₂の更新前）を読んでいる
- T₂は待たずにコミット成功する

Tsurugi SQLコンソール (tgsql) で トランザクション種別を指定する例

■ OCC

- begin transaction;
- begin;

■ LTX

- begin long transaction write preserve テーブル1,
テーブル2, ...;
 - LTXでは、更新対象 (insert/update/deleteするテーブル) を全て
write preserveで指定する必要がある。

■ RTX

- begin transaction read only deferrable;
- begin read only;

コミットオプション

- Tsurugiでは、コミット時に、Tsurugiサーバー内でどこまで処理したら制御が戻るかを指定できる
 - DEFAULT
 - Tsurugiサーバー側の設定（構成定義ファイルの内容）に従う
 - ACCEPTED
 - コミットを受け付けた（トランザクションを正当なスケジュールとして受理した）
 - AVAILABLE
 - コミットしたデータが他トランザクションから見えるようになった
 - STORED
 - コミットしたデータがローカルディスクに書かれた（永続化された）
 - PROPAGATED（未実装）
 - コミットしたデータが分散システムの適切な全てのノードに伝播された

余談：PostgreSQLのSERIALIZABLE

- PostgreSQLのデフォルトのトランザクション分離レベルはREAD COMMITTEDだが、SERIALIZABLEに変更することが出来る。
- 60トランザクション（同一テーブルにdelete-insertするが、キーは重複しない）を同時に実行したら、1つコミットした時点で他の59個がシリアライゼーションエラー。59トランザクションを再実行すると、また1つコミットして58個がシリアライゼーションエラー。これを繰り返して、結局1トランザクションずつしか完了しなかった。（見事に直列実行）
 - Tsurugiなら並列に実行可能
 - PostgreSQLでもREAD COMMITTEDなら並列に実行可能



Java通信ライブラリー

Tsubakuro

Tsubakuro (Java通信ライブラリー) (書籍の第11章)

■ JavaでTsurugiにアクセスする為の通信ライブラリー

– 基礎的な通信ライブラリー (Java11)

■ 様々な機能を持っている

- SQL実行 (JDBCではない)
- KVSアクセス (まだ非公開)
- データストア (テーブル単位のバックアップ・リストア)
- ...

■ 非同期API (使いやすさより実行効率優先)

– ソースコード

■ <https://github.com/project-tsurugi/tsubakuro>

– jarファイル (Mavenセントラルリポジトリ)

■ <https://central.sonatype.com/search?q=tsubakuro>

他のJavaライブラリーやツールは
Tsubakuroを使っている

Tsubakuro (Java通信ライブラリー) の例

```
try (var session = SessionBuilder.connect("tcp://localhost:12345").create();
     var sqlClient = SqlClient.attach(session)) {
    var option = TransactionOption.newBuilder()
        .setType(TransactionType.SHORT).build();
    try (var transaction = sqlClient.createTransaction(option).await()) {
        transaction.executeStatement("update tb set v = 20").await();
        transaction.commit().await();
    }
}
```

1. Sessionを生成する
2. SqlClient等のクライアントを生成する
3. createTransactionやexecuteStatementやcommitはFutureResponseを返す

エンドポイント

- エンドポイント（接続先）をURIの形で指定する
- TsurugiのDBサーバーに接続する方法は、現時点ではIPC接続とTCP接続の2種類
 - IPC接続
 - Linuxの共有メモリーを介してデータを送受信する方式
 - TCP接続より高速だが、Tsurugiサーバーと同一のマシン上でしか使用できない
 - 例) ipc:tsurugi
 - TCP接続
 - TCP/IPのソケット通信を使ってデータを送受信する方式
 - IPC接続より低速だが、どのマシンからでも接続可能
 - 例) tcp://localhost:12345

認証

- 通常のRDBMSでは、DBに接続する際にユーザーID・パスワードを入力するといった認証を行う。
- 現時点のTsurugiでは、認証が実装されていない。
 - 認証方法はいくつか用意されているが、どの認証方法を用いても接続できる。
 - 現時点では、基本的に無認証（no auth・NullCredential）を使用すればよい。

