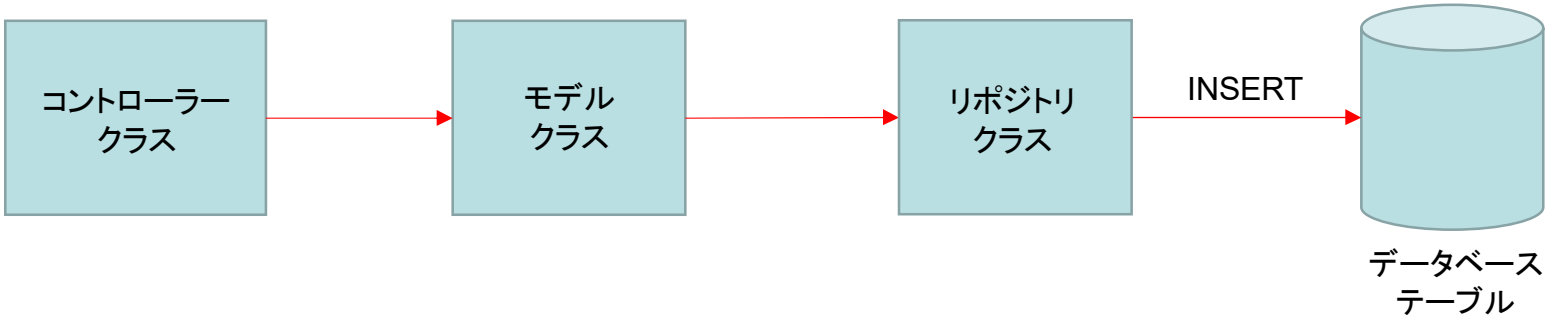
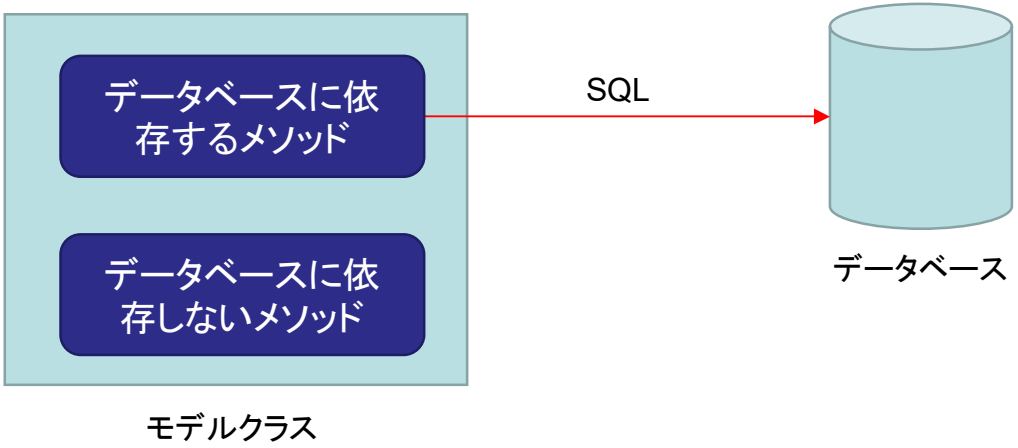
本ステップでおこなうこと

