SpringBoot 入门

第 24 章

SpringBoot 基础 2

优就业.JAVA 教研室

课程目标

目标 1：运用 SpringBoot Jpa 使用入门

目标 2：运用 SpringBoot 使用 Jpa 按照规定的接口命名方法来实现自动化查询

目标 3：运用 SpringBoot 使用 JPA 使用@Query 实现自定义查询语句

目标 4：运用 SpringBoot 使用 JPA 实现 Rest 风格

目标 5：SpringBoot 使用 Thymeleaf 模板引擎

目标 6：SpringBoot 使用 JPA、Thymeleaf 集成开发应用

一、SpringBoot 使用 JPA 操作数据库

1、JPA 介绍

JPA 顾名思义就是 Java Persistence API 的意思，是 JDK 5.0 注解或 XML 描述对象－关系表的映射关系，并将运行期的实体对象持久化到数据库中。

jpa 具有什么优势？

（1）、标准化

JPA 是 **JCP 组织**发布的 Java EE 标准之一，因此任何声称符合 JPA 标准的框架都遵循同样的架构，提供相同的访问 API，这保证了基于 JPA 开发的企业应用能够经过少量的修改就能够在不同的 JPA 框架下运行。

（2）、容器级特性的支持

JPA 框架中支持大数据集、事务、并发等容器级事务，这使得 JPA 超越了简单持久化框架的局限，在企业应用发挥更大的作用。

（3）、简单方便

JPA 的主要目标之一就是提供更加简单的编程模型：在 JPA 框架下创建实体和创建 Java 类一样简单，没有任何的约束和限制，只需要使用 javax.persistence.Entity 进行注释，JPA 的框架和接口也都非常简单，

没有太多特别的规则和设计模式的要求，开发者可以很容易的掌握。JPA 基于非侵入式原则设计，因此可以很容易的和其它框架或者容器集成。

（4）、查询能力

JPA 的查询语言是面向对象而非面向数据库的，它以面向对象的自然语法构造查询语句，可以看成是 Hibernate **HQL** 的等价物。JPA 定义了独特的 **JPQL**（Java Persistence Query Language），JPQL 是 EJB QL 的一种扩展，它是针对实体的一种查询语言，操作对象是实体，而不是关系数据库的表，而且能够支持批量更新和修改、JOIN、GROUP BY、HAVING 等通常只有 SQL 才能够提供的高级查询特性，甚至还能够支持子查询。

（5）、高级特性

JPA 中能够支持面向对象的高级特性，如类之间的继承、多态和类之间的复杂关系，这样的支持能够让开发者最大限度的使用面向对象的模型设计企业应用，而不需要自行处理这些特性在关系数据库的持久化。

2、SpringBoot 使用 JPA 入门

（1）、添加 jpa 起步依赖以及 mysqljdbc 驱动、连接池 druid 驱动

```
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.1.10</version>
        </dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

注意 jpa 使用不要在依赖 spring-boot-starter-jdbc 和 jpa 起步依赖包有冲突！

（2）、修改 springboot 配置文件 application.yml 增加如下 jpa 配置：

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/springboot-001?serverTimezone=GMT%2B8
    type: com.alibaba.druid.pool.DruidDataSource
```

```
    username: root
    password: 123
    driver-class-name: com.mysql.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

jpa.hibernate.ddl-auto 是 hibernate 的配置属性，其主要作用是：自动创建、更新、验证数据库表结构。该参数的几种配置如下：

• create：每次加载 hibernate 时都会删除上一次的生成的表，然后根据你的 model 类再重新来生成新表，哪怕两次没有任何改变也要这样执行，这就是导致数据库表数据丢失的一个重要原因。

• create-drop：每次加载 hibernate 时根据 model 类生成表，但是 sessionFactory 一关闭,表就自动删除。

• update：最常用的属性，第一次加载 hibernate 时根据 model 类会自动建立起表的结构（前提是先建立好数据库），以后加载 hibernate 时根据 model 类自动更新表结构，即使表结构改变了但表中的行仍然存在不会删除以前的行。要注意的是当部署到服务器后，表结构是不会被马上建立起来的，是要等应用第一次运行起来后才会。

• validate：每次加载 hibernate 时，验证创建数据库表结构，只会和数据库中的表进行比较，不会创建新表，但是会插入新值。

（3）、创建实体

```
package com.offcn.po;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
```

```
@Entity

@Table(name = "tb_person")
@Data //get 、set
@AllArgsConstructor //所有参数的有参数构造函数
@NoArgsConstructor  //无参数构造函数
public class Person {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "name", nullable = true, length = 20)
    private String name;

    @Column(name = "age", nullable = true, length = 4)
    private Integer age;

}
```

JPA 注解：

注解	作用
@Entity	声明类为实体或表
@Table	声明表名
@Basic	指定非约束明确的各个字段
@Embedded	指定类或它的值是一个可嵌入的类的实例的实体的属性
@Id	指定的类的属性，用于识别（一个表中的主键）
@GeneratedValue	指定如何标识属性可以被初始化，例如自动、手动、或从序列表中获得的值
@Transient	指定的属性，它是不持久的，即：该值永远不会存储在数据库中
@Column	指定持久属性栏属性

@SequenceGenerator	指定在@GeneratedValue 注解中指定的属性的值。它创建了一个序列
@TableGenerator	指定在@GeneratedValue 批注指定属性的值发生器。它创造了的值生成的表
@AccessType	这种类型的注释用于设置访问类型。如果设置@AccessType（FIELD），则可以直接访问变量并且不需要 getter 和 setter，但必须为 public。如果设置@AccessType（PROPERTY），通过 getter 和 setter 方法访问 Entity 的变量
@JoinColumn	指定一个实体组织或实体的集合。这是用在多对一和一对多关联
@UniqueConstraint	指定的字段和用于主要或辅助表的唯一约束
@ColumnResult	参考使用 select 子句的 SQL 查询中的列名
@ManyToMany	定义了连接表之间的多对多一对多的关系
@ManyToOne	定义了连接表之间的多对一的关系
@OneToMany	定义了连接表之间存在一个一对多的关系
@OneToOne	定义了连接表之间有一个一对一的关系
@NamedQueries	指定命名查询的列表
@NamedQuery	指定使用静态名称的查询

（4）、数据访问接口

```
package com.offcn.dao;


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


import com.offcn.po.Person;
```

```
public interface PersonRepository extends JpaRepository<Person, Long> {


}
```

（5）、创建 Controller

```
package com.offcn.controller;


import java.util.List;


import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.bind.annotation.RestController;


import com.offcn.po.Person;

import com.offcn.dao.PersonRepository;
```

```
@RestController
@RequestMapping(value="/person")
public class PerconController {

    @Autowired
    PersonRepository personRepository;

    @PostMapping(path="addPerson")
    public void addPerson(@RequestBody Person person) {
        personRepository.save(person);
    }

    @GetMapping(path="getAllPerson")
    public List<Person> getPerson(){
        return personRepository.findAll();
    }

    @DeleteMapping(path="deletePerson")
    public void deletePerson(@RequestParam Long id) {
        personRepository.deleteById(id);
    }
```