Tsubakuroのinsertの例

```
var sql = "insert into customer values(:id, :name, :age)";
var placeholder = List.of(
    Placeholders.of("id", AtomType.INT8),
    Placeholders.of("name", AtomType.CHARACTER),
    Placeholders.of("age", AtomType.INT4));

try (var ps = sqlClient.prepare(sql, placeholder).await()) {
    var parameter = List.of(
        Parameters.of("id", 1L),
        Parameters.of("name", "tsurugi"),
        Parameters.of("age", 0));
    transaction.executeStatement(ps, parameter).await();
}

transaction.commit().await();
```

Tsubakuroのselectの例

```
var sql = "select * from customer";
try (var rs = transaction.executeQuery(sql).await()) {
    var columnList = rs.getMetadata().getColumns();
    while (rs.nextRow()) {
        for (int i = 0; rs.nextColumn(); i++) {
            var column = columnList.get(i);
            String name = column.getName();
            var value = switch (column.getAtomType()) {
                case INT4 -> rs.fetchInt4Value();
                case INT8 -> rs.fetchInt8Value();
                case CHARACTER -> rs.fetchCharacterValue();
                default -> throw new AssertionError(column);
            };
            System.out.printf("%s=%s\n", name, value);
        }
    }
}
transaction.commit().await();
```



Javaライブラリー

Iceaxe

Iceaxe (Javaライブラリー) (書籍の第8章)

- TsurugiでSQLを実行するJavaライブラリー
 - Tsubakuroをラップして、便利な機能を提供している
 - 同期API
 - SQL実行機能のみ (将来的にはKVSアクセスにも対応する予定)
 - IceaxeもJDBCではない
 - シリアライゼーションエラー発生時にリトライする機能を持つ
 - ソースコード
 - <https://github.com/project-tsurugi/iceaxe>
 - jarファイル (Mavenセントラルリポジトリ)
 - <https://central.sonatype.com/search?q=iceaxe-core>

```
implementation 'com.tsurugidb.iceaxe:iceaxe-core:1.3.0'
```

Iceaxeの例

```
var connector = TsurugiConnector.of("tcp://localhost:12345");
try (var session = connector.createSession()) {
    try (var ps = session.createStatement("update tb set v = 20")) {
        var setting = TgTmSetting.ofAlways(TgTxOption.ofOCC());
        var tm = session.createTransactionManager(setting);
        tm.execute(transaction -> {
            transaction.executeAndGetCount(ps);
        });
    }
}
```

- TransactionMangerを生成する
 - その中でTransactionインスタンスが生成される
 - tm.execute()に渡すラムダ式に、トランザクション1回分の処理を記述する
 - シリアライゼーションエラーが発生したときに再度実行される

Iceaxeのinsertの例

```
var sql = "insert into customer values(:id, :name, :age)";
var mapping = TgParameterMapping.of(CustomerEntity.class)
    .addLong("id", CustomerEntity::getId)
    .addString("name", CustomerEntity::getName)
    .addInt("age", CustomerEntity::getAge);

try (var ps = session.createStatement(sql, mapping)) {
    tm.execute(transaction -> {
        var entity = new CustomerEntity();
        entity.setId(1);
        entity.setName("tsurugi");
        entity.setAge(0);

        transaction.executeAndGetCount(ps, entity);
    });
}
```

Iceaxeのselectの例

```
var sql = "select c_id,c_name,c_age from customer";
var mapping = TgResultMapping.of(CustomerEntity::new)
    .addLong("c_id", CustomerEntity::setId)
    .addString("c_name", CustomerEntity::setName)
    .addInt("c_age", CustomerEntity::setAge);

try (var ps = session.createQuery(sql, mapping)) {
    List<CustomerEntity> entityList = tm.execute(transaction -> {
        return transaction.executeAndGetList(ps);
    });

    for (var entity : entityList) {
        System.out.println(entity);
    }
}
```

IceaxeとJDBCのクラスの対比

Iceaxe	Tsubakuro	JDBC
TsurugiConnector	Connector	(Driver)
TsurugiSession	Session	Connection
TsurugiTransaction	Transaction	Connection
TsurugiSqlQuery TsurugiSqlStatement		Statement
TsurugiSqlPreparedQuery TsurugiSqlPreparedStatement	PreparedStatement	PreparedStatement
TsurugiQueryResult	ResultSet	ResultSet
TsurugiResultEntity		