モデルクラス・リポジトリクラスを作り、簡易的なデータベース操作を行います。



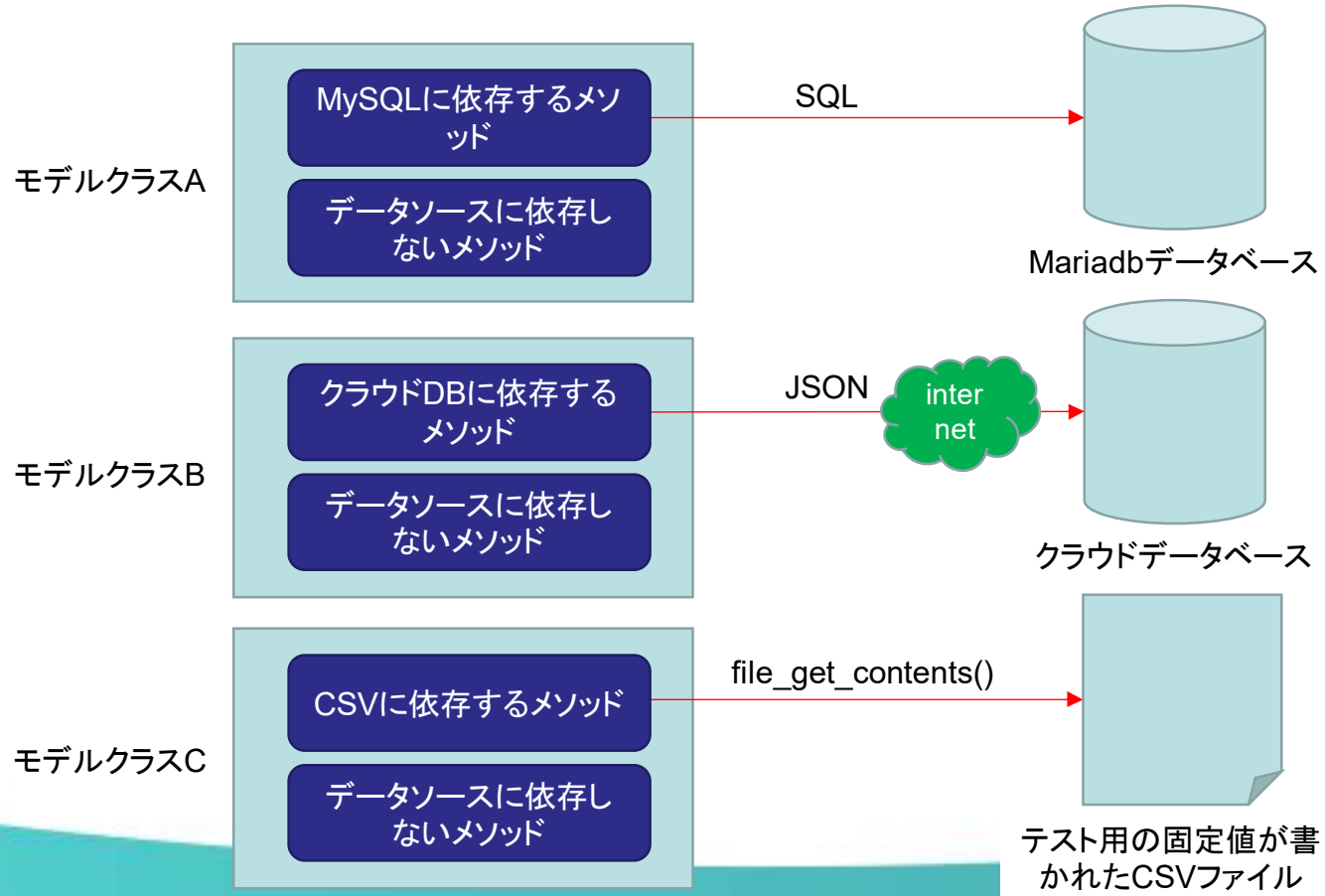
リポジトリクラスを作る理由(1)

