

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 8

インターネット・アカデミー

# Spring Boot 8 目次

- REST APIにおけるCRUDの概要確認
- REST APIでのCRUD操作 2



# REST APIにおけるCRUDの概要確認

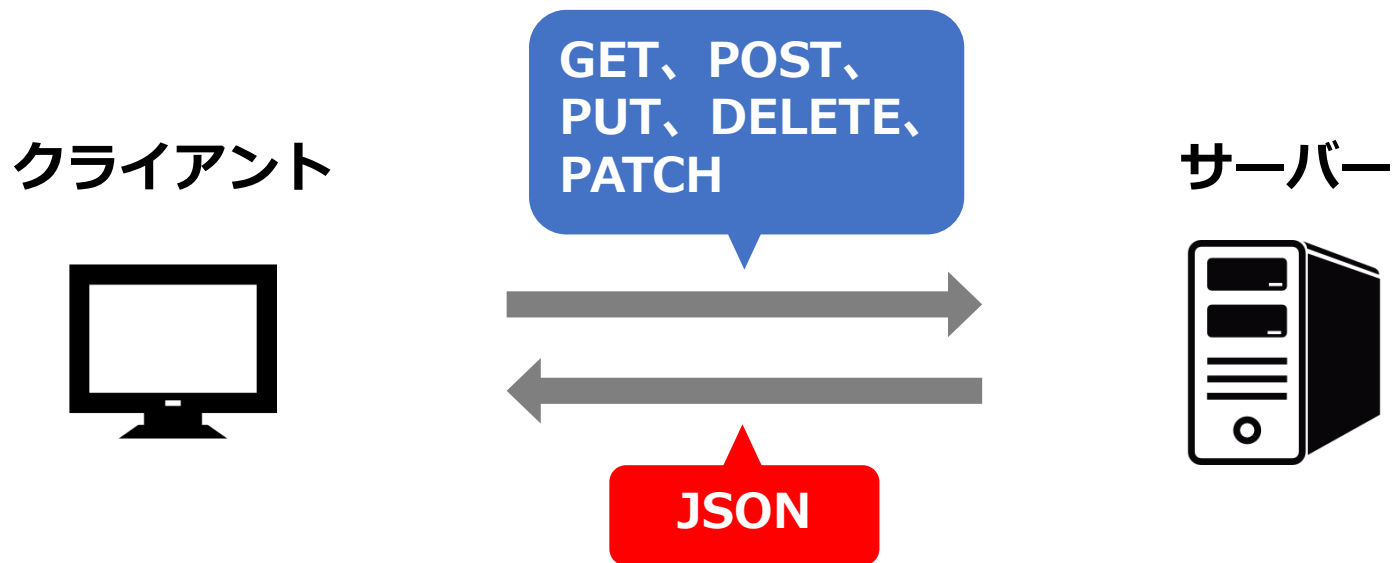
# フロントエンドとバックエンド



# RESTとは

REST(REpresentational State Transfer)とは、APIの設計ルール(アーキテクチャスタイル)。2000年に、ロイ・フィールドینگが考案。

## 【REST形式】



## ！ポイント

JSON形式で返ってくるため、次のページに飛ばずにページの内容を書き換えて済む

# REST APIでのCRUD処理

メソッド	CRUD	URL	処理内容	JPAのメソッド
GET	Read	/view	全件を取得	findAll()
GET	Read	/view/{id}	1件を取得	findById(id).get()
POST	Create	/create	登録	save(task)
PUT	Update	/update/{id}	更新	save(task)
DELETE	Delete	/del/{id}	削除	deleteById(id)
PATCH	Update	/update/{id}	部分更新	save(task)

## ！ポイント



RESTでは、ReadがGET、CreateがPOSTとなる点に注意しましょう

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

# REST APIでのCRUD操作 2

# 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
JDBC API	JDBCライブラリを使う
H2 Database	データベースにH2を使う
検証	バリデーション
Lombok	Javaの記述を簡素にする
<b>Restリポジトリ</b>	<b>REST APIを作る</b>

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

☐ MySQL Driver

☒ Spring Web

☒ JDBC API

☒ Spring Boot DevTools

☐ Thymeleaf

☒ Lombok

☒ Spring Data JPA

☒ 検証

使用可能:

選択済み:

▶ 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

▶ セキュリティー

▶ Spring Cloud

▶ Spring Cloud Circuit Breaker

▶ Spring Cloud Config

X Spring Boot DevTools

X Lombok

X 検証

X JDBC API

X Spring Data JPA

X H2 Database

X Spring Web

X Rest リポジトリ

デフォルトにする

選択をクリア

?