- JDBCのConnectionはSessionとTransactionを兼ねている
 - Connectionは1接続1トランザクションだが、Tsurugiでは1セッションで複数トランザクションが可能

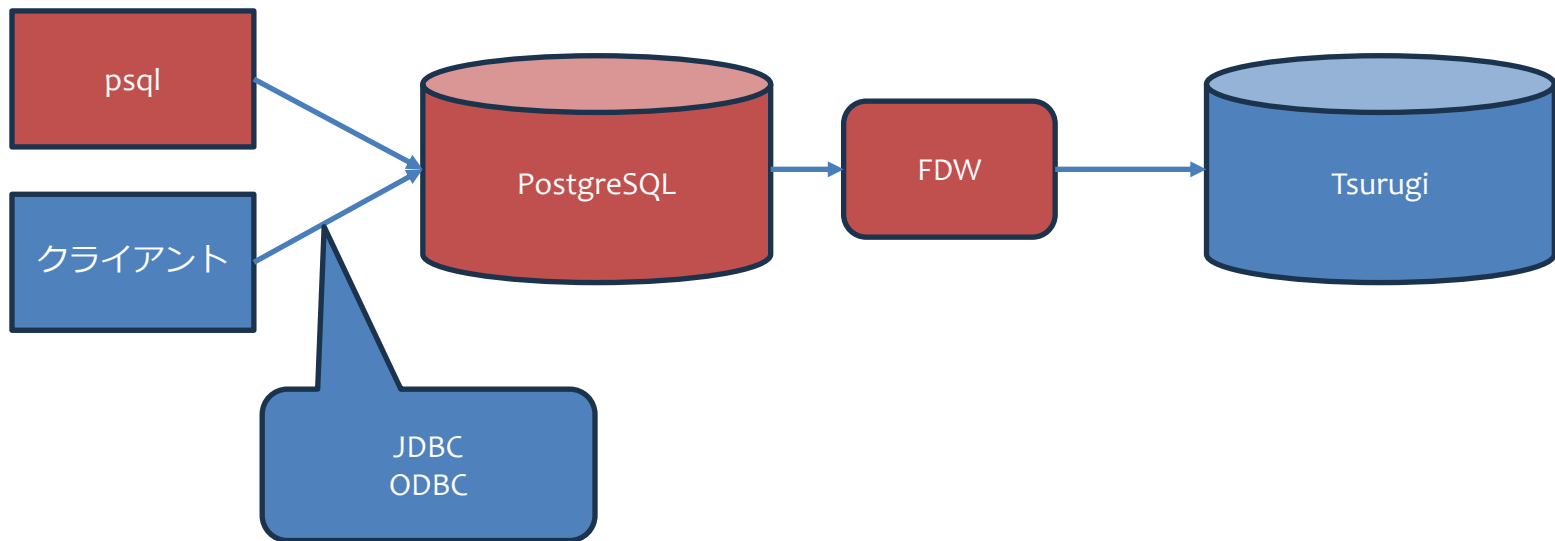


PostgreSQL (JDBC) 経由でTsurugiにアクセスする

PostgreSQL FDW

PostgreSQL FDW（書籍の第7章）

- PostgreSQL経由でTsurugiにアクセスする
 - PostgreSQLのForeign Data Wrapper機能を利用
 - PostgreSQLのテーブルを操作すると、裏でTsurugiにアクセスする
 - psqlやPostgreSQLのJDBC・ODBCが使用可能
 - 大量データを扱うのは強く非推奨



PostgreSQL FDW（書籍の第7章）

- FDWを使うにはextension・server・tablespaceを定義する
- PostgreSQL側とTsurugi側のテーブルを作成する
 - create foreign tableは、Tsurugiにテーブルを作成する

```
CREATE EXTENSION tsurugi_fdw;  
CREATE SERVER tsurugidb FOREIGN DATA WRAPPER tsurugi_fdw;  
CREATE TABLESPACE tsurugi LOCATION '~'/tsurugi';  
  
CREATE TABLE tb (  
    ~  
) TABLESPACE tsurugi;  
  
CREATE FOREIGN TABLE tb (  
    ~  
) SERVER tsurugidb;
```