モデルから直接データベース操作を行う例をみてみましょう。
モデルには業務ロジックを書きますが、必ずしも「業務ロジック＝データベース操作」とは限りません。



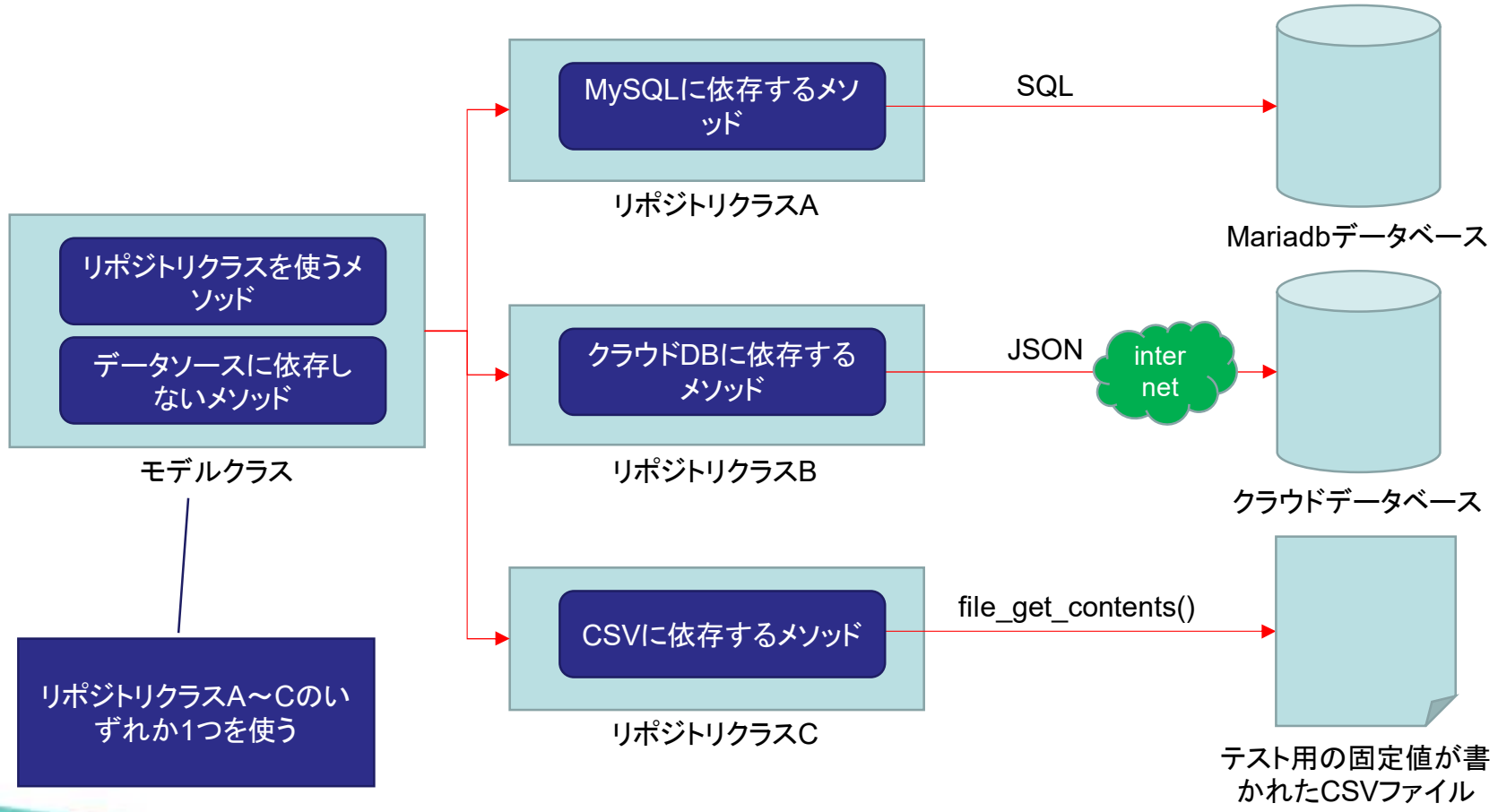
リポジトリクラスを作る理由(2)

もし、将来、データソースそのものが切り替わったときは、どうしたらいいでしょうか？そのときは、モデルクラスを作り直すことになります。このとき、「データベースに依存しないメソッド」は同じものを使うことになります。



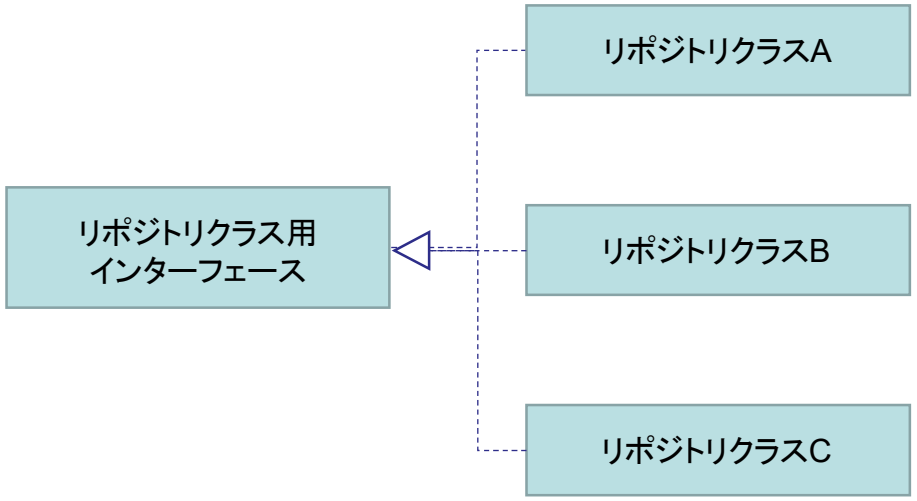
リポジトリクラスを作る理由(3)

