

A faint, light blue world map is visible in the background of the slide, centered behind the text.

INTERNET ACADEMY

Institute of Web Design & Software Services

Spring Boot 6

インターネット・アカデミー

Spring Boot 6 目次

- JPAによるDB操作 2 (H2 database)
- LOMBOK

A faint, light blue world map is visible in the background of the slide, centered behind the title text.

JPAによるDB操作 2 (H2 database)

JPAによるDB操作(INSERT)

http://localhost:8080/form

JPA(INSERT : 登録)入力画面

名前:

次へ進む



http://localhost:8080/confirm

JPA(INSERT : 登録)確認画面

入力内容ここから

テスト太郎

入力内容ここまで

入力画面に戻る

完了画面へ進む



http://localhost:8080/complete

JPA(INSERT : 登録)完了画面

[H2コンソールへ](#)

[一覧ページへ](#)

▼登録後のH2コンソール

jdbc:h2:mem:test

Run Run Selected Auto c

SELECT * FROM STAFF

STAFF

INFORMATION_SCHEMA

Sequences

Users

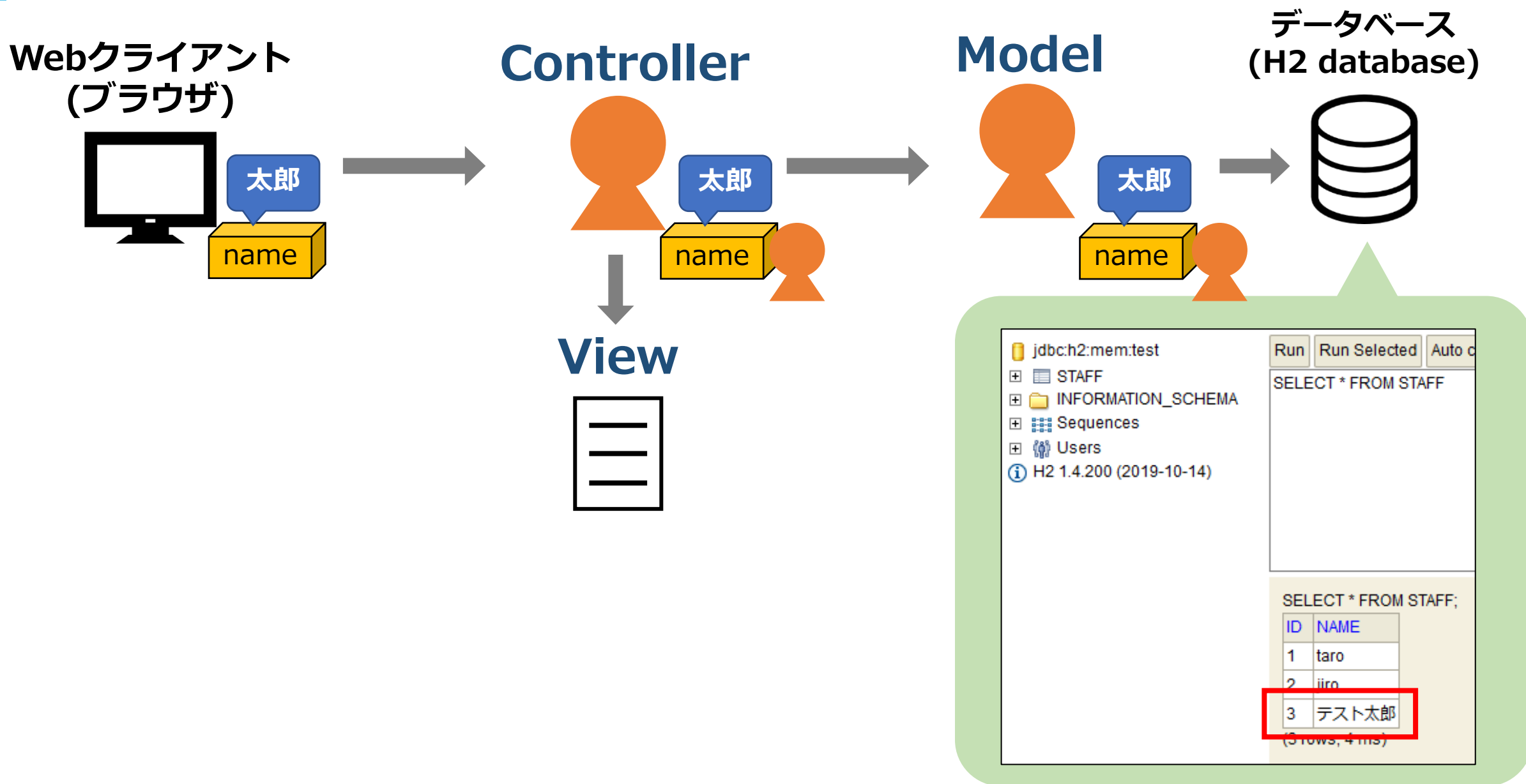
H2 1.4.200 (2019-10-14)

