6-6.テーブルのデータを一覧表示する

テーブルのデータを一覧表示する

本章では、データベースに格納されているToDoをブラウザに一覧を表示します。

今回使用するデータベースは 本カリキュラムでも使用した **postgreSQL** です。
pgAdminを起動し、下記データベースを作成してください。

データベース名：

> lesson_springboot

テーブル情報

列名	内容	データ型	制約	備考
id	1～	SERIAL	PRIMARY KEY	連番(自動採番)
title	件名	TEXT		
importance	重要度	INTEGER		0:低, 1:高
urgency	緊急度	INTEGER		0:低, 1:高
deadline	期限	DATE		
done	完了	TEXT		'Y':完了 'N':未完了

テーブルは下記SQLで作成することが出来ます。

> CREATE文

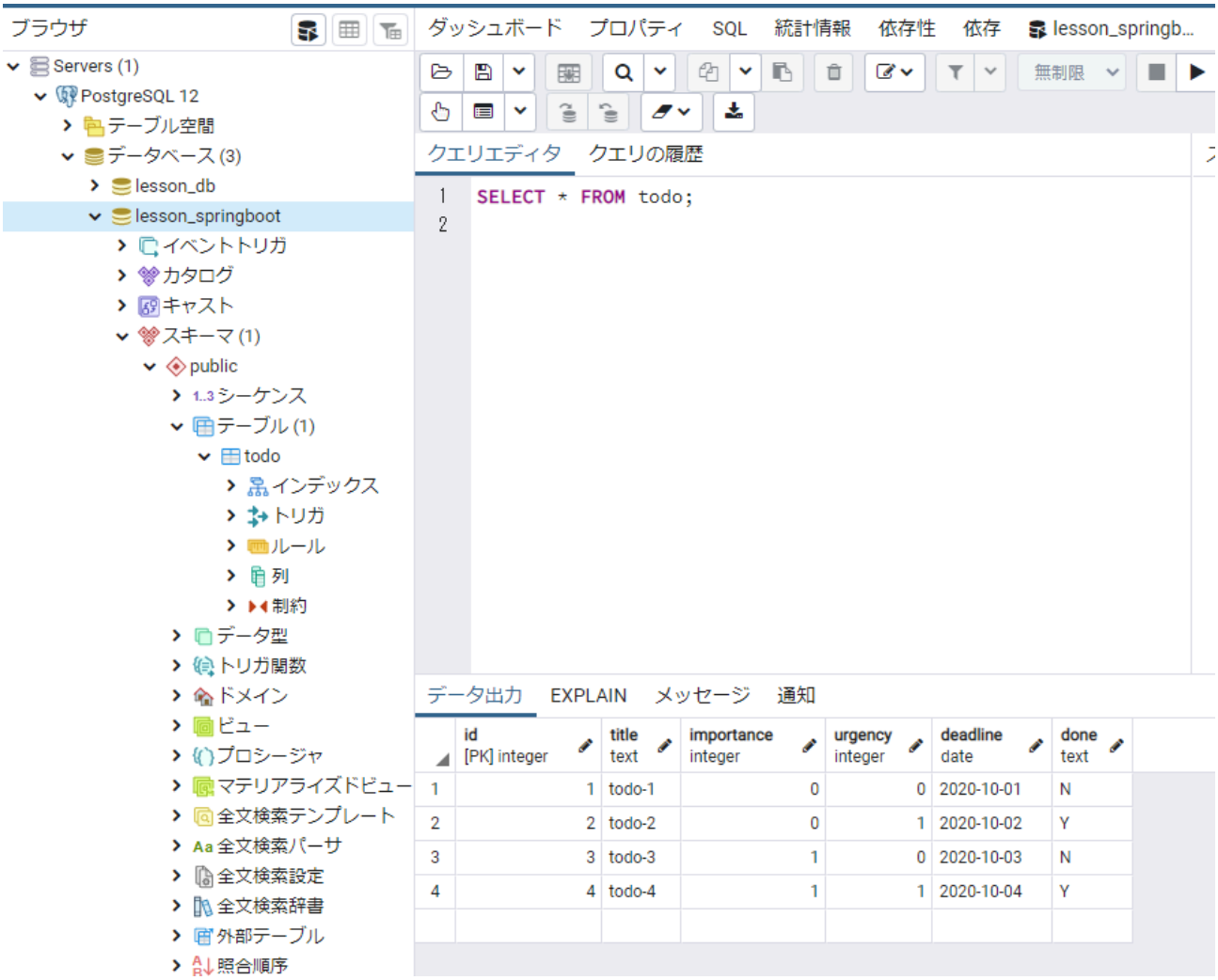
```
CREATE TABLE todo(
        id SERIAL NOT NULL,
        title TEXT,
        importance INTEGER,
        urgency INTEGER,
        deadline DATE,
        done TEXT,
        PRIMARY KEY (id)
)
```