```
    @PutMapping(path="updatePerson")
    public void updatePerson(@RequestBody Person person) {
        personRepository.saveAndFlush(person);
    }

}
```

注意：接收传递 json 格式的对象，需要增加注解：@RequestBody

@RequestBody 注解常用来处理 content-type 不是默认的 application/x-www-form-urlcoded 编码的内容，比如说：application/json 或者是 application/xml 等。一般情况下来说常用其来处理 application/json 类型。

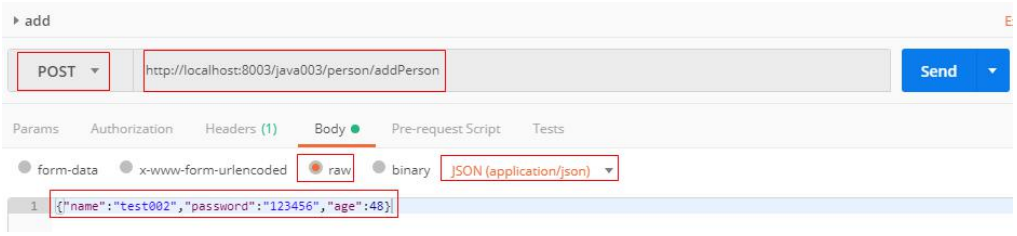
（6）、测试 添加数据

启动应用程序，看到自动创建了数据表

```
2018-10-18 20:46:57.224  INFO 11300 --- [           main] org.hibernate.dialect.Diale
Hibernate: create table hibernate_sequence (next_val bigint) engine=MyISAM
Hibernate: insert into hibernate_sequence values ( 1 )
Hibernate: create table person (id bigint not null, agee integer, name varchar(20), p
```

开启 postman，设置发出 post 请求，请求地址：
http://localhost:8080/person/addPerson

请求参数，选择 body,选择 raw 方式，发送 JSON(application/json)请求



在程序控制台看到：

```
Hibernate: select person0_.id as id1_0_0_, person0_.agee as agee2_0_0_, person0_.name
Hibernate: select next_val as id_val from hibernate_sequence for update
Hibernate: update hibernate_sequence set next_val= ? where next_val=?
Hibernate: insert into person (agee, name, password, id) values (?, ?, ?, ?)
```

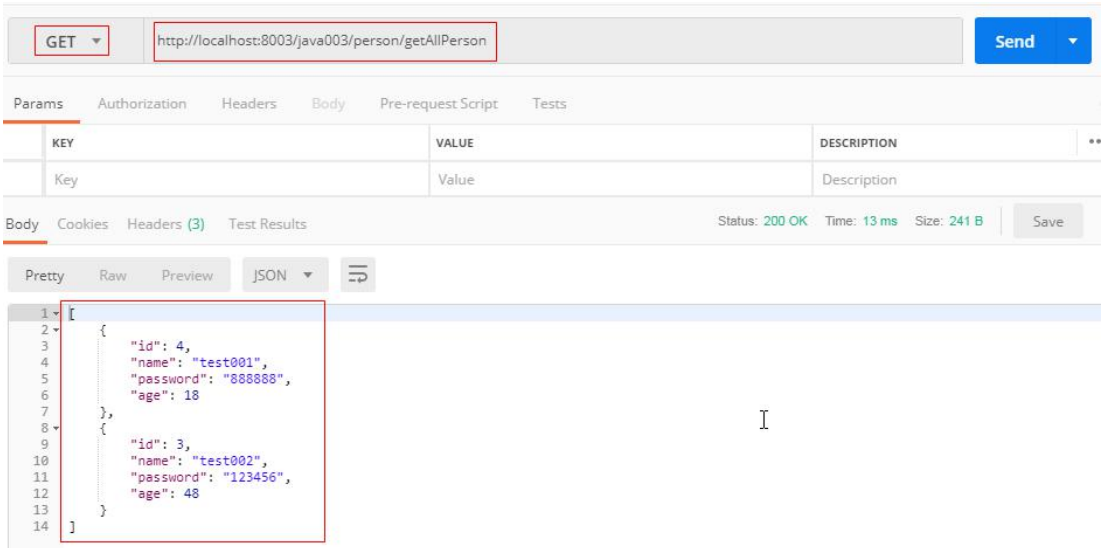
查看数据表：

id	agee	name	password
3	48	test002	123456

（7）、测试 获取数据

开启 postman，设置发出 get 请求，请求地址：

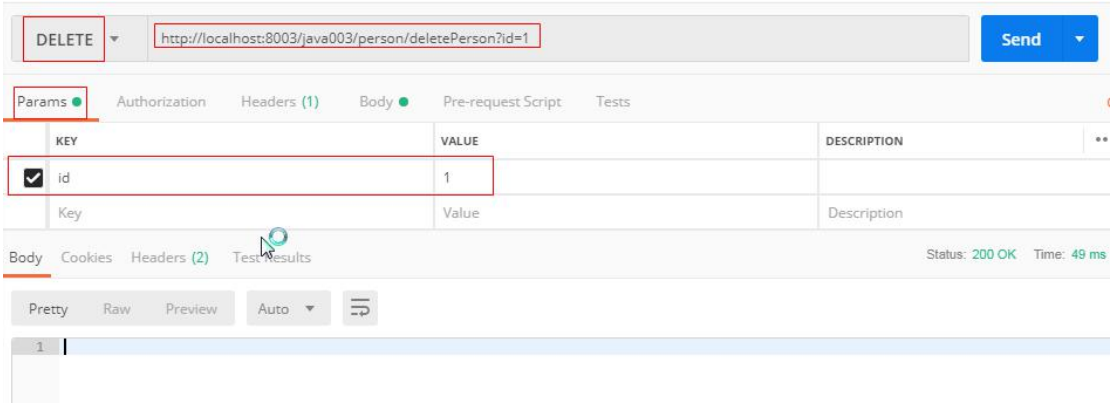
http://localhost:8080/person/getAllPerson



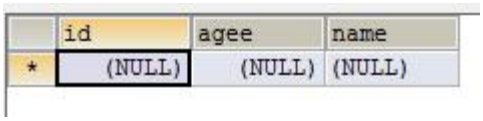
（8）、测试 删除数据

开启 postman，设置发出 delete 请求，请求地址：
http://localhost:8080/person/deletePerson

Parsms 参数设置，key 名称 id value 值 1



查看数据库：

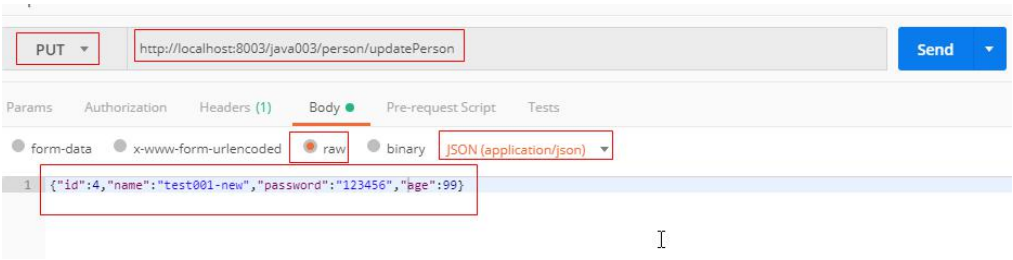


	id	agee	name
*	(NULL)	(NULL)	(NULL)

（9）、测试 修改数据

开启 postman，设置发出 put 请求，请求地址：
http://localhost:8080/person/updatePerson

请求参数，选择 body,选择 raw 方式，发送 JSON(application/json)请求



查看数据库：

	id	agee	name	password
☐	4	99	test001-new	123456

3、SpringBoot 使用 JPA 按照规定的接口命名方法来实现自动化查询

我们通过继承 JpaRepository 接口，除了可以获得上面的基础 CRUD 操作方法之外，还可以通过 Spring 规定的接口命名方法自动创建复杂的 CRUD 操作，以下是我在 Spring Data JPA 文档中找到的命名规则表：

Keyword	Sample	JPQL snippet
And	findBy**LastnameAndFirstname**	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastname**Or**Firstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstname**Is**,findByFirstname**Equals**	… where x.firstname = ?1
Between	findByStartDate**Between**	… where x.startDate between ?1 and ?2
LessThan	findByAge**LessThan**	… where x.age < ?1
LessThanEqual	findByAge**LessThanEqual**	… where x.age <= ?1
GreaterThan	findByAge**GreaterThan**	… where x.age > ?1

Keyword	Sample	JPQL snippet
GreaterThanEqual	findByAge**GreaterThanEqual**	… where x.age >= ?1
After	findByStartDate**After**	… where x.startDate > ?1
Before	findByStartDate**Before**	… where x.startDate < ?1
IsNull	findByAge**IsNull**	… where x.age is null
IsNotNull,NotNull	findByAge**IsNotNull**	… where x.age not null
Like	findByFirstname**Like**	… where x.firstname like ?1
NotLike	findByFirstname**NotLike**	… where x.firstname not like ?1
StartingWith	findByFirstname**StartingWith**	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstname**EndingWith**	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstname**Containing**	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAge**OrderBy**Lastname**Desc**	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?1)

（1）、编辑 PersonRepository，新增如下方法