PostgreSQL FDW経由のJDBCの例

- https://github.com/project-tsurugi/tsurugi_fdw/blob/master/docs/java_sample/transaction_sample.java

```
String url = "jdbc:postgresql://localhost:5432/postgres";
Connection connection = DriverManager.getConnection(url);

Statement st = connection.createStatement();
st.execute("SELECT tg_start_transaction()");

String sql = "INSERT INTO tg_sample (col, tm) VALUES (?, ?)";
PreparedStatement ps = connection.prepareStatement(sql);
ps.setInt(1, 1);
ps.setTime(2, Time.valueOf(LocalTime.now()));
ps.executeUpdate();

st.execute("SELECT tg_commit()");
```

JDBCが使用できても対応できなさそうなこと

- Tsurugi向けのJDBCが使用できたとしても、既存アプリケーション向けの単純な置き換えは難しそう
 - トランザクションオプションやコミットオプションの指定
 - select for updateでロックしない
 - 現在のTsurugiではselectにfor updateを付けるとエラー
 - シリアライゼーションエラー発生時のリトライ
 - たぶん、既存のJDBCアプリケーションはリトライの仕組みが無い
 - たぶん、コミット時に処理可能な例外（シリアライゼーションエラー）が発生する想定が無い
 - フレームワークが実施してくれると嬉しい



対話形式でSQLを実行するツール（REPL）

Tsurugi SQLコンソール（tgsql）

Tsurugi SQLコンソール (tgsql) (書籍の第6章)

- TsurugiのDBに対してSQLを実行するCLIツール
 - Tsubakuroを使っている
 - JavaなのでLinuxでもWindowsでも使用可能

```
$ $TSURUGI_HOME/bin/tgsql -c ipc:tsurugi
tgsql> begin;
transaction started. option=[
  type: OCC
]
Time: 48.87 ms
tgsql> select * from tb;
[v: INT4]
[10]
(1 row)
Time: 24.594 ms
tgsql> commit;
transaction commit(DEFAULT) finished.
Time: 7.054 ms
```

tgsqlで使用しているライブラリー

■ JCommander

- コマンドライン引数の解析（アノテーションで定義）

■ JLine

- REPLを実現するライブラリー
 - JShellでも使われている
 - 主要な端末（LinuxやWindows）に対応している
- 複数行入力（SQLは複数行にまたがる事も多い）
- 文字色・太字
- キーバインド
 - カーソルキーの上下で履歴移動、Ctrl+Rでコマンド履歴検索
 - タブキーで入力補完
 - 補完候補は自力で用意する必要あり
 - » カーソル位置に応じた補完候補生成にはパーサーも必要なため、今のtgsqlでは簡易的なもののみ実装

tgsqlのTODO

■ select結果の整形

- 現状ではList#toString()で出力している ^ ^ ;

```
tgsql> select * from customer;  
[c_id: INT8, c_name: CHARACTER, c_age: INT4]  
[1, Hello, 51]  
[2, World, 130]  
[3, Tsurugi, 0]  
(3 rows)  
Time: 9.61 ms
```

- 表形式で（いわゆる全角文字も含めて）桁数を揃えて出力できるライブラリーがあったら、教えて下さいorz



Embulk (ETLツール) でTsurugiにアクセス

Embulkプラグイン

Embulkプラグイン

- Embulkは、OSSのETLツール
 - プラグインで各種RDBMS等に対応している
- embulk-input-tsurugidb
- embulk-output-tsurugidb
 - Tsurugiの公式ツールではない
 - <https://github.com/hishidama/embulk-input-tsurugidb>
 - <https://github.com/hishidama/embulk-output-tsurugidb>
 - Embulk0.10以降
 - Embulkの公式な対応JavaバージョンはJava8だが、Java11が必要
 - configの書き方はPostgreSQLのEmbulkプラグインと互換
 - Tsurugi側で未実装な機能があるため、完全互換ではない

native libraryのロード問題

- embulk-input-tsurugidbとembulk-output-tsurugidbでIPC接続を同時に使おうとすると
`java.lang.UnsatisfiedLinkError: Native Library libtsubakuro.so already loaded in another classloader`
 - TsubakuroのIPC接続ではnative library (libtsubakuro.so) を使っている
 - Foreign Function and Memory APIがJava11にあれば...
 - Javaでは複数のクラスローダーで同一のnative libraryをロードすることが出来ない
 - Embulkではプラグイン毎にクラスローダーを用意してjarファイルを読み込む