そこで、データソースを操作する役割だけを切り出して、リポジトリクラスに持たせる設計にします。



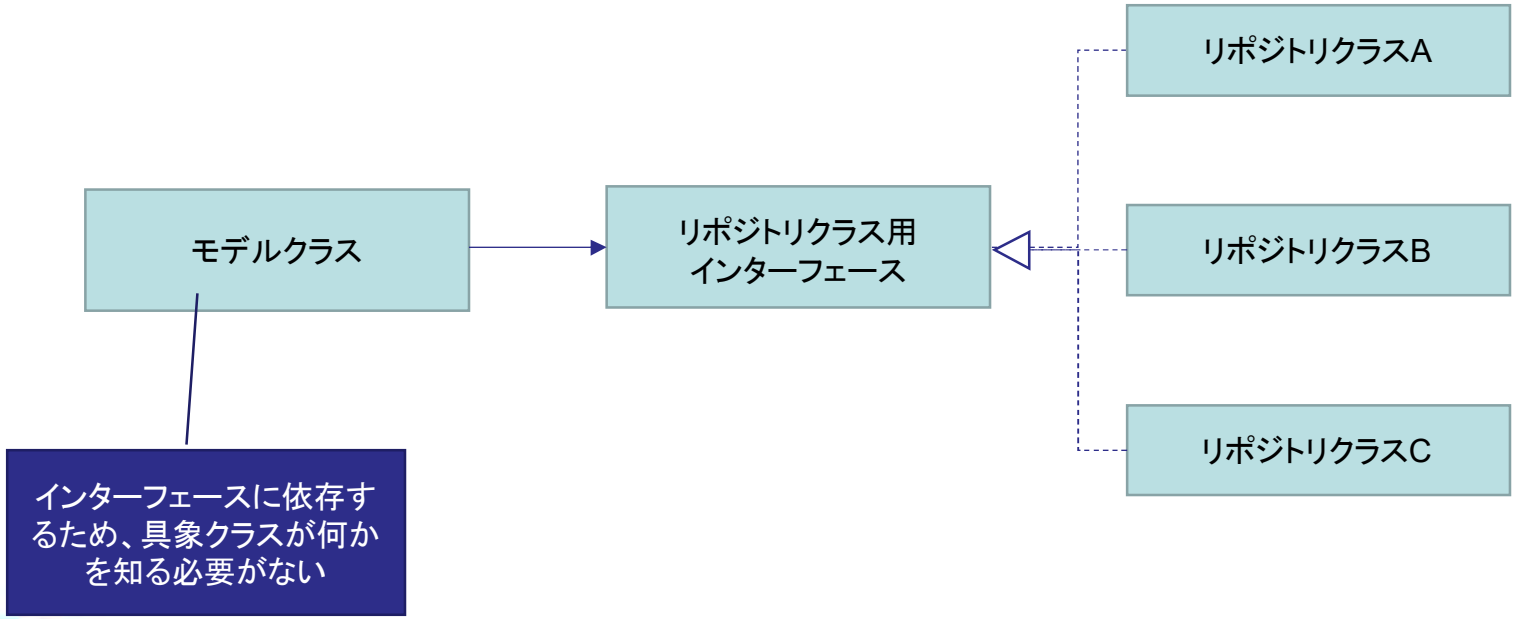
リポジトリクラスの交換可能にする(1)