SELECT * FROM STAFF;

ID	NAME
1	taro
2	jiro
3	テスト太郎

(3 rows, 4 ms)

JPAによるDB操作(INSERT)



アプリケーション作成時の選択

項目	目的
Spring Boot DevTools	開発を便利にするツール (コード変更時の自動再起動など)
Thymeleaf	テンプレートエンジン
Spring Web	Spring MVCを使う
Spring Data JPA	JPAライブラリを使う
H2 Database	データベースにH2を使う

 新規 Spring スターター・プロジェクト依存関係

Spring Boot バージョン: 2.4.5

使用頻度高:

☒ H2 Database

☐ JDBC API

☐ Lombok

☐ MySQL Driver

☒ Spring Boot DevTools

☒ Spring Data JPA

☒ Spring Web

☒ Thymeleaf

☐ 検証

使用可能:

検索する依存関係を入力

▶ Alibaba

▶ Amazon Web サービス

▶ 開発ツール

▶ Google Cloud Platform

▶ I/O

▶ メッセージング

▶ Microsoft Azure

▶ NoSQL

▶ Observability

▶ Ops

▶ SQL

▶ セキュリティ

▶ Spring Cloud

▶ Spring Cloud Circuit Breaker

▶ Spring Cloud Config

選択済み:

X Spring Boot DevTools

X Spring Data JPA

X H2 Database

X Thymeleaf

X Spring Web

デフォルトにする

選択をクリア

?

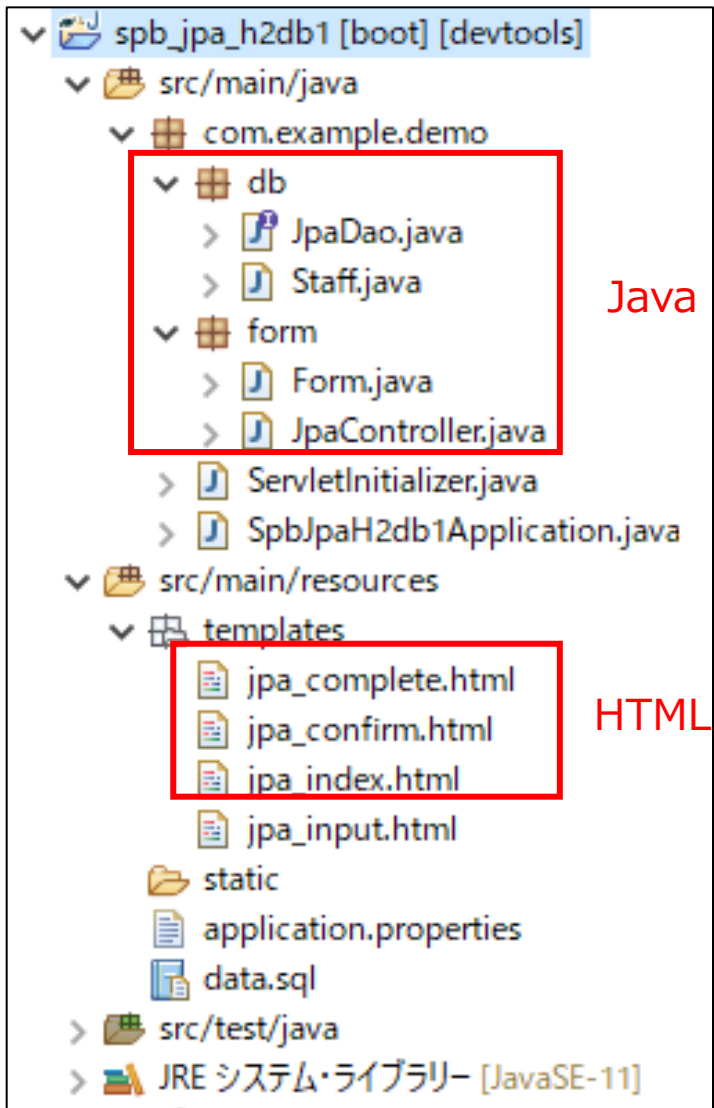
< 戻る(B)

