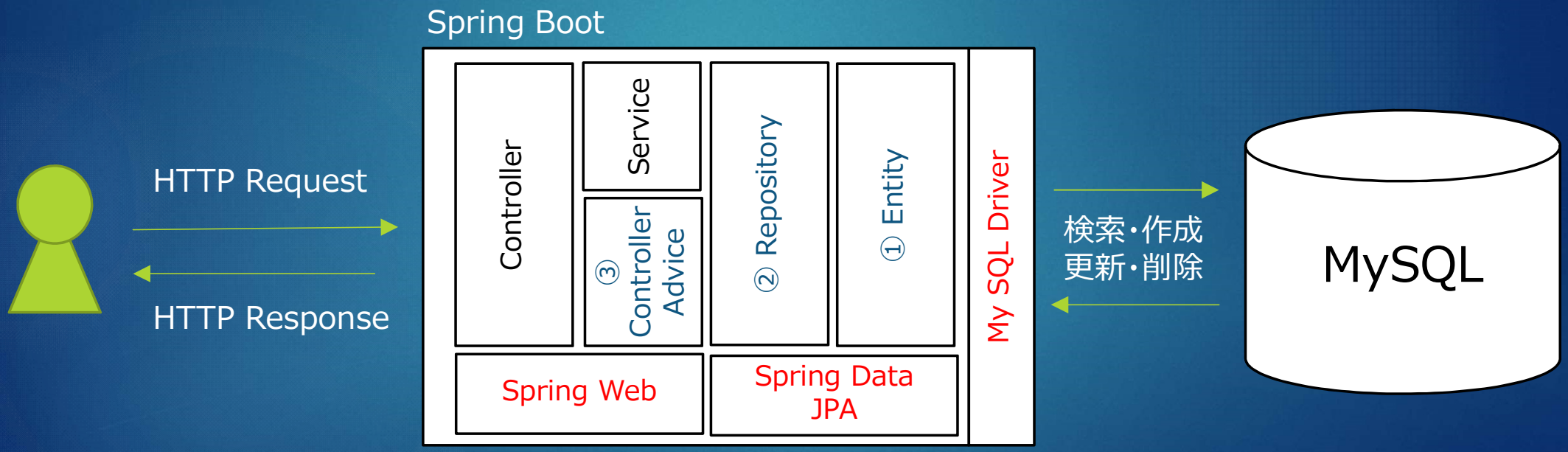
Rest APIをデータベース(MySQL)に対応させる

演習で作成したRest APIをデータベースと連携する形に修正していきます。赤字は、依存関係の指定を示します。
Spring Data JPAを活用した、①・②　および、クライアントに例外を返すための③を本パートで学びます。



Spring Data JPAとは (1/2)

Spring Data JPAは、JPAを実装しているHibernateを活用し、データアクセスを簡単に実装する事を可能とします。

実装内容

Java Object

```java
@Entity(name="m_item")
public class Item {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long itemId;
    private String itemName;
    private String itemCategory;
```

Relational Database

m_itemテーブル

Field	Type	Null	Key	Default	Extra
item_id	bigint	NO	PRI	NULL	auto_increment
item_category	varchar(255)	YES		NULL	
item_name	varchar(255)	YES		NULL	

application.properties

```
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

ポイント

- ✓ @Entity：Entityクラスと示し、name属性に実際の対応するテーブル名を指定します。
- ✓ @Id：キー値(ID)となる項目に指定します。
- ✓ @GeneratedValue：ID生成方針を定義します。GenerationType.IDENTITYは、キー値生成をDBの機能で行う。→ MySQLではauto_incrementに対応します。
- ✓ spring.jpa.hibernate.ddl-auto：createを指定すると、アプリケーション起動時にEntityに対応するテーブルがあれば削除し、新規作成します。
- ✓ spring.jpa.show-sql：trueと指定すると実際に流れるSQLを表示します。
- ✓ spring.jpa.properties.hibernate.format_sql：trueと指定すると表示されるSQLを見やすく出力します。

Spring Data JPAとは (2/2)

Spring Data JPAは、インターフェースを定義するだけで、あらかじめ用意されたデータ操作を行う事ができます。

実装内容	ポイント

Repository
インターフェース

```java
@Repository
public interface ItemRepository extends CrudRepository<Item, Long>{

}
```

- ✓ @Repository：DBアクセスを行う事を示します
- ✓ CrudRepository ：エンティティクラスと、IDを指定する事で、CRUD機能を提供します
- ✓ Interfaceだけを定義。実装クラスは不要。

Service
クラス

```java
@Service
public class ItemService {

    @Autowired
    private ItemRepository itemRepository;

    public List<Item> getAllItems(){
        List<Item> allItems = new ArrayList<>();
        itemRepository.findAll().forEach(allItems::add);

        return allItems;
    }
```

- ✓ Serviceクラスで@Autowiredをつけてそのまま利用するだけ。
- ✓ CrudRepositoryで使用できるメソッドを紹介

- save ： 指定したエンティティを登録・更新する
- findById (id) ：キー値を指定すると、指定されたエンティティを検索する
- findAll ：対応するテーブルの全件を検索する
- deleteById (id) ：指定されたキー値のデータを削除する

Controllerで例外を共通処理として対処する @ControllerAdvice

@ControllerAdviceを利用して、Controllerにて共通に例外処理を行い、HTTPレスポンスとして、クライアントにエラー内容を任意のステータスで返すことができます。

実装内容

独自に作成した実行時例外

```java
public class ItemNotFoundException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    public ItemNotFoundException(Long itemId) {
        super("商品コード:" + itemId + "は見つかりません。");
    }

}
```

エラー処理をコントローラの共通処理とする(ハンドリング)

```java
@ControllerAdvice
public class ItemNotFoundExceptionControllerAdvice {

    @ResponseBody
    @ExceptionHandler(ItemNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public String itemNotFoundHandler (ItemNotFoundException ex) {
        return ex.getMessage();
    }

}
```

コントローラで独自例外をThrow

```java
@GetMapping("/items/{itemId}")
public Item getItem(@PathVariable("itemId") Long itemId) {
    return itemService.getItem(itemId).orElseThrow(()
            -> new ItemNotFoundException(itemId));
}
```

クライアントの表示

ポイント

- ✓ @ControllerAdvice　：　すべてのControllerクラスで発生した例外に対して、共通の設定を行う事ができます
- ✓ @ResponseBody ： レスポンスとしてJSONを返却する
- ✓ @ExceptionHandler ：指定した例外クラスがControllerで発生した場合に、該当メソッドでハンドリングする
- ✓ @ResponseStatus ： クライアントにレスポンスを返すステータスコードを指定する。この場合は404エラー
- ✓ Controllerであらかじめ設定した例外がThrowされると、@ControllerAdviceで指定したクラスがハンドリングする

実機演習の流れ

1） spring3itemプロジェクトの新規作成 (Spring Web / Spring Data JPA / My SQL Driver を構成)

2）エンティティクラスを修正 (以前のItemクラスを利用します)

3）リポジトリインターフェースを作成

4）サービスクラスをリポジトリインタフェース経由でデータ操作をするように修正 (以前のItemServiceクラスを利用)

5）指定された商品が見つからない場合の例外クラスを作成

6）例外がControllerで共通的に処理されるよう、ConrollerAdviceクラスを作成

7）Controllerクラスを修正 (以前のItemControllerクラスを利用)

8）Applicationクラスに起動時にデータを投入する処理を追加 (※演習のため。実装内容はご参考。)

9）application.properites に、データベース接続情報、Spring Data JPAの設定を記載

10） 動作確認