> INSERT文

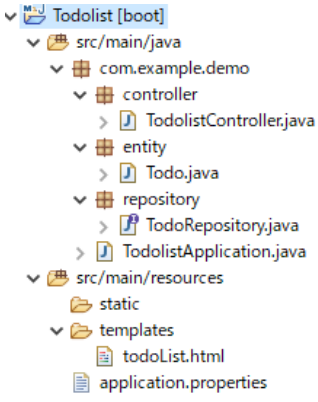
```
INSERT INTO todo(title, importance, urgency, deadline , done)
VALUES('todo-1', 0, 0, '2020-10-01', 'N');
INSERT INTO todo(title, importance, urgency, deadline, done)
VALUES('todo-2', 0, 1, '2020-10-02', 'Y');
INSERT INTO todo(title, importance, urgency, deadline, done)
VALUES('todo-3', 1, 0, '2020-10-03', 'N');
INSERT INTO todo(title, importance, urgency, deadline, done)
VALUES('todo-4', 1, 1, '2020-10-04', 'Y');
```

上記SQLを実行し、下記画像のように表示されていればテーブルの準備はOKです。

?



プロジェクト構成



※ TodoRepository.java はインターフェースです

プロジェクトの依存関係は下記です。

- 開発ツール→ **Spring Boot DevTools** , **Lombok**
- テンプレートエンジン→ **Thymeleaf**
- Web→ **Spring Web**
- I/O→ **検証**
- SQL→ **Spring Data JPA**, **PostgreSQL Driver**

JPA とは？

JPA は **Java Persistence API** の略です。
Persistence は日本語で「 **永続化** 」、簡単に言えば「データベースにデータを格納する」という意味です。
Spring Data JPA を使うとJava オブジェクトへの操作(メソッド)が、自動的にSQL文へ変換されます。つまりSQL文を書くことなく、テーブルをアクセスできるようになります。

PostgreSQL Driver は文字通り、PostgreSQL接続用のドライバーです。

エンティティクラスの作成

Todo.javaを下記に編集してください。

```
package com.example.demo.entity;

import java.sql.Date;
import javax.persistence.Column;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import lombok.Data;

@Entity
@Table(name = "todo")
@Data
public class Todo {
        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        @Column(name = "id")
        private Integer id;

        @Column(name = "title")
        private String title;

        @Column(name = "importance")
        private Integer importance;

        @Column(name = "urgency")
        private Integer urgency;

        @Column(name = "deadline")
        private Date deadline;

        @Column(name = "done")
        private String done;
}
```

`Todo`クラスは todoテーブルの列と **1 対 1** で対応しています。
つまり Todoオブジェクト1つでtodoテーブルの1レコードを表せます。
このように **テーブルのレコードを表すクラス** を、Spring Boot では「 **エンティティクラス** 」と呼びます。

