基于Spring Boot+JPA Restful 风格的数据

Restful简介

Restful是一种软件架构风格，设计风格而**不是**标准，只是提供了一组设计原则和约束条件。它主要用于客户端和服务器交互类的软件。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。

在服务器端，应用程序状态和功能可以分为各种资源。资源是一个有趣的概念实体，它向客户端公开。资源的例子有：应用程序对象、数据库记录、算法等等。每个资源都使用 URI (Universal Resource Identifier) 得到一个唯一的地址。所有资源都共享统一的接口，以便在客户端和服务器之间传输状态。使用的是标准的 HTTP 方法，比如 GET、PUT、POST 和 DELETE。Hypermedia 是应用程序状态的引擎，资源表示通过超链接互联。

在基于Rest的设计中，使用一组通用的动词来操作资源：

- 要创建资源：应使用HTTP POST
- 要检索资源：应该使用HTTP GET
- 要更新资源：应使用HTTP PUT
- 要删除资源：应该使用HTTP DELETE

第一章：使用SpringBoot搭建 RESTful Web Service

1.1、构建开发环境

JDK 1.8或更高版本

Maven的3.0+

Intellij IDEA集成开发环境

Web Service 将使用GET请求 处理 /greeting，我们可以填充 name 参数在请求的字符串中，GET请求将会返回一个 200的状态码，并在响应体中返回JSON数据格式如下：

```
{
    "resultData": [
        {
            "id": 999,
            "content": "Hello, jack!",
            "city": [
                {
                    "code": 1001,
                    "name": "北京"
                },
                {
                    "code": 1002,
                    "name": "上海"
                }
            ]
        }
    ]
```

```
}
```

构建springboot pom.xml的配置

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.yztc.restful</groupId>
   <artifactId>workshop</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>

   <name>workshop</name>
   <description>Demo project for Spring Boot</description>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.6.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
      <java.version>1.8</java.version>
   </properties>

   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-web</artifactId>
      </dependency>
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <version>5.1.40</version>
      </dependency>

      <dependency>
         <groupId>com.jayway.jsonpath</groupId>
```

```
            <artifactId>json-path</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

创建一个POJO类对象，其中包含了id、content和对应的构造方法。

```
package com.yztc.restful.workshop.resource;

import java.util.List;

public class Greeting {

    private long id;
    private String content;
    private List<City> city;

    public Greeting() {

    }

    public void setId(long id) {
        this.id = id;
    }

    public void setContent(String content) {
        this.content = content;
    }

    public Greeting(long id, String content, List<City> city) {
        this.id = id;
        this.content = content;
        this.city = city;
```

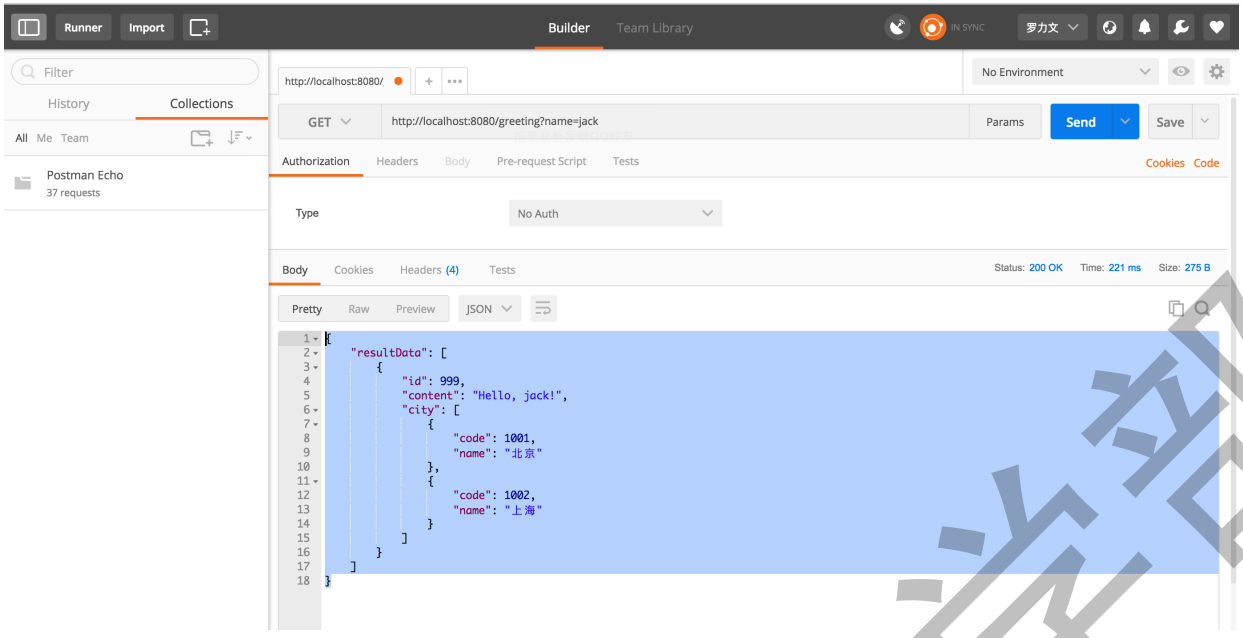
```
    }

    public List<City> getCity() {
        return city;
    }

    public void setCity(List<City> city) {
        this.city = city;
    }

    public long getId() {
        return id;
    }

    public String getContent() {
        return content;
    }

}
```

```
package com.yztc.restful.workshop.resource;

public class City {
    private int code;
    private String name;

    public City(int code, String name) {
        this.code = code;
        this.name = name;
    }

    public int getCode() {
        return code;
    }

    public void setCode(int code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

创建一个 resource controller