```
package com.offcn.demo.dao;

import java.util.List;

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

import com.offcn.demo.bean.Person;

public interface PersonRepository extends JpaRepository<Person, Long> {



    //查询指定用户姓名的用户

    public Person findByNameIs(String name);

    //查询指定用户姓名和密码都相同的用户

    public Person findByNameIsAndPassword(String name,String password);

    //查询包含指定名字的用户

    public List<Person> findByNameContaining(String name);

}
```

（2）、新增 Controller 类 Person2Controller

```
package com.offcn.demo.jpa.controller;



import java.util.List;



import org.springframework.beans.factory.annotation.Autowired;
```

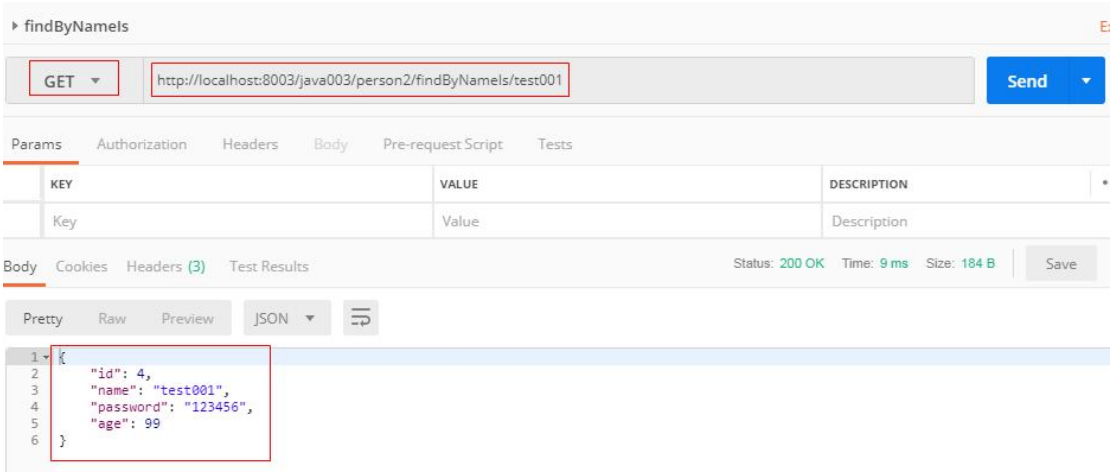
```
import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;



import com.offcn.demo.bean.Person;

import com.offcn.demo.dao.PersonRepository;



@RestController

@RequestMapping("/person2")

public class Person2Controller {



    @Autowired

    PersonRepository personRepository;



    @GetMapping("findByNameIs/{name}")

    public Person findByNameIs(@PathVariable String name) {

        return personRepository.findByNameIs(name);

    }



    @GetMapping("findByNameIsAndPassword/{name}/{password}")

    public Person findByNameIsAndPassword(@PathVariable String
name,@PathVariable String password) {
```

```
        return personRepository.findByNameIsAndPassword(name, password);
    }