Todo クラスで使っているアノテーションは次の通りです。

アノテーション	**内容**
@Entitty	このクラスがエンティティであることを示す
@Table	このエンティティに対応付けるテーブルを指定する →これによりTodoオブジェクトへの操作は、自動的にtodoテーブルのレコードに対する操作となる
@Id	テーブルの主キー(PRIMARY KEY) に対応するプロパティであることを表す。 →プロパティ名はid以外でも構わない。「@Idが付与されているプロパティが主キーに対応する」と解釈される
@GeneratedValue	主キーが自動採番されることを表す PostgreSQLでSERIAL型とした場合、strategyにはGenerationType.IDENTITYを指定する。
@column	プロパティに対応するテーブルの列を指定する。 プロパティ名と列名が同じなら省略可能（ただし大文字/小文字は区別されるので注意） 異なる名前にするとは指定が必要。 →テーブルのdone列をcheckedプロパティに対応付けるなら、以下のように付与する。 @column(name = "done") private String checked;

リポジトリの作成

次に TodoRepository.java を編集します。
SpringBootには、エンティティ(=テーブル)に対する処理を自動生成する仕組みがあります。これは「自分でコードを書かなくてよい」ということです。それを実現するのが、この「 **リポジトリ(Repository)** 」です。

```
package com.example.demo.repository;

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

import com.example.demo.entity.Todo;

@Repository
public interface TodoRepository extends JpaRepository<Todo, Integer> {

}
```

TodoRepositoryインターフェースのポイントは、次の2か所です。

@Repository

- このインターフェースがリポジトリであることを示すアノテーション
- @Repositoryを付与したインターフェースの名称は、対象エンティティクラス名＋"Repository" とするのが一般的です

JpaRepository<Todo, Integer>

- インターフェースの継承元
- 型引数<Todo, Integer>
  第1引数：このリポジトリが対象とするエンティティ（クラス）
  第2引数：対象エンティティで@Idが指定されているプロパティのクラス(Todoの場合、idなのでInteger)

このインターフェースには抽象メソッドがありません。しかし、継承されている **JpaRepository** により、テーブルに対するCRUDでの操作が一通りできるようになります。

コントローラーの作成

TodoListControllerの役目は、GETリクエストを受け取ったらtodoテーブルを検索し、その結果を一覧画面に渡すことです。検索には前節のTodoRepositoryを使います。

```
package com.example.demo.controller;

import java.util.List;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.servlet.ModelAndView;
import com.example.demo.entity.Todo;
import com.example.demo.repository.TodoRepository;
import lombok.AllArgsConstructor;

@Controller
@AllArgsConstructor
public class TodoListController {
	private final TodoRepository todoRepository;

	@GetMapping("/todo")
	public ModelAndView showTodoList(ModelAndView mv) {
		// 一覧 を 検索 し て 表示 する
		mv.setViewName("todoList");
		List<Todo> todoList = todoRepository.findAll();
		mv.addObject("todoList", todoList);
		return mv;
	}

}
```

@Autowired

@Autowired を付与すると、コントローラークラス起動時、付与しているフィールドに自動的にインスタンスがセットされます。
つまり、フィールドを自分で初期化する必要がありません。

```
@Autowired
private final TodoRepository todoRepository;
```

しかし、今回は **@Autowired** が付与されていません。
これは、Springbootには「 **フィールドにインスタンスの値をセットするコンストラクタ** が 1 つしかない場合、@Autowiredは省略できる」というルールがあります。

@AllArgsConstructor

@Autowired は **フィールドにインスタンスの値をセットするコンストラクタ** が 1 つしかない場合省略できると学びました。
しかし、Controllerクラスには フィールドに値をセットするコンストラクタが定義されていません。
実は、フィールドにインスタンスの値をセットするコンストラクタは **@AllArgsConstructor** をクラスに付与することで、そのクラスのフィールドにインスタンスの値を格納する **コンストラクタ** を省略することができます。

@AllArgsConstructor を付与しない場合

```
@Controller
public class TodoListController {
        private final TodoRepository todoRepository;

        public TodoListController(TodoRepository todoRepository) {
                this.todoRepository = todoRepository;
        }
 ～省略～
```

コンストラクタ(**@AllArgsConstructor** も含む)を使用せずにフィールドに **@Autowired** を付加するやり方を **フィールドインジェクション** といい、コンストラクタを使用してフィールドに値を格納するやり方を **コンストラクタインジェクション** といいます。
現在は、 **コンストラクタインジェクション** が推奨されています。