(余談) Embulkのバージョン

■ Embulk 0.9以前

- Java8
 - Java9以降では動作しない (javax系が無い)
- Rubyに依存 (JRubyを使用)
- プラグインはruby gemsで管理

■ Embulk 0.10

- 依存関係の整理用 (Embulk 0.11への過渡期)

■ Embulk 0.11

- Java8
 - Java9以降でも動作する
- Rubyから脱却
- プラグインはMavenセントラルリポジトリで管理
 - 実行時にMavenローカルリポジトリからjarファイルを読む



Tsurugiのクライアントツール・ライブラリーの紹介

クライアントツール・ライブラリー

tgctlコマンド（書籍の第12章）

- Tsurugiサーバーの起動や停止を行うCLIツール
- 構成定義ファイルを指定する
 - Tsurugiサーバーの設定値（IPC接続のデータベース名やTCP接続のポート番号、永続化データを置くディレクトリー等）を記述する
- 例
 - `tgctl start --conf tsurugi.ini`
 - `tgctl shutdown --conf tsurugi.ini`
 - `tgctl status --conf tsurugi.ini`

Tsurugi テーブルダンプツール (tgdump)

- Tsurugiのテーブルのデータをファイルに出力するCLI ツール
 - IPC接続のみ（Tsurugiと同じサーバー上でしか使えない）
 - 現時点ではParquetとArrowに対応（csvには未対応）

```
$ bin/tgdump -c ipc:tsurugi --to /tmp/dump --profile parquet tb  
$ ls /tmp/dump/tb/  
d1711356156_0_0.parquet
```

- テーブルにロードするツールは未実装

Belayer (Web API) (書籍の第9章)

- Webインターフェース (REST API)
 - 任意のSQLを実行するものではなく、DBの運用管理が目的
 - WebAdmin (WebブラウザーによるGUIツール) と Remote CLI (クライアントのCUIツール) だけ有償
- データベースのバックアップ・リストア
- テーブルのダンプ・ロード
 - csvおよびparquetファイル
 - 大量データアクセス向け
 - Asakusa Frameworkもこの機能を使う予定

Altimeter（開発中）

■ イベントログサービス

- 各種イベントログや監査ログを検索・集計するツール
 - DB開始・終了
 - セッション開始・終了
 - トランザクション開始・終了
 - SQL実行開始・終了
- セッション・トランザクション・SQLの状態を取得できる
- 有償

クライアントC++ライブラリー（予定）

- Tsurugiサーバーにアクセスする為のクライアントライブラリーは、現在はJavaのみ提供（Tsubakuro）
- C++ライブラリーを提供予定
 - C++ライブラリーを呼ぶことが出来る言語であれば、これを使ってTsurugiサーバーにアクセスできる。
 - 非同期APIである事やselect結果の扱い方（ResultSet）はTsubakuroと同様になる予定。
 - Futureは使わないかも
 - 通信データはprotocol buffersでエンコードされているので、それをデコードするのはライブラリーを呼ぶ側が行う予定。



終わり

まとめ

まとめ

- Tsurugiはメニーコア・大容量メモリーを前提としたインメモリーのRDBMS
- Tsurugiのトランザクション分離レベルはSERIALIZABLE
 - トランザクション開始時にトランザクション種別（OCC・LTX・RTX）を指定する。
 - select for updateでの排他制御は行わない。
 - SQL実行時やコミット時にシリアライゼーションエラーが発生する可能性がある。（selectのみのトランザクションでもコミットが必要）
 - シリアライゼーションエラーが発生したら、アプリケーション側でトランザクションを再実行する。

まとめ（Java向け）

- 今のところ、TsurugiのクライアントライブラリーはJavaのみ
 - Tsubakuro・Iceaxe（JDBCではない）
 - JDBCはPostgreSQL FDW
- ゆる募
 - （tgsqlのselect結果）桁数を揃えて出力するライブラリー
 - native libraryの重複エラーを回避する方法