次へ(N) >

完了(F)

キャンセル

ファイル構成



データベースの設定※確認用

application.properties

```
spring.datasource.driverClassName: org.h2.Driver  
spring.datasource.url: jdbc:h2:mem:test  
spring.datasource.username: sa  
spring.datasource.password:  
spring.h2.console.enabled: true
```

#JPAによるテーブル生成

```
spring.jpa.hibernate.ddl-auto=update
```

data.sql

```
INSERT INTO staff (id,name) VALUES (1,'taro');  
INSERT INTO staff (id,name) VALUES (2,'jiro');
```

※schema.sqlは作成しない

エンティティの指定※確認用

Staff.java(前半)

```
package com.example.demo.db;
```

```
import javax.persistence.Entity;  
import javax.persistence.GeneratedValue;  
import javax.persistence.GenerationType;  
import javax.persistence.Id;  
import javax.persistence.Table;
```

```
@Entity
```

```
@Table(name="staff")
```

```
public class Staff {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Integer id;
```

```
    private String name;
```

```
    public Staff() {}
```

```
    :
```

```
    :
```

※続きのコードは次のスライドにて

エンティティの指定※確認用

Staff.java(後半)

```
:
:
public Integer getId() {
    return id;
}
public void setId(Integer id) {
    this.id = id;
}
public String getName() {
    return name;
}
public void setName(String name) {
    this.name = name;
}

@Override
public String toString() {
    return "Staff [id=" + id + ", name=" + name + "]";
}
}
```

JpaController.java
内で使います

Model(DAO)の指定※確認用

JpaDao.java ※インターフェイスです

```
package com.example.demo.db;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface JpaDao extends JpaRepository<Staff, Long>{

}
```

Controllerの指定(追記あり)

JpaController.java

```
:
@Controller
public class JpaController {

    //DAOのオブジェクトが座る椅子を用意
    private final JpaDao jpadao;

    //DAOを予め控室に待機させておき、必要なときに呼んで座らせる
    @Autowired
    public JpaController(JpaDao jpadao) {
        this.jpadao = jpadao;
    }

    ※この部分は次のスライドにて記載

}
```

Controllerの指定(追記あり)

JpaController.java ※中の部分その1

```
// 【JPAでSELECT】  
:
```

```
//入力画面(localhost:8080/form)  
@RequestMapping("/form")  
public String form(Model model, Form form) {  
    return "jpa_input";  
}
```

入力画面用の処理

```
//確認画面(localhost:8080/confirm)  
@RequestMapping("/confirm")  
public String confirm(Model model, @Validated Form form, BindingResult result) {  
  
    //エラーがあれば入力画面に戻す  
    if(result.hasErrors()) {  
        return "jpa_input";  
    }  
  
    return "jpa_confirm";  
}  
:
```

確認画面用の処理

Controllerの指定(追記あり)

JpaController.java ※中の部分その2

```
// 【JPAでSELECT】
:
//入力画面(localhost:8080/form)
:
//確認画面(localhost:8080/confirm)
:
// 【JPAでINSERT】
//完了画面・登録(INSERT)
@RequestMapping ("/complete")
public String complete(Form form, Model model){

    //フォームの値をエンティティに入れ直す
    Staff s1 = new Staff();
    s1.setName(form.getName());

    //JPAでINSERT実行
    jpadao.save(s1);

    return "jpa_complete";
}
}
```

完了画面用の処理

Controllerの指定(追記あり)

JpaController.java ※中の部分その2

```
// 【JPAでSELECT】
:
//入力画面(localhost:8080/form)
:
//確認画面(localhost:8080/confirm)
:
// 【JPAでINSERT】
//完了画面・登録(INSERT)
@RequestMapping ("/complete")
public String complete(Form form, Model model){

    //フォームの値をエンティティに入れ直す
    Staff s1 = new Staff();
    s1.setName(form.getName());

    //JPAでINSERT実行
    jpadao.save(s1);

    return "jpa_complete";
}
}
```

Controllerの指定(追記あり)