```
package com.yztc.restful.workshop.controller;

import com.yztc.restful.workshop.resource.City;
import com.yztc.restful.workshop.resource.Greeting;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

import java.util.*;
import java.util.concurrent.atomic.AtomicLong;


@RestController
public class GreetingController {
    private static final String template = "Hello, %s!";
    private final AtomicLong counter = new AtomicLong();

    @RequestMapping(value = "/greeting",method = RequestMethod.GET)
    public Map<String,List<Greeting>> greeting(@RequestParam(value="name",
defaultValue="World") String name) {
        Map<String,List<Greeting>> map = new HashMap<>();
        List<Greeting> list  =new ArrayList<>();
        List<City> city1 = new ArrayList<>();
        city1.add(new City(1001,"北京"));
        city1.add(new City(1002,"上海"));

        Greeting greeting = new Greeting();
        greeting.setId(999);
        greeting.setContent(
                String.format(template, name));
        greeting.setCity(city1);

        list.add(greeting);
        map.put("resultData",list);
        return  map;
    }

}
```

使用Postman测试的结果如下：



第二章：基于 JPA 的 Restful 风格的数据

2.1、构建 基于Springboot的pom.xml文件

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>

   <groupId>com.spring.restful</groupId>
   <artifactId>spring_ful</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <packaging>jar</packaging>

   <name>spring_ful</name>
   <description>Demo project for Spring Boot</description>

   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>1.5.6.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>

   <properties>
      <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
      <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
      <java.version>1.8</java.version>
   </properties>
```

```
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2.1、配置application.properties属性文件

主要是配置数据库的连接、以及jpa的一些属性

```
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=root
spring.datasource.url=jdbc:mysql://localhost:3306/mydb
#jpa的属性
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
```

2.2、定义Restful风格访问的URL

```
package com.spring.restful.constants;

public class EmpRestURIConstants {
```

```
    public static final String GET_EMP = "/emps";

    public static final String CREATE_EMP = "/emp/create";

    public static final String DELETE_EMP = "/emp/delete/{id}";

    public static final String UPDATE_EMP = "/emp/update/{id}";

    public static final String FIND_EMP = "/emp/findone/{id}";

    public static final String DELETE_ALL = "/emp/deleteall";

}
```

2.3、定义JPA 的Repository接口

```
package com.spring.restful.repository;

import com.spring.restful.domain.Employee;
import org.springframework.data.jpa.repository.JpaRepository;

public interface EmpRepository extends JpaRepository<Employee,Integer> {

}
```

2.4、定义JPA的POJO类(包含注解)

```
package com.spring.restful.domain;

import javax.persistence.*;

@Entity
@Table(name = "employee")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    @Column
    private String name;
    @Column
    private String address;
    @Column
    private String photo;

    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;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhoto() {
        return photo;
    }

    public void setPhoto(String photo) {
        this.photo = photo;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Employee)) return false;

        Employee employee = (Employee) o;

        if (!id.equals(employee.id)) return false;
        if (!name.equals(employee.name)) return false;
        if (!address.equals(employee.address)) return false;
        return photo.equals(employee.photo);
    }

    @Override
    public int hashCode() {
        int result = id.hashCode();
        result = 31 * result + name.hashCode();
        result = 31 * result + address.hashCode();
        result = 31 * result + photo.hashCode();
        return result;
```

```
    }
}
```

2.3、定义Service组件以及实现类

```
package com.spring.restful.service;

import com.spring.restful.domain.Employee;
import org.springframework.data.domain.Example;

import java.util.List;

public interface EmpService {
    public List<Employee> findAllEmps();
    public Employee findEmpByID(Integer id);
    public void deleteEmp(Integer id);
    public void deleteAllEmp();
    public void updateEmp(Employee employee);
    public void saveEmp(Employee employee);
    public boolean existEmp(Example<Employee> example);
}
```

实现类：