< 戻る(B)

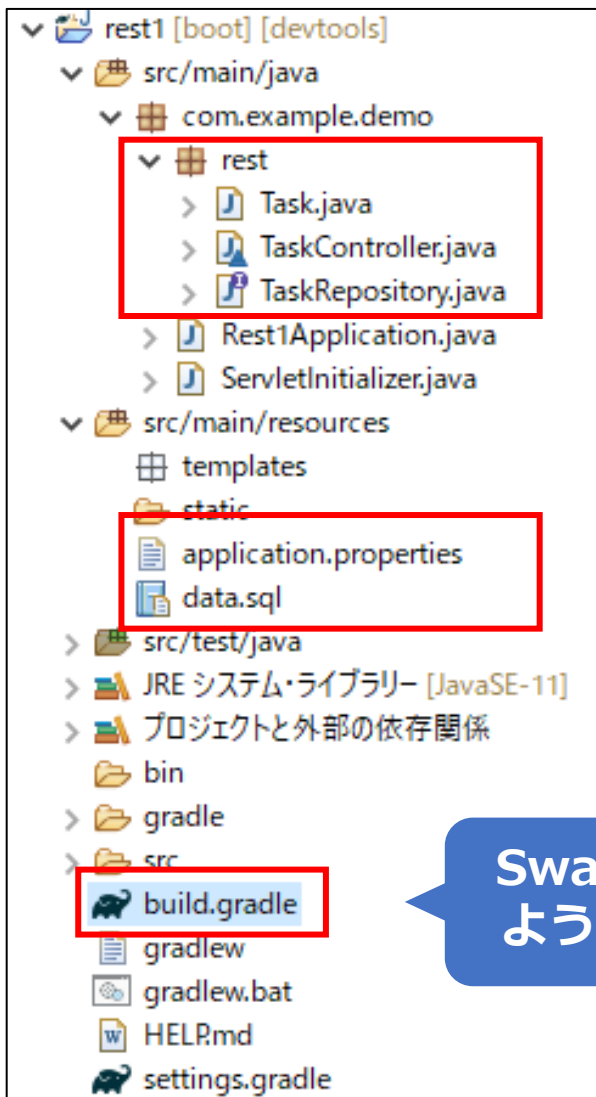
次へ(N) >

完了(E)

キャンセル



# ファイル構成



Swagger UIを使える  
ようにする指定を追記

# データベースの設定

## 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
```

**#バージョン2.5.0以降から必要(JPAの機能: @Entityがついたクラスと同名のテーブルを自動生成させる)**

```
spring.jpa.defer-datasource-initialization:true
```

## data.sql

```
INSERT INTO task (id, name, completed) VALUES (1, 'タスク 1', false);
```

※schema.sqlは作成しない

# Swagger UIを使うための設定

## build.gradle

```
:
dependencies {
    implementation 'org.springdoc:springdoc-openapi-ui:1.5.2'
    implementation 'org.springdoc:springdoc-openapi-data-rest:1.5.2'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-data-rest'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    compileOnly 'org.projectlombok:lombok'
    developmentOnly 'org.springframework.boot:spring-boot-devtools'
    runtimeOnly 'com.h2database:h2'
    annotationProcessor 'org.projectlombok:lombok'
    providedRuntime 'org.springframework.boot:spring-boot-starter-tomcat'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
}
```

# エンティティの指定

## Task.java ※前半

```
package com.example.demo.rest;

import java.io.Serializable;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.validation.constraints.NotBlank;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.AllArgsConstructor;
import lombok.Getter;
import lombok.NoArgsConstructor;
import lombok.Setter;

@Entity
@Getter
:
```

# エンティティの指定

## Task.java ※後半

```
:
@Entity
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
public class Task implements Serializable{
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @JsonProperty("id")
    private long id;

    @NotBlank
    @Size(max = 255)
    @JsonProperty("name")
    private String name;

    @NotNull
    @JsonProperty("completed")
    private Boolean completed;
}
```

# Model(DAO)の指定

## TaskRepository.java ※インターフェイスです

```
package com.example.demo.rest;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface TaskRepository extends JpaRepository<Task, Long>{

}
```

# Controllerの指定 1

## TaskController.java

```
package com.example.demo.rest;

import java.util.List;

import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import io.swagger.v3.oas.annotations.Operation;
import lombok.RequiredArgsConstructor;

@CrossOrigin
:
```

インポート文

# Controllerの指定 2

## TaskController.java

```
:
@CrossOrigin
@RequestMapping("/api1")
@RequiredArgsConstructor
@RestController
public class TaskController {
    private final TaskRepository repository;

    @Operation(summary = "タスクのテスト")
    @RequestMapping("/")
    Task test() {
        return new Task((long)1,"タスクのサンプル",false);
    }

    @Operation(summary = "タスクの全件取得")
    @GetMapping("/view")
    List<Task> findAll() {
        return repository.findAll();
    }
}
```

**CORS(Cross-Origin Resource Sharing)**とは、日本語で「オリジン間リソース共有」です。標準では制限がかかっており、異なるオリジン(ドメイン、プロトコル、ポート)のリクエストができません。つまり、`http://localhost:8080`で動かしている場合は、それ以外からAPIが呼び出せません。CORSを有効にして、APIを呼び出せるようにしています



# Controllerの指定 3

## TaskController.java