    @GetMapping("findByNameContaining/{name}")
    public List<Person> findByNameContaining(@PathVariable String name) {
        return personRepository.findByNameContaining(name);
    }
}
```

（3）、测试 查询指定用户名方法 findByNameIs

开启 **postman，设置发出 get 请求，请求地址：**

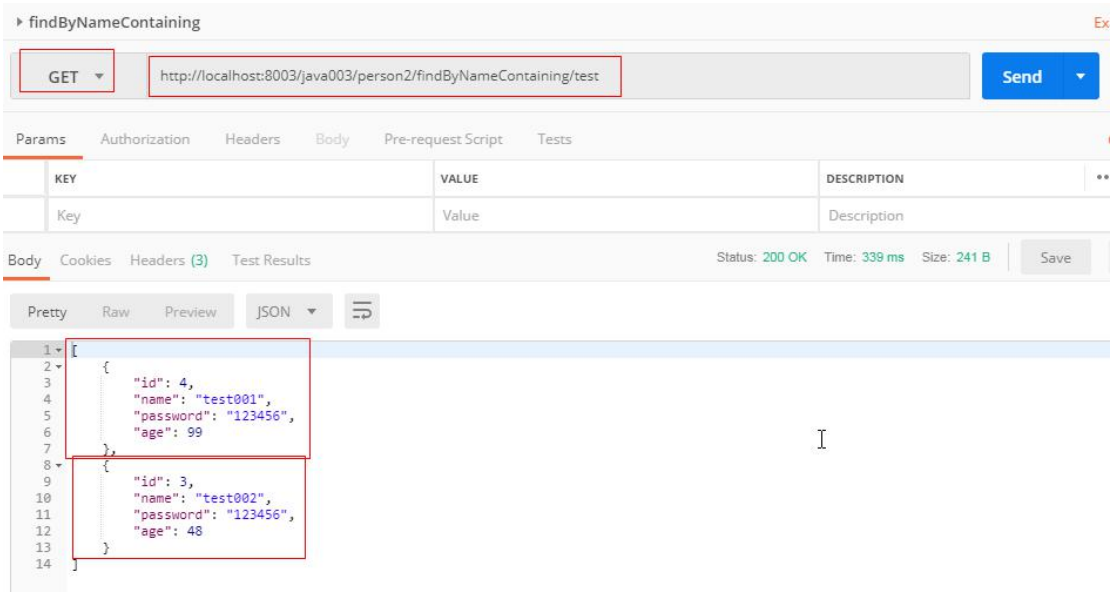
http://localhost:8080/person2/findByNameIs?name=test001



（4）、测试 查询包含指定用户名方法 findByNameContaining

开启 postman，设置发出 get 请求，请求地址：

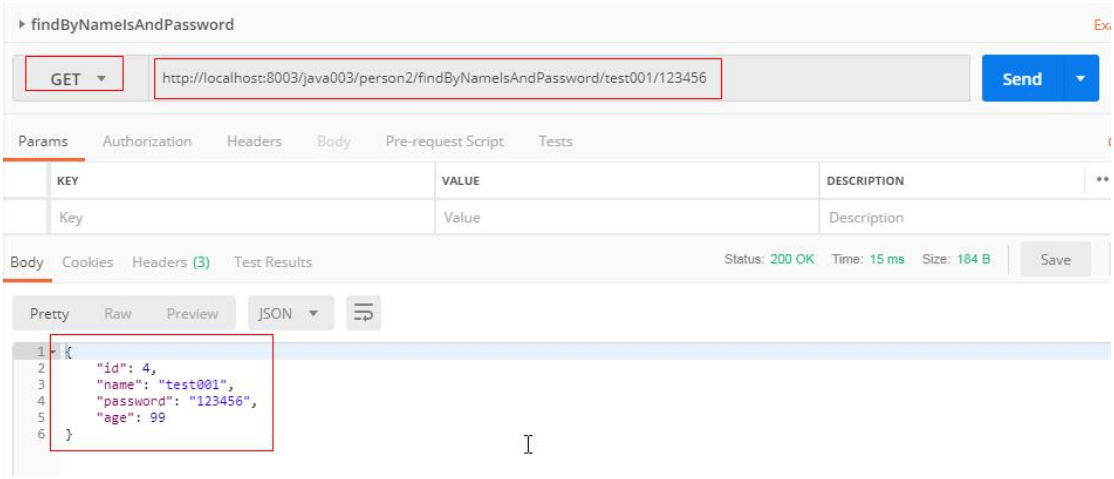
http://localhost:8080/person2/findByNameContaining/test



（5）、测试 查询用户名、密码验证都一致的 用户信息方法 findByNameIsAndPassword

开启 postman，设置发出 get 请求，请求地址：

http://localhost:8080/person2/findByNameIsAndPassword/test001/123456



4、SpringBoot 使用 JPA 使用@Query 实现自定义查询语句

Jpa 提供了非常大的自由度给开发者，我们可以在接口方法中通过定义@Query 注解自定义接口方法的 JPQL 语句。

（1）、编辑 PersonRepository，新增如下方法

```
    //查询指定用户姓名的用户
    @Query("select p from Person p where p.name=:name")
    public Person getPerson(@Param("name") String name);
    //用户登录验证
    @Query("select p from Person p where p.name=?1 and p.password=?2")
    public Person login(@Param("name") String name,@Param("password") String password);

    //模糊查询用户名里面包含指定字符
```

```
    @Query("select p from Person p where p.name like %:name%")
    public List<Person> getNamesLike(@Param("name") String name);


    //查询密码位数是 5 位数的全部用户,使用 mysql 原始 sql 语句进行查询
    @Query(value="select * from person where
length(password)=5",nativeQuery=true)
    public List<Person> getPasswordisFive();
```

注意:@Query 设置查询的 jpsql 语句，是面向对象的语法，类似 HQL；也可以设置 nativeQuery=true 来使用原始 sql 语句。

注意:当 jdk 版本大于 1.8.0_101 @Parm 需要设置，传递给 JPSQL 语句的参数，对应在语句中用 :变量名 来获取变量.

原生 sql 传递多个参数在 sql 里面使用 ?序号 方式获取对应参数，比如 ?1 ?2

（2）、新增 Controller 类 Person3Controller

```
package com.offcn.demo.jpa.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.offcn.demo.bean.Person;

import com.offcn.demo.dao.PersonRepository;
```

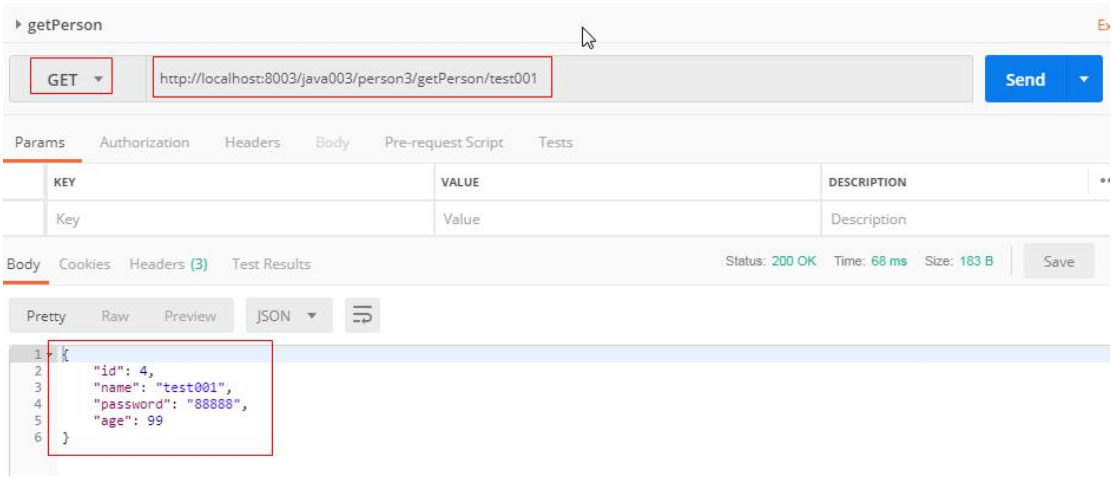
```
@RestController
@RequestMapping("/person3")
public class Person3Controller {
    @Autowired
    PersonRepository personRepository;
    @GetMapping("getPerson/{name}")
    public Person getPerson(@PathVariable String name) {
        return personRepository.getPerson(name);
    }

    @GetMapping("login/{name}/{password}")
    public Person login(@PathVariable String name,@PathVariable String
password) {
        return personRepository.login(name, password);
    }

    @GetMapping("getNamesLike/{name}")
    public List<Person> getNamesLike(@PathVariable String name) {
        return personRepository.getNamesLike(name);
    }

    @GetMapping("getPasswordisFive")
```

```
    public List<Person> getPasswordisFive() {

        return personRepository.getPasswordisFive();

