构建RESTful服务

教学内容

- 第一节 RESTful介绍
- 第二节 构建RESTful应用接口
- 第三节 使用Swagger生成Web API文档

RESTful介绍

- RESTful是目前流行的互联网软件服务架构设计风格。
- REST（Representational State Transfer，表述性状态转移）一词是由Roy Thomas Fielding在2000年的博士论文中提出的，它定义了互联网软件服务的架构原则，如果一个架构符合REST原则，则称之为RESTful架构。
- REST并不是一个标准，它更像一组客户端和服务端交互时的架构理念和设计原则，基于这种架构理念和设计原则的Web API更加简洁，更有层次。

RESTful的特点

- 每一个URI代表一种资源
- 客户端使用GET、POST、PUT、DELETE四种表示操作方式的动词对服务端资源进行操作：GET用于获取资源，POST用于新建资源（也可以用于更新资源），PUT用于更新资源，DELETE用于删除资源。
- 通过操作资源的表现形式来实现服务端请求操作。
- 资源的表现形式是JSON或者HTML。
- 客户端与服务端之间的交互在请求之间是无状态的，从客户端到服务端的每个请求都包含必需的信息。

RESTful API

- 符合RESTful规范的Web API需要具备如下两个关键特性：
- 安全性：安全的方法被期望不会产生任何副作用，当我们使用GET操作获取资源时，不会引起资源本身的改变，也不会引起服务器状态的改变。
- 幂等性：幂等的方法保证了重复进行一个请求和一次请求的效果相同（并不是指响应总是相同的，而是指服务器上资源的状态从第一次请求后就不再改变了），在数学上幂等性是指N次变换和一次变换相同。

HTTP Method

- HTTP提供了POST、GET、PUT、DELETE等操作类型对某个Web资源进行Create、Read、Update和Delete操作。
- 一个HTTP请求除了利用URI标志目标资源之外，还需要通过HTTP Method指定针对该资源的操作类型，一些常见的HTTP方法及其在RESTful风格下的使用：

HTTP 方法	操　作	返回值	特定返回值
POST	Create	201（Created），　提交或保存资源	404（Not Found），409（Conflict）资源已存在
GET	Read	200（OK），获取资源或数据列表，支持分页、排序和条件查询	200（OK）返回资源，404（Not Found）资源不存在
PUT	Update	200（OK）或 204 （No Content），修改资源	404（Not Found）资源不存在，405（Method Not Allowed）禁止使用改方法调用
PATCH	Update	200（OK）or 204 （No Content），部分修改	404（Not Found）资源不存在
DELETE	Delete	200（OK），资源删除成功	404 （Not Found）资源不存在，405（Method Not Allowed）禁止使用改方法调用

HTTP状态码

- HTTP状态码就是服务向用户返回的状态码和提示信息，客户端的每一次请求，服务都必须给出回应，回应包括HTTP状态码和数据两部分。
- HTTP定义了40个标准状态码，可用于传达客户端请求的结果。状态码分为以下5个类别：
- 1xx：信息，通信传输协议级信息
- 2xx：成功，表示客户端的请求已成功接受
- 3xx：重定向，表示客户端必须执行一些其他操作才能完成其请求
- 4xx：客户端错误，此类错误状态码指向客户端
- 5xx：服务器错误，服务器负责这写错误状态码

HTTP状态码

- RESTful API中使用HTTP状态码来表示请求执行结果的状态，适用于REST API设计的代码以及对应的HTTP方法。

HTTP 状态码	返 回 值	HTTP Method	特定返回值
200	OK	GET	服务器成功返回用户请求的数据，该操作是幂等的（Idempotent）
201	Created	POST/PUT/PATCH	用户新建或修改数据成功
202	Accepted	*	表示一个请求已经进入后台排队（异步任务）
204	NO CONTENT	DELETE	用户删除数据成功
400	INVALID REQUEST	POST/PUT/PATCH	用户发出的请求有错误，服务器没有进行新建或修改数据的操作，该操作是幂等的
401	Unauthorized	*	表示用户没有权限（令牌、用户名、密码错误）
403	Forbidden	*	表示用户得到授权（与 401 错误相对），但是访问是被禁止的
404	NOT FOUND	*	用户发出的请求针对的是不存在的记录，服务器没有进行操作，该操作是幂等的
406	Not Acceptable	GET	用户请求的格式不可得（比如用户请求JSON格式，但是只有 XML 格式）
410	Gone	GET	用户请求的资源被永久删除，且不会再得到
422	Unprocesable entity	POST/PUT/PATCH	当创建一个对象时，发生一个验证错误
500	INTERNAL SERVER ERROR	*	服务器发生错误，用户将无法判断发出的请求是否成功

Spring Boot实现RESTful API

- Spring Boot提供的spring-boot-starter-web组件完全支持开发RESTful API，提供了与REST操作方式（GET、POST、PUT、DELETE）对应的注解。
- @GetMapping：处理GET请求，获取资源。
- @PostMapping：处理POST请求，新增资源。
- @PutMapping：处理PUT请求，更新资源。
- @DeleteMapping：处理DELETE请求，删除资源。
- @PatchMapping：处理PATCH请求，用于部分更新资源。

Spring Boot实现RESTful API

- 在RESTful架构中，每个网址代表一种资源，所以URI中建议不要包含动词，只包含名词即可，而且所用的名词往往与数据库的表格名对应。
- 用户管理模块API示例：

HTTP Method	接口地址	接口说明
POST	/user	创建用户
GET	/user/id	根据 id 获取用户信息
PUT	/user	更新用户
DELETE	/user/id	根据 id 删除对应的用户