```
package com.spring.restful.service;

import com.spring.restful.domain.Employee;
import com.spring.restful.repository.EmpRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Example;
import org.springframework.stereotype.Repository;
import org.springframework.stereotype.Service;

import javax.annotation.Resource;
import java.util.List;

@Service
public class EmpServiceImp implements EmpService {

    @Autowired
    private EmpRepository repository;

    @Override
    public List<Employee> findAllEmps() {
        return repository.findAll();
    }

    @Override
```

```
    public Employee findEmpByID(Integer id) {
        return repository.findOne(id);
    }

    @Override
    public void deleteEmp(Integer id) {
        repository.delete(id);
    }

    @Override
    public void deleteAllEmp() {
        repository.deleteAll();

    }

    @Override
    public boolean existEmp(Example<Employee> example) {
       return repository.exists(example);
    }

    @Override
    public void updateEmp(Employee employee) {
        repository.saveAndFlush(employee);
    }

    @Override
    public void saveEmp(Employee employee) {
        repository.save(employee);
    }
}
```

2.4、定义Controller控制器

```
package com.spring.restful.controller;

import com.spring.restful.constants.EmpRestURIConstants;
import com.spring.restful.domain.Employee;
import com.spring.restful.service.EmpService;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Example;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.util.UriComponentsBuilder;

import java.util.List;
```

```
@RestController
@RequestMapping("/api")
public class EmpController {
    public static final Logger logger =
LoggerFactory.getLogger(EmpController.class);

    @Autowired
    private EmpService service;

    /**
     * @param id
     * @return
     */
    @RequestMapping(value = EmpRestURIConstants.FIND_EMP, method =
RequestMethod.GET)
    public ResponseEntity<?> findEmpByID(@PathVariable("id") Integer id) {
        logger.info("employee id is " + id);
        Employee employee = service.findEmpByID(id);
        if (employee == null) {
            logger.info("employee is not found");
            return new ResponseEntity<Object>("employee is not found",
HttpStatus.NOT_FOUND);
        }
        return new ResponseEntity<Employee>(employee, HttpStatus.OK);
    }

    /**
     * @return
     */
    @RequestMapping(value = EmpRestURIConstants.GET_EMP, method =
RequestMethod.GET)
    public ResponseEntity<?> findAllEmp() {
        List<Employee> list = service.findAllEmps();
        if (list.isEmpty()) {
            return new ResponseEntity(HttpStatus.NO_CONTENT);
        }
        return new ResponseEntity<List<Employee>>(list, HttpStatus.OK);
    }

    /**
     * @param id
     * @return
     */
    @RequestMapping(value = EmpRestURIConstants.DELETE_EMP, method =
RequestMethod.GET)
    public ResponseEntity<?> deleteEmp(@PathVariable("id") Integer id) {
        logger.info("delete employee by id");
        Employee employee = service.findEmpByID(id);
        if (employee == null) {
```

```
            return new ResponseEntity("unable to delete with id:" + id,
HttpStatus.NOT_FOUND);
        }
        service.deleteEmp(id);
        return new ResponseEntity<Employee>(HttpStatus.OK);
    }


    @RequestMapping(value = EmpRestURIConstants.CREATE_EMP, method =
RequestMethod.POST)
    public ResponseEntity<?> createEmp(@RequestBody Employee employee,
UriComponentsBuilder builder) {
        // logger.info("employee:" + employee);
        System.out.println("----->>");
        if (service.existEmp(Example.of(employee))) {
            logger.info("对象已经存在了");
            return new ResponseEntity<Object>("name is exist" +
employee.getName(), HttpStatus.CONFLICT);
        }
        service.saveEmp(employee);

        return new ResponseEntity<Employee>(HttpStatus.OK);
    }

    @RequestMapping(value = EmpRestURIConstants.UPDATE_EMP, method =
RequestMethod.POST)
    public ResponseEntity<?> updateEmp(@PathVariable("id") Integer id,
@RequestBody Employee employee) {
        Employee current = service.findEmpByID(id);
        if (current == null) {
            return new ResponseEntity(HttpStatus.NOT_FOUND);
        }
        current.setAddress(employee.getAddress());
        current.setName(employee.getName());
        current.setPhoto(employee.getPhoto());
        service.saveEmp(current);
        return new ResponseEntity<Employee>(current, HttpStatus.OK);
    }

    @RequestMapping(value = EmpRestURIConstants.DELETE_ALL, method =
RequestMethod.GET)
    public ResponseEntity<?> deleteALlEmp() {
        service.deleteAllEmp();
        return new ResponseEntity<Employee>(HttpStatus.OK);
    }
}
```