    }

}
```

（3）、测试 查询指定用户名方法 getPerson

开启 postman，设置发出 get 请求，请求地址：

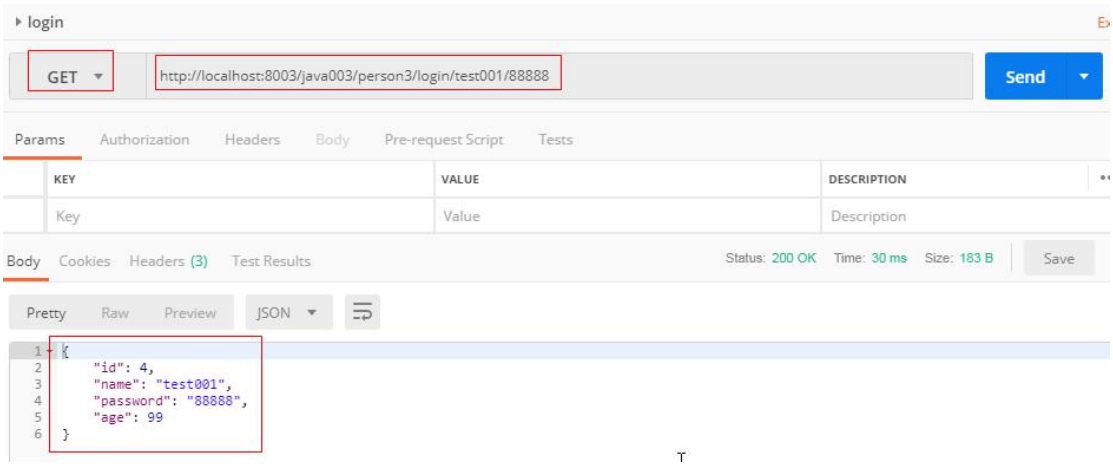
http://localhost:8080/java003/person3/getPerson/test001



（4）、测试 用户登录验证方法 login

开启 postman，设置发出 get 请求，请求地址：

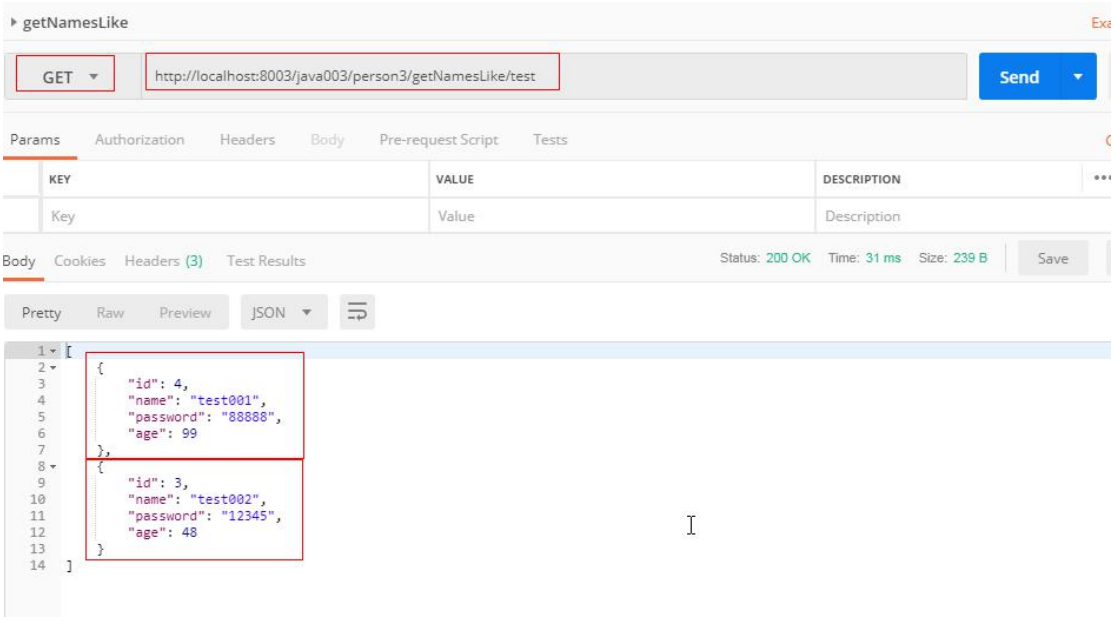
http://localhost:8080/person3/login/test001/123456



（4）、测试 模糊查询用户名里面包含指定字符 方法 getNamesLike

开启 postman，设置发出 get 请求，请求地址：

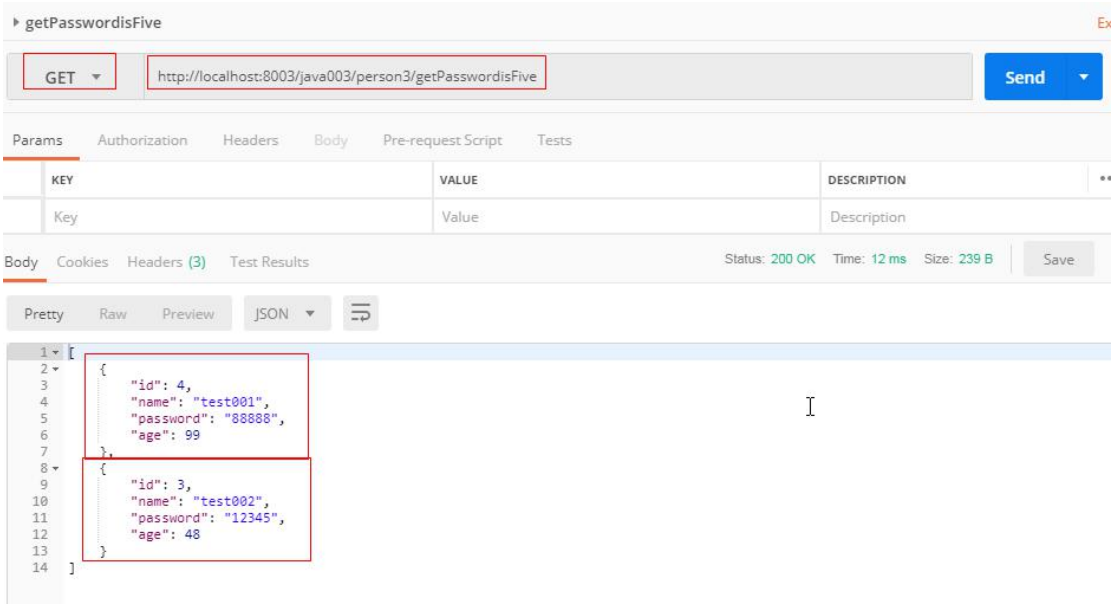
http://localhost:8080/person3/getNamesLike/test



（5）、测试 查询密码位数是 5 位数的全部用户,使用 mysql 原始 sql 语句进行查询 方法 getPasswordisFive

开启 postman，设置发出 get 请求，请求地址：

http://localhost:8080/person3/getPasswordisFive



5、SpringBoot 使用 JPA 实现 Rest 风格数据处理

前面我们基于 jdbc、mybatis 整合 springBoot 来实现了 rest 风格的数据处理，接下来我们使用 JPA 来处理 rest 风格。

（1）、添加 jpa 起步依赖以及 mysqljdbc 驱动、连接池 druid 驱动

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-rest</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
    <groupId>com.alibaba</groupId>
```

```
    <artifactId>druid</artifactId>
    <version>1.1.10</version>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
</dependency>
```