リポジトリクラスを切り替えやすくするために、あらかじめ、リポジトリクラス用のインターフェースを実装しておきます。
インターフェースがあることで、リポジトリクラスA～Cが、必ず同じメソッドを持っていることが約束されます。



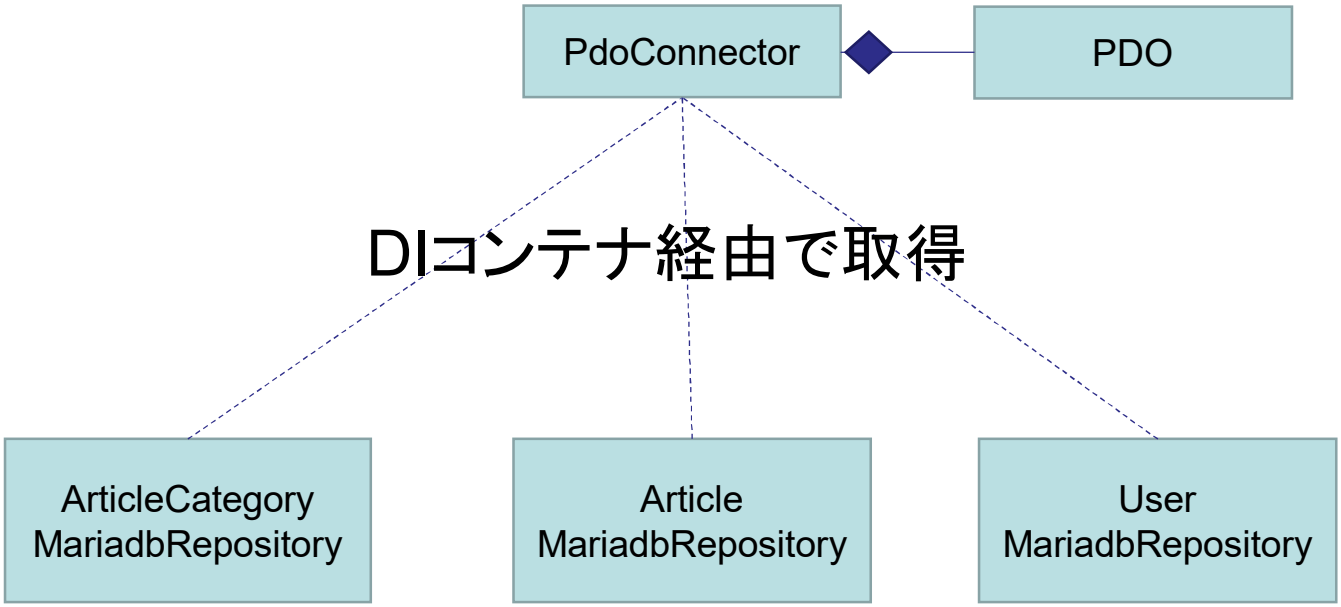
リポジトリクラスの交換可能にする(2)

次に、リポジトリクラスA～Cのいずれか1つを、フロントコントローラからDIコンテナに登録しておきます。
モデルクラスはDIコンテナからそのインスタンスを受け取りますが、インターフェースがあるため、その正体が何なのかを知る必要はありません。



データベース接続クラスを作る

データベース接続は、新たに作成するPdoConnectorクラスを使って行います。
各リポジトリクラスが、PdoConnector経由でPDOインスタンスを取得できるようにします。



データベース接続を一元化したい理由(1)