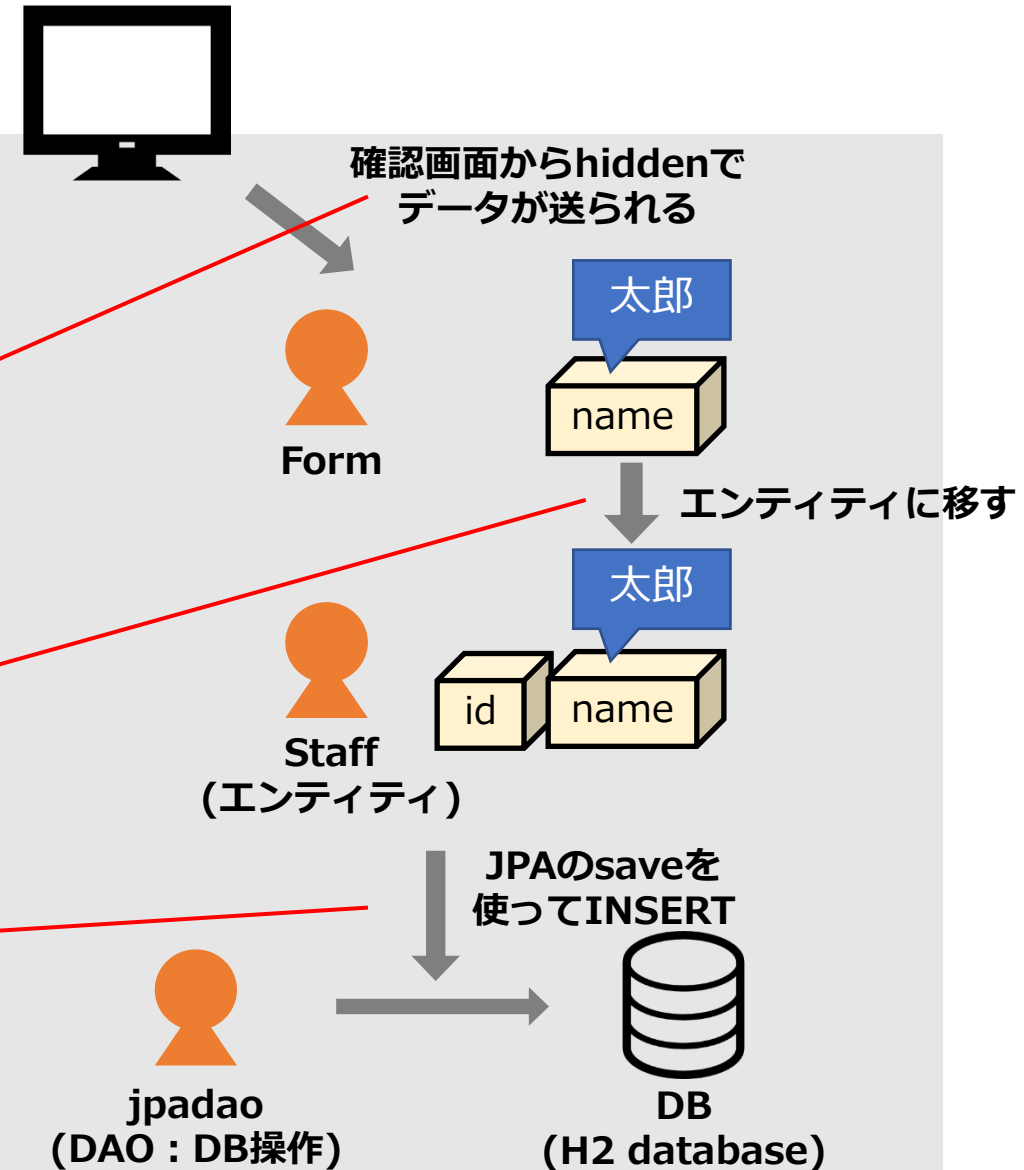
JpaController.java ※中の部分その2

```
// 【JPAでSELECT】
:
//入力画面(localhost:8080/form)
:
//確認画面(localhost:8080/confirm)
:
// 【JPAでINSERT】
//完了画面・登録(INSERT)
@RequestMapping ("/complete")
public String complete(Form form, Model model){

    //フォームの値をエンティティに入れ直す
    Staff s1 = new Staff();
    s1.setName(form.getName());

    //JPAでINSERT実行
    jpadao.save(s1);

    return "jpa_complete";
}
```