（2）、修改 springboot 配置文件 application.yml 增加如下 jpa 配置：

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/springboot-001?serverTimezone=GMT%2B8
    type: com.alibaba.druid.pool.DruidDataSource
    username: root
    password: 123
    driver-class-name: com.mysql.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
```

（3）、创建实体

```
package com.offcn.po;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
@Data //get 、set
@AllArgsConstructor //所有参数的有参数构造函数
@NoArgsConstructor  //无参数构造函数
```

```
public class Person {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "name", nullable = true, length = 20)
    private String name;

    @Column(name = "age", nullable = true, length = 4)
    private int age;

}
```

（4）、数据访问接口

```
package com.offcn.dao;

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

import com.offcn.po.Person;

public interface PersonRepository extends JpaRepository<Person, Long> {

}
```

（5）、启动主程序准备测试

经过如上几步，一个 Restful 服务就构建成功了，可能大家会问，老师你“什么也没有写啊！”，是的接下来就是见证 springBoot 整合 JPA 的奇迹的时刻，跟我一起数 123，启动程序！

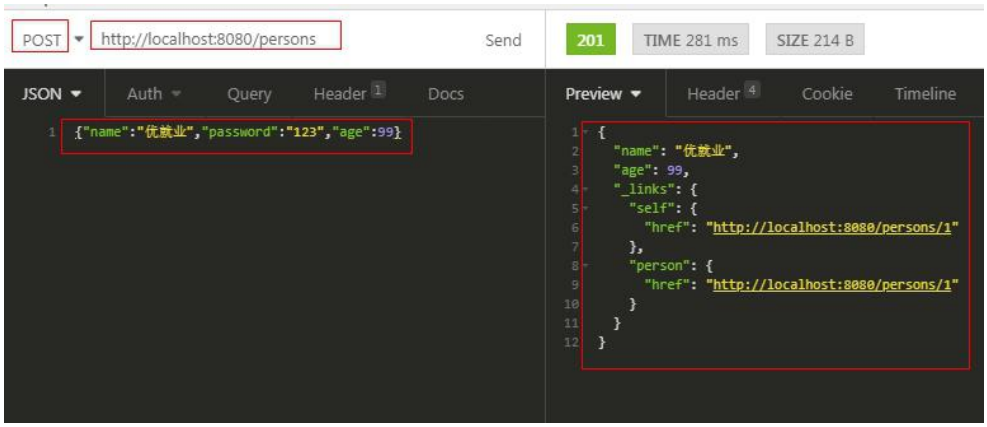
（5.1）、数据新增测试

开启 postman，设置发出 post 请求，请求地址：http://localhost:8080/persons

请求参数，选择 body,选择 raw 方式，发送 JSON(application/json)请求