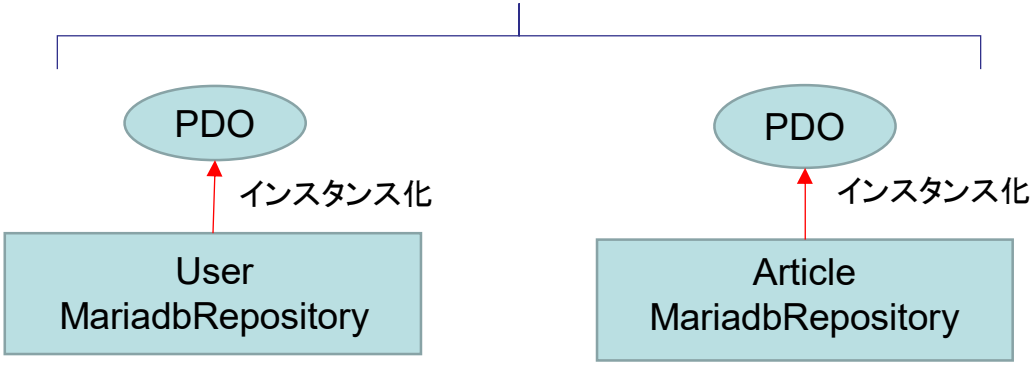
以下のような接続処理を、毎回書かなくていいようにするため。

```
$connection = new PDO('mysql:host=mariadb; dbname=enjoy_eats', 'root', 'abcde123');
$connection->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$connection->setAttribute(PDO::ATTR_EMULATE_PREPARES, false);
```

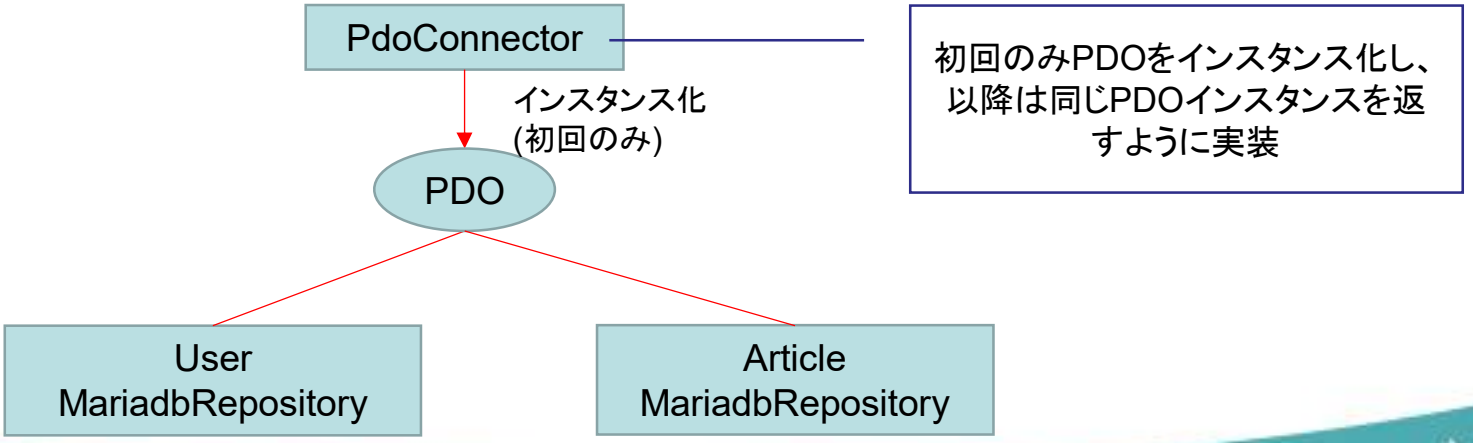
データベース接続を一元化したい理由(2)