Viewの指定(入力画面)

jpa_input.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>JPA</title>
<meta charset="utf-8" />
</head>
<body>
  <h1>JPA(INsert : 登録)入力画面</h1>
  <form action="#" method="get" th:action="@{/confirm}" th:object="${form}">
    <p>名前: <input type="text" name="name" th:value="*{name}"></p>
    <div th:if="${#fields.hasErrors('name')}" th:errors="*{name}" class="error">エラー</div>
    <input type="submit" value="次へ進む">
  </form>
</body>
</html>
```

Viewの指定(確認画面)

jpa_confirm.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>JPA</title>
<meta charset="utf-8" />
</head>
<body>
  <h1>JPA(INSERT : 登録)確認画面</h1>
  <p>入力内容ここから</p>
  <p th:text="${form.name}"></p>
  <p>入力内容ここまで</p>
  <form action="#" method="get" th:action="@{/form}">
    <input type="hidden" name="name" th:value="${form.name}">
    <input type="submit" value="入力画面に戻る">
  </form>
  <form action="#" method="get" th:action="@{/complete}">
    <input type="hidden" name="name" th:value="${form.name}">
    <input type="submit" value="完了画面へ進む">
  </form>
</body>
</html>
```

Viewの指定(完了画面)

jpa_complete.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<title>JPA</title>
<meta charset="utf-8" />
</head>
<body>
    <h1>JPA(INsert : 登録)完了画面</h1>

    <p><a href="http://localhost:8080/h2-console">H2コンソールへ</a></p>
    <p><a href="/">一覧ページへ</a></p>

</body>
</html>
```

【まとめ】 JPAのDB操作メソッド

メソッド	説明
<code>jpadao.findAll()</code>	データを全件取得(SELECT)
<code>jpadao.findById(id)</code>	データを 1 件取得(SELECT)
<code>jpadao.save(entity)</code>	データを保存(INSERT)
<code>jpadao.deleteById(id)</code>	データを 1 件削除(DELETE)

！ ポイント

UPDATEは仕様がないため、メソッドを自作する必要があります(Native Query)

DAOへのUPDATE文の追加

JpaDao.java ※インターフェイスです

```
package com.example.demo.db;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface JpaDao extends JpaRepository<Staff, Long>{

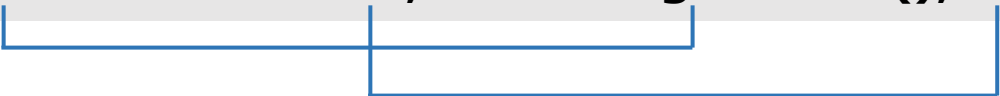
    @Modifying
    @Query(value = "UPDATE sample AS s SET s.name = :pname WHERE s.id = pid", nativeQuery = true)
    void updateDb(@Param("pname") String pname, @Param("pid") Long pid);

}
```

素のSQL(ネイティブ
クエリ)を利用する宣言

▼JDBCのときのDAO(SampleDao.javaより引用)

```
db.update("UPDATE sample SET name = ? WHERE id = ?", entform.getName(), id);
```



DAOへのUPDATE文の追加

JpaDao.java ※インターフェイスです

```
package com.example.demo.db;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface JpaDao extends JpaRepository<Staff, Long>{

    @Modifying
    @Query(value = "UPDATE sample AS s SET s.name = :pname WHERE s.id = pid", nativeQuery = true)
    void updateDb(@Param("pname") String pname, @Param("pid") Long pid);

}
```

UPDATE文の場合に必要

素のSQL(ネイティブ
クエリ)を利用する宣言

▼JDBCのときのDAO(SampleDao.javaより引用)