```
{"name":"优就业","age":99}
```

查看数据，发现数据已经成功插入到数据了！



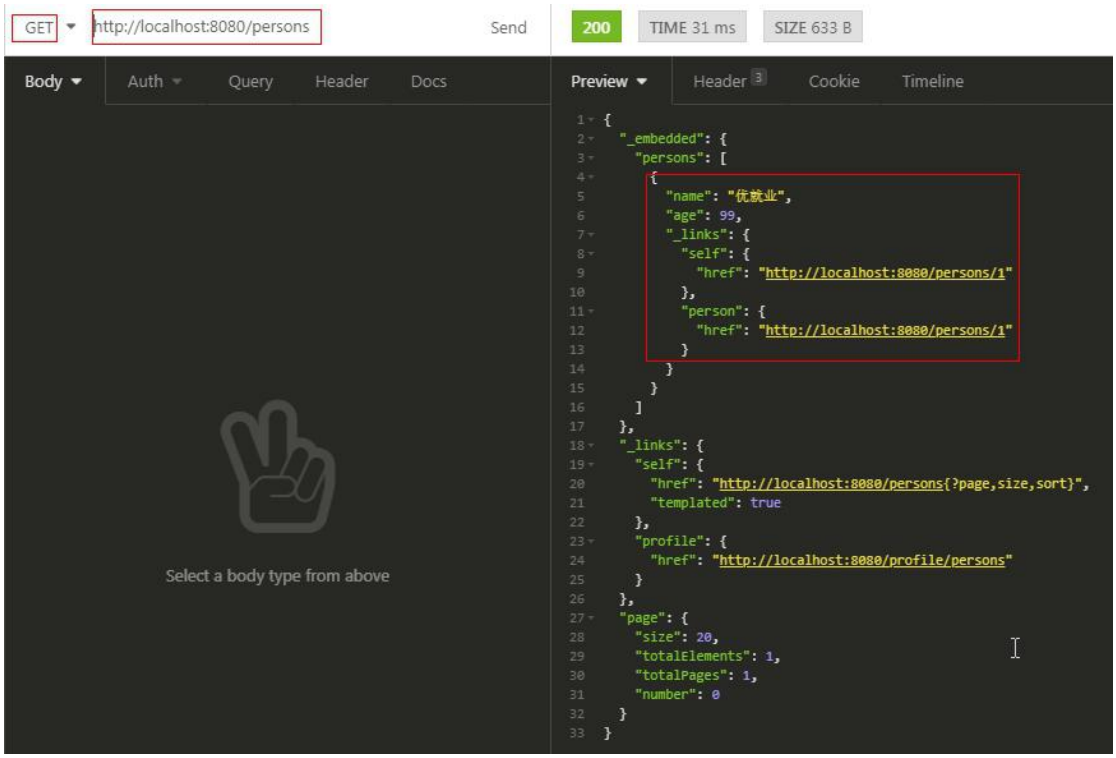
	id	agee	name
☐	1	99	优就业
*	(NULL)	(NULL)	(NULL)

（5.2）、数据查询测试，分页查询

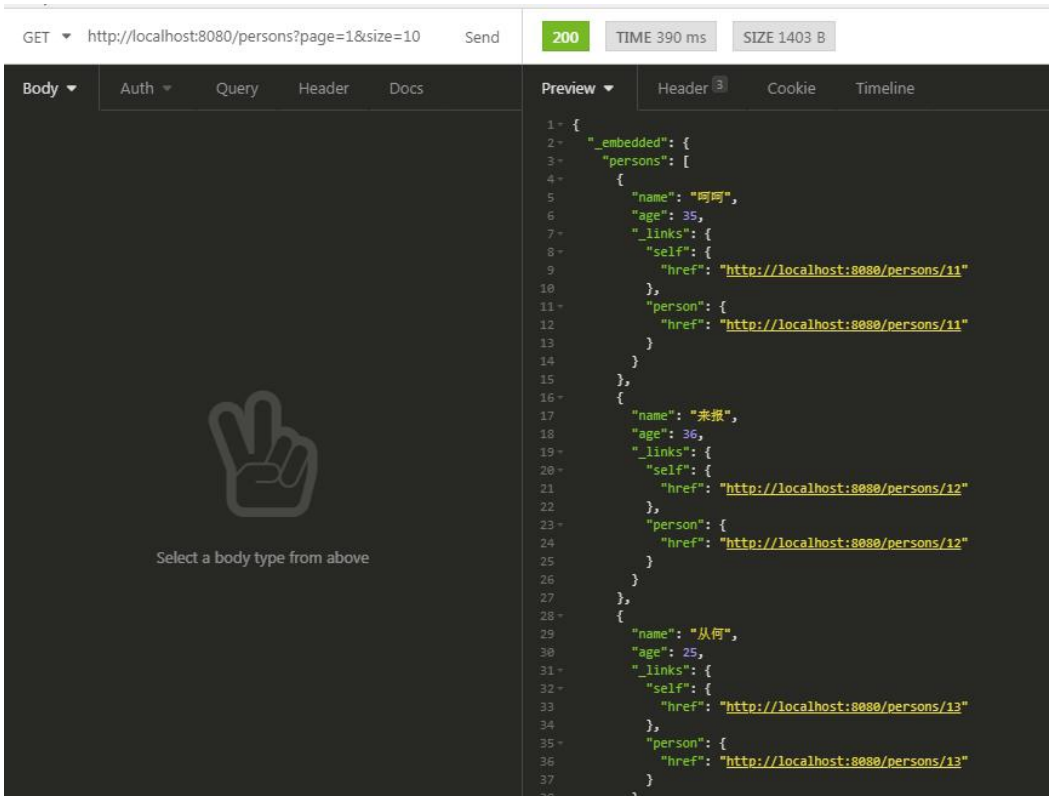
1、查询是 GET 请求，查询请求路径为/persons,请求 URL 如下：

http://localhost:8080/persons

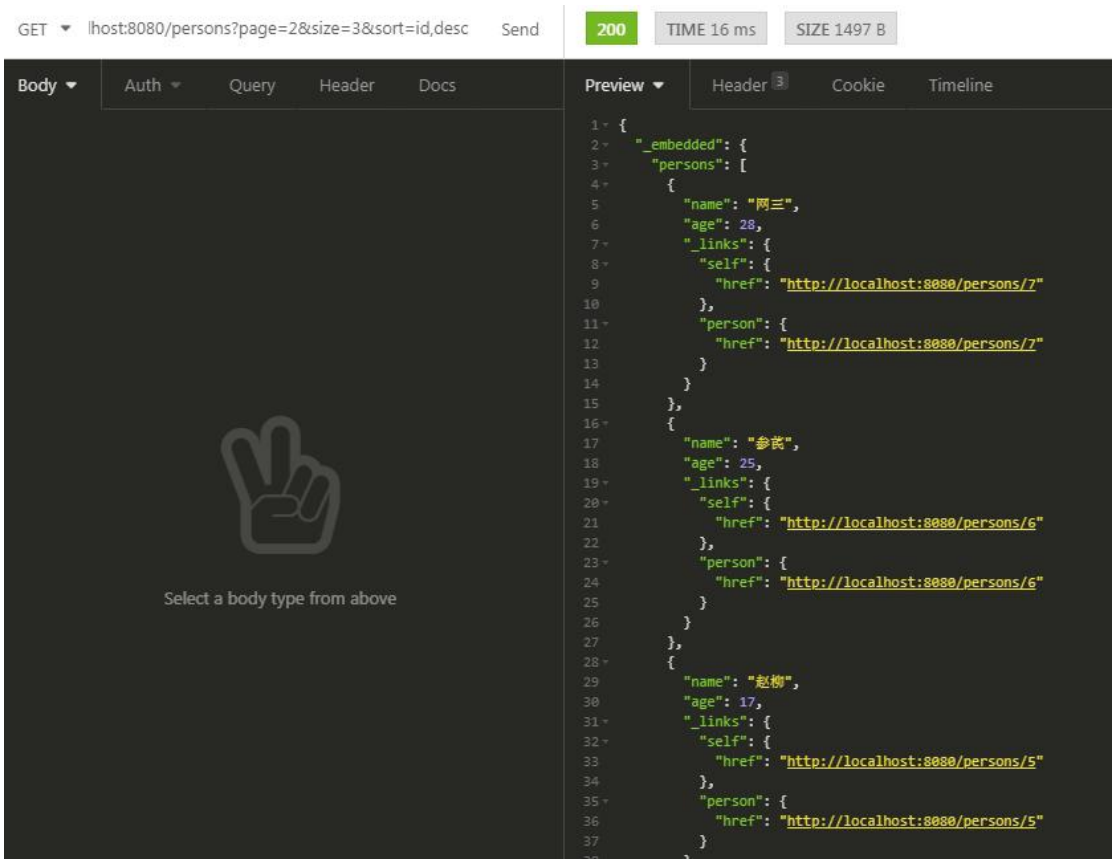
分页查询默认每页 20 条记录，页数为 0



2、还可以发出一个带指定页码、每页记录数的分页查询请求，只需要在请求地址中带上相关参数即可，请求 URL 为：http://localhost:8080/persons?page=1&size=10



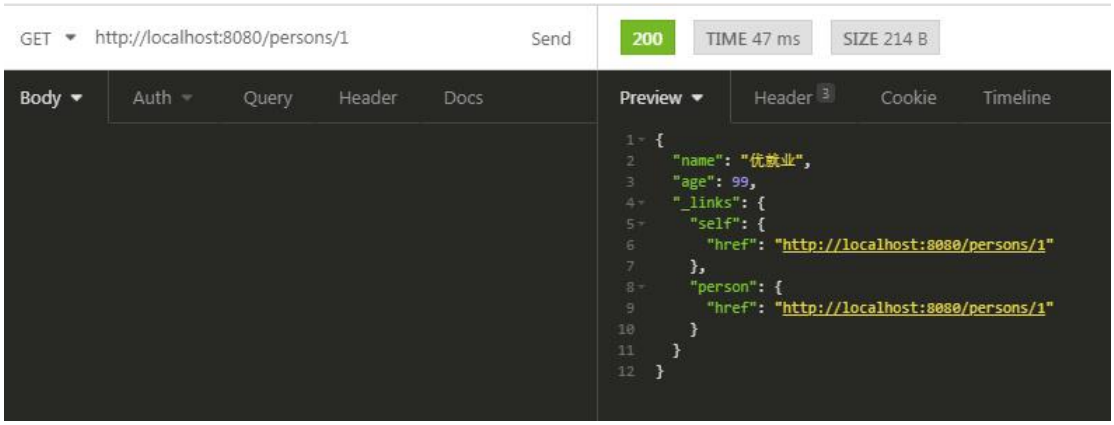
3、除了分页外，默认还支持排序，例如想查询第 2 页数据，每页记录数为 3，并且按照 id 倒序排序，请求 URL 为：http://localhost:8080/persons?page=2&size=3&sort=id,desc



（5.3）、数据查询测试，查询指定 id 的数据

如果按照 id 查询，只需要在/persons 后面追加上 id 即可，例如查询 id 为 1 的 person

，请求 URL 为:http://localhost:8080/persons/1



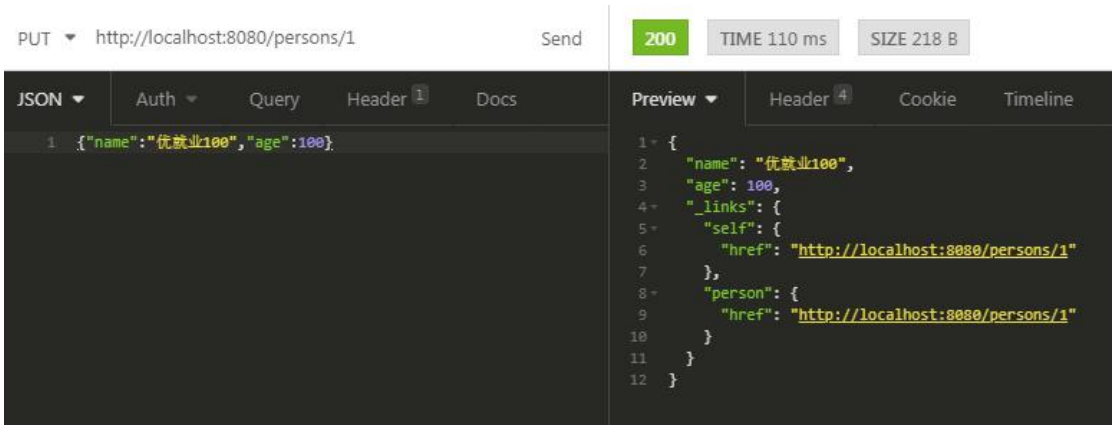
（5.4）、数据修改测试

发送 PUT 请求可实现对数据的修改，对数据的修改是通过 id 进行的，请求路径中要有 id，比如修改 id 为 2 的记录，具体请求 URL 地址：http://localhost:8080/persons/2

请求参数，选择 body,选择 raw 方式，发送 JSON(application/json)请求