アプリケーション全体で、1つのPDOインスタンスを共有できるようにするため。

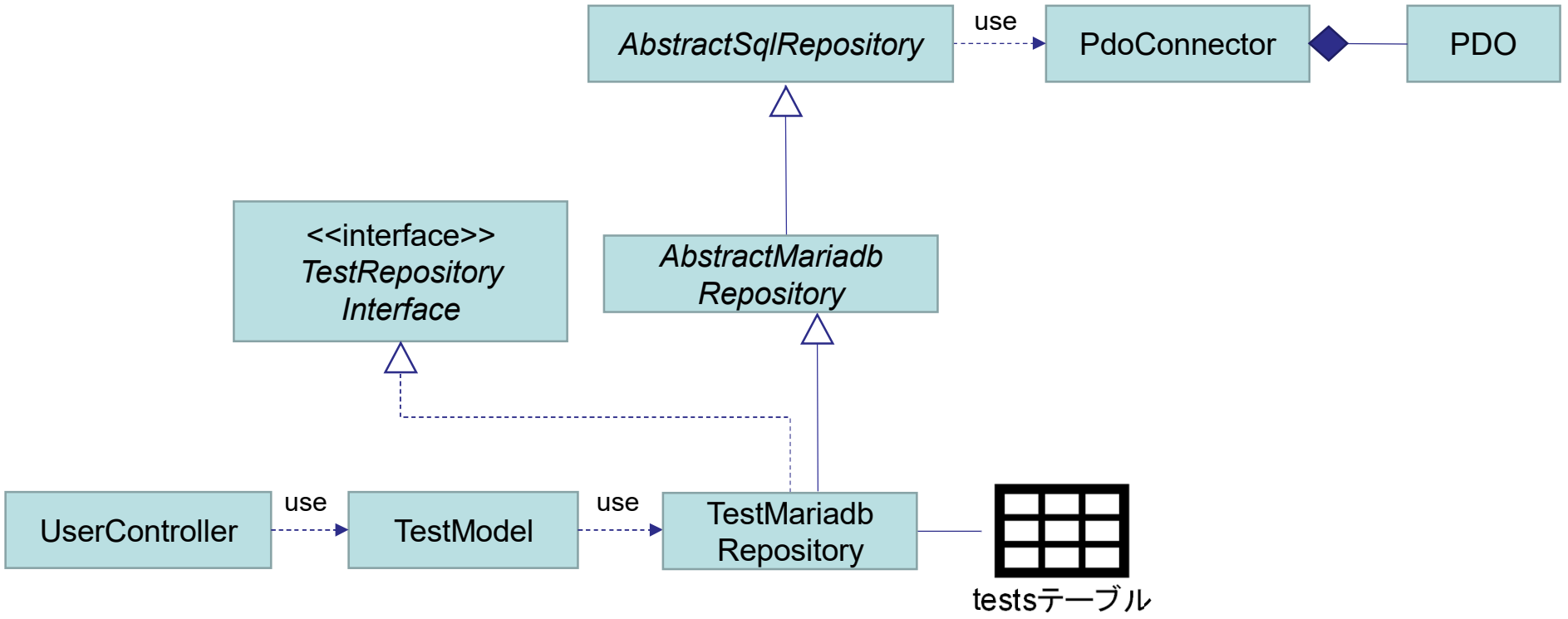
★2つのPDOインスタンスをまたいだトランザクション管理はできない



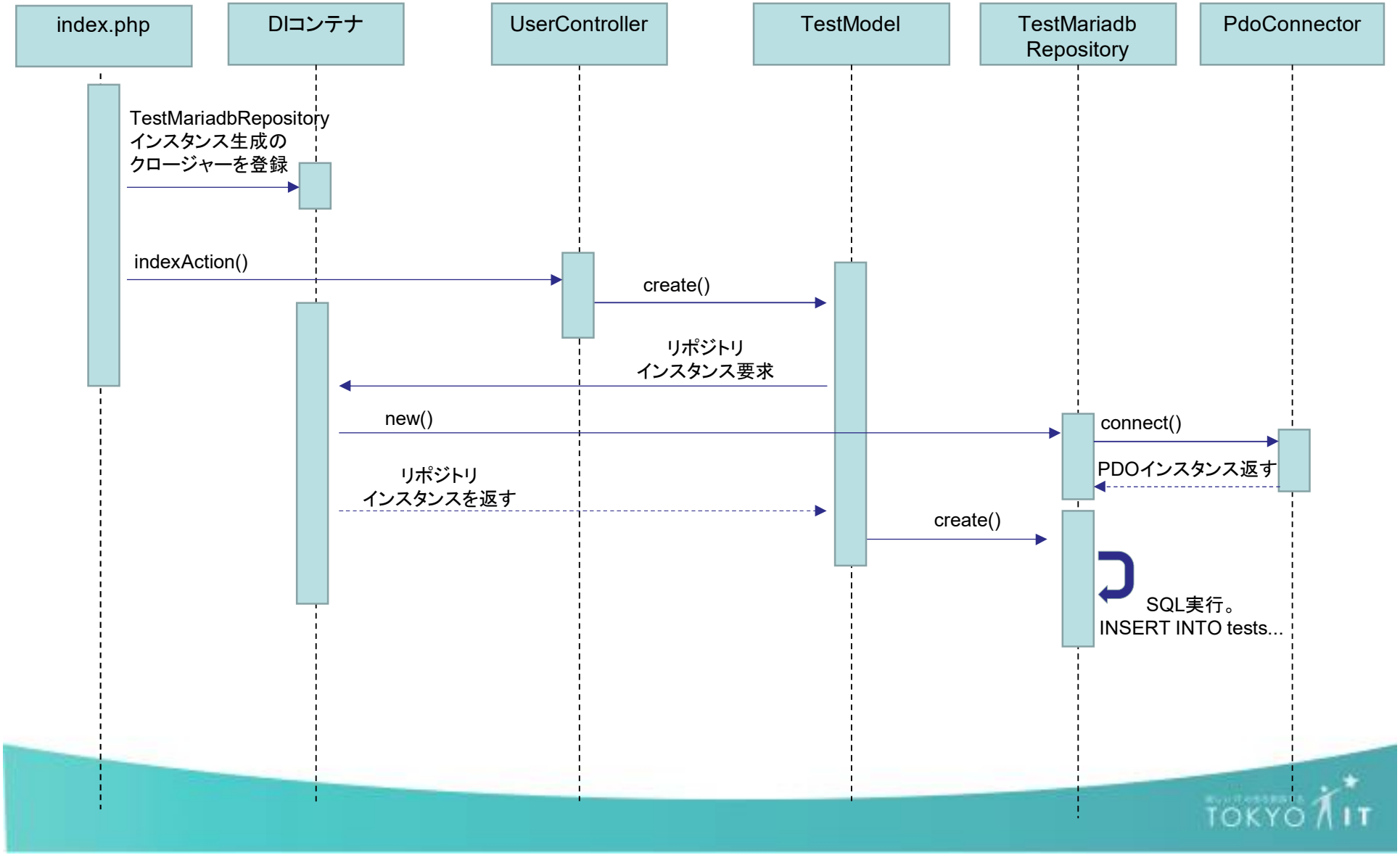
★この方式なら、モデルをまたいだトランザクション管理ができる



本ステップのクラス構成



本ステップの処理の流れ



PDOクラスを参照するときの注意

名前空間に所属するクラスからPDOクラスを参照するときは、名前空間の違いに配慮する必要があります。

```
namespace App¥Libs;

class PdoConnector {
    public function connect() {
        $connection = new PDO();
    }
}
```

このように単に new PDO(); と書くだけだとエラーになります。
自クラスが所属する名前空間を起点に、App¥Libs¥PDO を参照しようとするためです。

プログラムの冒頭に use PDO; と書いておくか、new ¥PDO(); と書くようにしましょう。

他のPHP標準クラスも同様です。たとえば、throw new Exception(); ではなく
throw new ¥Exception(); のようにアクセスします。

本ステップの変更ファイル一覧

●追加したファイル

- app/Libs/DataSource/DataSourceConnectorInterface.php
  → データソースに接続するためのインターフェース

- app/Libs/DataSource/PdoConnector.php
  → データベースに接続し、PDOインスタンスを返すクラス。
  DataSourceConnectorInterfaceを実装する。

- app/Models/TestModel.php
  → テスト用に作った簡易的なモデルクラス

- app/Repositories/TestMariadbRepository.php
  → testsテーブルを操作するためのリポジトリクラス

- app/Repositories/TestRepositoryInterface.php
  → testsテーブルを操作するためのリポジトリクラス用のインターフェース

- app/Libs/AbstractMariadbRepository.php
  → Mariadbリポジトリクラス用のスーパークラス

- app/Libs/AbstractSqlRepository.php
  → リレーショナルデータベースリポジトリクラス用のスーパークラス

本ステップの変更ファイル一覧

●変更したファイル

- app/Modules/User/Controllers/UserController.php
  → モデルを呼び出す処理を追加

- app/Core/container.php
  → PdoConnectorをDIコンテナに登録する処理を追加
  → TestMariadbRepositoryをDIコンテナに登録する処理を追加

参考情報

- PHP本格入門(上)
  「6-3 PHPプログラムからデータベース操作する - PDOの利用」