```
:  
@Operation(summary = "タスクの登録")  
@PostMapping("/create")  
Task save(@RequestBody Task task) {  
    return repository.save(task);  
}
```

リクエストに付加されているデータ  
(JSON形式で受け取ります)を取得し、  
それを用いてDBへ登録

```
@Operation(summary = "タスクの削除")  
@DeleteMapping("/del/{id}")  
void delete(@PathVariable Long id) {  
    repository.deleteById(id);  
}  
:
```

URLから、削除するidを受け取って、  
deleteById()でDBから削除

# Controllerの指定 4

## TaskController.java

```
:  
@Operation(summary = "タスクの1件取得")  
@GetMapping("/view/{id}")  
Task findOne(@PathVariable Long id) {  
    return repository.findById(id).get();  
}  
  
@Operation(summary = "タスクの更新")  
@PutMapping("/update/{id}")  
Task save(@RequestBody Task updateTask, @PathVariable Long id){  
    return repository.save(updateTask);  
}  
}
```

URLから、検索するidを受け取って、  
findById()でDBから検索する。  
リスト型になっているため、  
get()で1件のデータを取り出す

更新するidを、URLから受け取ります。  
更新するデータを、リクエストに付加されているデータ(JSON形式で受け取ります)から取得します。  
「更新」ですが、save()で上書きをして実行します

# 実行結果

最後の/(スラッシュ)忘れない

http://localhost:8080/api1/ にアクセス

## ▼Chrome



## ▼Firefox

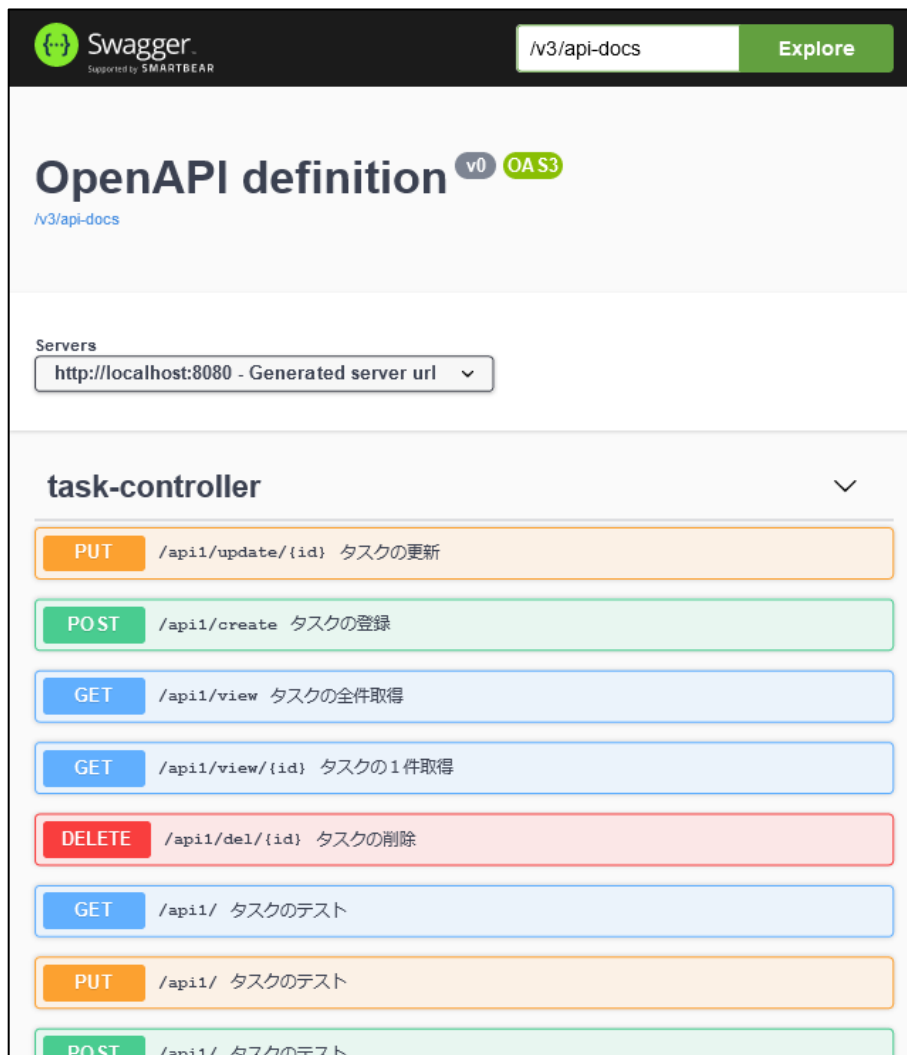


## ！ポイント

Firefoxだと、結果がわかりやすく表示されます

# 実行結果(Swagger UI)

Swagger UIの画面を出すために「<http://localhost:8080/swagger-ui.html>」 にアクセスします



`@PutMapping("/update/{id}")`

`@PostMapping("/create")`

`@GetMapping("/view")`

`@GetMapping("/view/{id}")`

`@GetMapping("/del/{id}")`

`@RequestMapping("/")`