`showTodoList` メソッドも理屈がわかればシンプルです。
`@GetMapping`で **/todo** としていしているため、 `http://localhost:8080/todo` がリクエストされたら、 **showTodoList()** を実行します。

ポイントは次の 2行 です。

```
        List<Todo> todoList = todoRepository.findAll();
        mv.addObject("todoList", todoList);
```

`findAll()` は **JpaRepository** により自動実装されるメソッドの 1つです。これは **テーブルの全レコード** を検索します。
SELECT文 で表せば「 **SELECT * FROM todo** 」に相当します。
結果はList 型オブジェクトとして返されるので、そのまま一覧画面に渡して表示させます。

テーブルから取得されたものを表示する

最後は検索した ToDo 一覧表示画面(**View**)です。

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="UTF-8">
<title>ToDoList</title>
</head>
<body>
        <table border="1">
                <tr>
                        <th>id</th>
                        <th>件名</th>
                        <th>重要度</th>
                        <th>緊急度</th>
                        <th>期限</th>
                        <th>完了</th>
                </tr>
                <tr th:each="todo:${todoList}">
                        <td th:text="${todo.id}"></td>
                        <td th:text="${todo.title}"></td>
                        <td th:text="${todo.importance==1?'★★★':'★'}"></td>
                        <td th:text="${todo.urgency==1?'★★★':'★'}"></td>
                        <td th:text="${todo.deadline}"></td>
                        <td th:text="${todo.done=='Y'?'完了':''}"></td>
                </tr>
        </table>
</body>
</html>
```

要素の繰り返し

th:each 属性はjavaの **拡張for文** と同じように、コレクションオブジェクトの要素を処理するために使います。
右辺は「 **変数 ： ${コレクションオブジェクト}** 」という意味です。
ここでは、コレクションオブジェクトが `todoList` となっています。
これはTodoControllerが `addObject`メソッドによって、画面に渡したオブジェクトの名前です。
今回の場合 `findAll` メソッドで取得したレコードの数だけ `<tr>` 要素が繰り返されます。

変数の値を出力

th:text 属性は **"${ }"** の中に書かれている変数の値をタグのテキストに変換します。

検索結果がセットされれば、あとはそれを `th:text` で表示するだけです。このうち「重要度」「緊急度」「完了」は、条件に応じて表示内容を変えています。

項目	条件	表示内容
重要度	todo.importance == 1	★★★
重要度	上記以外	★
緊急度	todo.urgency == 1	★★★
緊急度	上記以外	★
完了	todo.done == 'Y'	完了
完了	上記以外	(何も表示しない)

この処理には **条件演算子(三項演算子)** を使っています。
構文は以下の通りです。

```
条件式 ? 式1 : 式2
```

条件式が **true** なら式1を返し、 **false** なら式2を返します。
thymeleaf も if文相当の属性を持っていますが、「true / false」で表示内容を変える」といった場合は、こちらの方がシンプルに書けます。

---

application.properties の編集

最後は **application.properties** です。このファイルには、プロジェクト全般にかかわる情報を定義します。
以下のように編集してください。

```
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql:lesson_springboot
spring.datasource.username=postgres
spring.datasource.password=postgres
```

※ **username** と **パスワード** は環境によって異なります。

application.properties には **PostgreSQL** への接続情報を記述します。
SpringBootはプロジェクト起動時、このファイルを読み取り、PostgreSQLにアクセスできるよう準備します。(PostgreSQLに接続するコードを書く必要はありません)。

さて、実行してみましょう。
下記画像のように表示されていれば成功です。

id	件名	重要度	緊急度	期限	完了
1	todo-1	★	★	2020-10-01	
2	todo-2	★	★★★	2020-10-02	完了
3	todo-3	★★★	★	2020-10-03	
4	todo-4	★★★	★★★	2020-10-04	完了

最終更新日時: 2022年 09月 10日(土曜日) 09:38