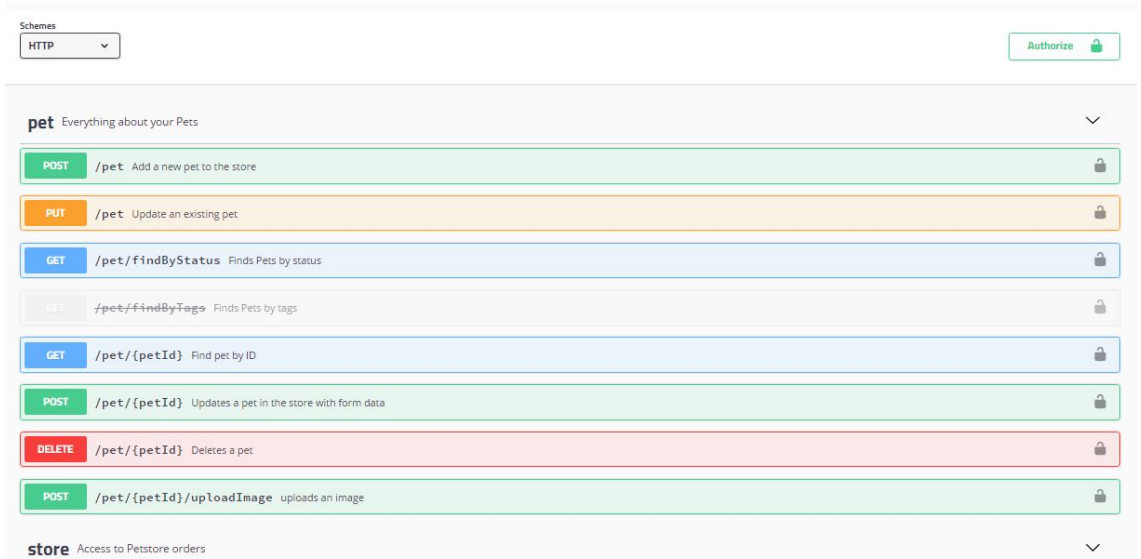
Spring Boot实现RESTful API

```java
@RestController
public class UserController {

    @GetMapping("/user/{id}")
    public String getUserById(@PathVariable int id){
        return "根据ID获取用户";
    }
    @PostMapping("/user")
    public String save(User user){
        return "添加用户";
    }
    @PutMapping("/user")
    public String update(User user){
        return "更新用户";
    }
    @DeleteMapping("/user/{id}")
    public String deleteById(@PathVariable int id){
        return "根据ID删除用户";
    }
}
```

什么是Swagger

- Swagger是一个规范和完整的框架，用于生成、描述、调用和可视化RESTful风格的Web服务，是非常流行的API表达工具。
- Swagger能够自动生成完善的RESTful API文档，，同时并根据后台代码的修改同步更新，同时提供完整的测试页面来调试API。



使用Swagger生成Web API文档

- 在Spring Boot项目中集成Swagger同样非常简单，只需在项目中引入springfox-swagger2和springfox-swagger-ui依赖即可。

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

使用Swagger生成Web API文档

- 在Spring Boot项目中集成Swagger同样非常简单，只需在项目中引入springfox-swagger2和springfox-swagger-ui依赖即可。

```
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

配置Swagger

```java
@Configuration // 告诉Spring容器，这个类是一个配置类
@EnableSwagger2 // 启用Swagger2功能
public class Swagger2Config {
    @Bean
    public Docket createRestApi() {
        return new Docket(DocumentationType.SWAGGER_2)
                .apiInfo(apiInfo()) Docket
                .select() ApiSelectorBuilder
                // com包下所有API都交给Swagger2管理
                .apis(RequestHandlerSelectors.basePackage("com"))
                .paths(PathSelectors.any()).build();
    }
    // API文档页面显示信息
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
                .title("演示项目API") // 标题
                .description("学习Swagger2的演示项目") // 描述
                .build();
    }
}
```

注意事项

- Spring Boot 2.6.X后与Swagger有版本冲突问题，需要在application.properties中加入以下配置：

```
spring.mvc.pathmatch.matching-strategy=ant_path_matcher
```

使用 Swagger2 进行接口测试

- 启动项目访问 http://127.0.0.1:8080/swagger-ui.html ， 即可打开自动生成的可视化测试页面

Swagger常用注解

- Swagger提供了一系列注解来描述接口信息，包括接口说明、请求方法、请求参数、返回信息等

注　解	属　性	值	说　明	示　例
@Api	value	字符串	可用在 class 头上，class 描述	@Api(value = "xxx", description = "xxx")
	description	字符串		
@ApiOperation	value	字符串	可用在方法头上，参数的描述容器	@ApiOperation(value = "xxx", notes = "xxx")
	notes	字符串		
@ApiImplicitParams		@ApiImplicitParam 数组	可用在方法头上，参数的描述容器	@ApiImplicitParams({@ApiImplicitParam1,@ApiImplicitParam2,...})
@ApiImplicitParam	name	字符串 与参数命名对应	可用在 @ApiImplicitParams 中	用例参见项目中的设置
	value	字符串	参数中文描述	
	required	布尔值	true/false	
	dataType	字符串	参数类型	
	paramType	字符串	参数请求方式：query/path	
	defaultValue	字符串	在 API 测试中的默认值	
@ApiResponses	{}	@ApiResponse 数组	可用在方法头上，参数的描述容器	@ApiResponses({@ApiResponse1,@ApiResponse2,...})
@ApiResponse	code	整数	可用在@ApiResponses 中	@ApiResponse(code = 200, message = "Successful")
	message	字符串	错误描述	
@ApiIgnore			忽略这个 API	
@ApiError			发生错误的返回信息	