```
db.update("UPDATE sample SET name = ? WHERE id = ?", entform.getName(), id);
```

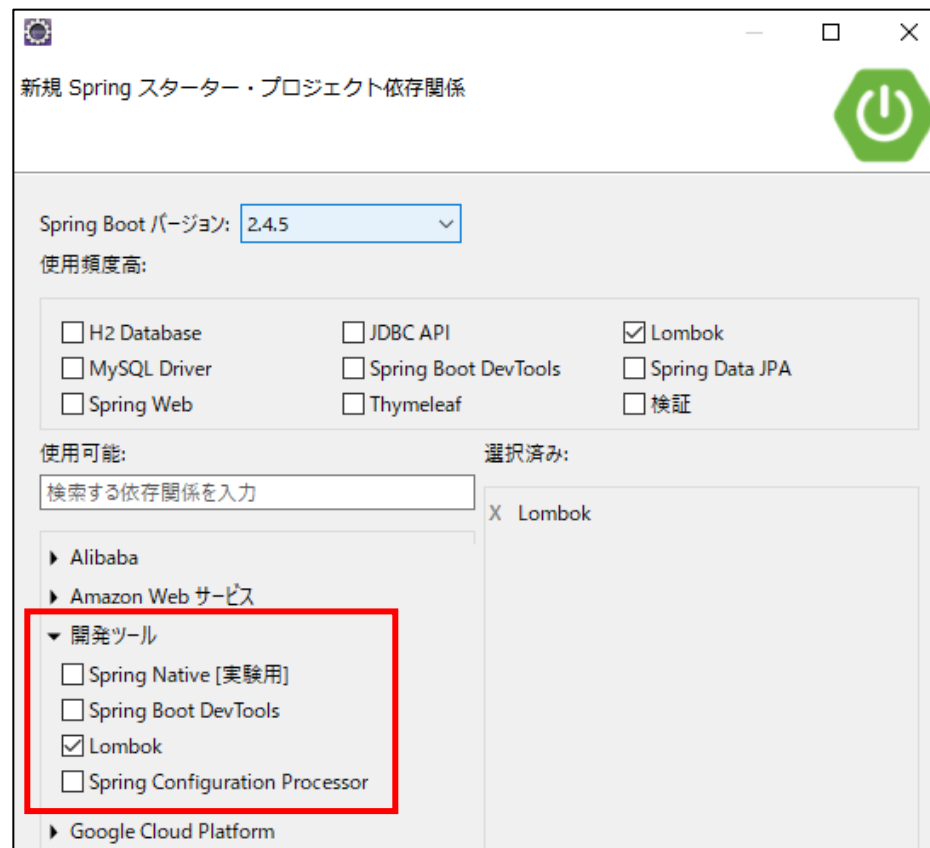


LOMBOK

LOMBOKとは

LOMBOK(ロンボック、ロンボク)とは、Java特有の冗長なコードを、アノテーションを使って簡潔に記述できるようになるオープンソースのライブラリ。

※EclipseでSpringBoot用アプリケーションを作成するときに、LOMBOKを選択すればすぐに使えます



@Data

getterメソッド、setterメソッド、toString()メソッドなど、自動生成します。

```
public class EntForm{
    private int id;
    private String name;

    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String toString() {
        return "EntForm [id=" + id + ", name=" + name + "]";
    }
}
```



```
import lombok.Data;

@Data
public class EntForm{
    private int id;
    private String name;
}
```

@Getter、@Setter

getterメソッド、setterメソッドを自動生成します。

```
public class EntForm{  
    private int id;  
  
    public int getId() {  
        return id;  
    }  
    public void setId(int id) {  
        this.id = id;  
    }  
}
```



```
import lombok.Data;  
  
public class EntForm{  
    @Getter  
    @Setter  
    private int id;  
}
```

@toString

toStringメソッドを自動生成します。

```
public class EntForm{  
    private int id;  
    private String name;  
  
    public String toString() {  
        return "EntForm [id=" + id + ", name=" + name + "];"  
    }  
}
```



```
import lombok.Data;  
  
@ToString  
public class EntForm{  
    private int id;  
    private String name;  
}
```

@AllArgsConstructor

すべてのフィールドを引数に持つコンストラクタを自動生成します。

```
public class EntForm{  
    private int id;  
    private String name;  
  
    public EntForm(int id, String name) {  
        this.id = id;  
        this.name = name;  
    }  
}
```



```
import lombok.Data;  
  
@AllArgsConstructor  
public class EntForm{  
    private int id;  
    private String name;  
}
```

！ ポイント

@Dataと@AllArgsConstructorを併用して書くことができます

@NoArgsConstructor

引数なしのコンストラクタを自動生成します。

```
public class EntForm{  
    private int id;  
    private String name;  
  
    public EntForm() {  
    }  
}
```



```
import lombok.Data;  
  
@NoArgsConstructor  
public class EntForm{  
    private int id;  
    private String name;  
}
```

！ポイント

@Dataと@AllArgsConstructorを併用する場合、引数なしのコンストラクタは生成されません。引数なしのコンストラクタも必要な場合には、@NoArgsConstructorを、さらに付与します(合計3つのアノテーションを書きます)。