```
{"name":"优就业 100","age":100}
```

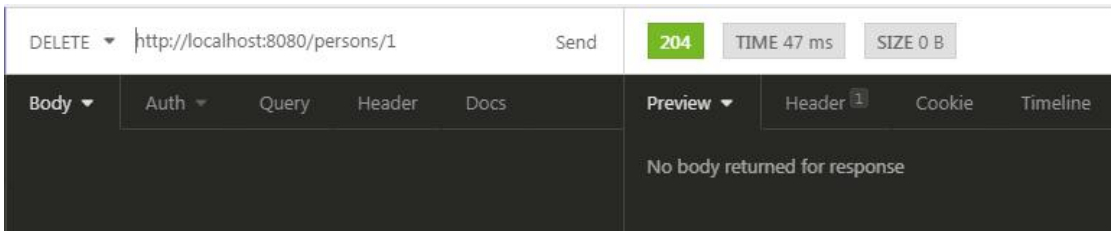
PUT 请求返回的结果就是被修改之后的记录



（5.5）、数据删除测试

发送 DELETE 请求可以实现对数据的删除操作，列入删除 id 为 1 的记录，请求 URL 如下：http://localhost:8080/persons/1

DELETE 请求没有返回值，上面请求发送成功后，id 为 1 的记录就被删除。



（6）、自定义请求路径

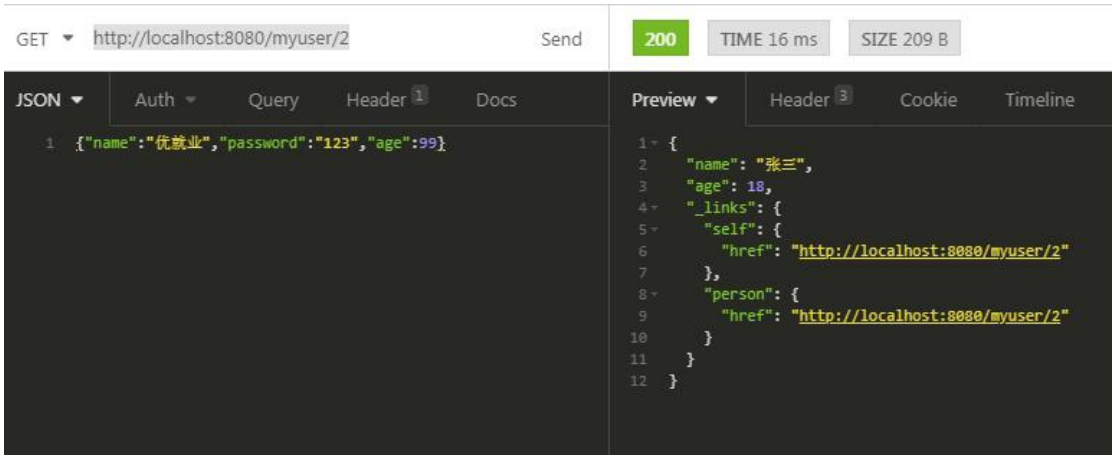
默认情况下，请求路径都是实体名小写加 s，如果开发者想对请求路径进行自定义，通过注

解@RepositoryRestResource 注解即可实现，下面我们在 PersonRepository 上增加注解

```
@RepositoryRestResource(path="myuser")
public interface PersonRepository extends JpaRepository<Person, Long> {

}
```

请求路径即可变为：

http://localhost:8080/myuser/2



（7）、自定义查询方法

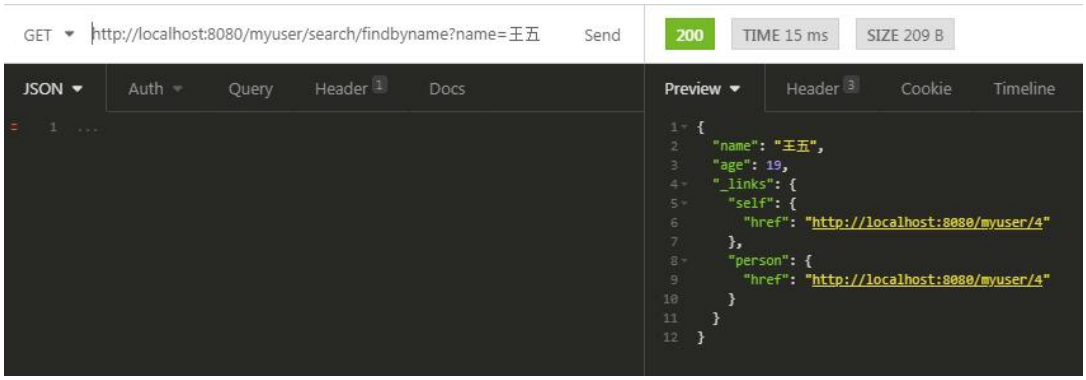
默认的查询方法支持分页查询、排序查询以及按照 id 查询，如果开发者想要按照某个属性查询，只需要在 PersonRepository

中定义相关方法并保罗出去即可，代码如下：

```
@RepositoryRestResource(path="myuser")
public interface PersonRepository extends JpaRepository<Person, Long> {
    @RestResource(path="findbynames")
    List<Person> findByNameContains(@Param("name") String name);

    @RestResource(path="findbyname")
    Person findByNameEquals(@Param("name") String name);
}
```

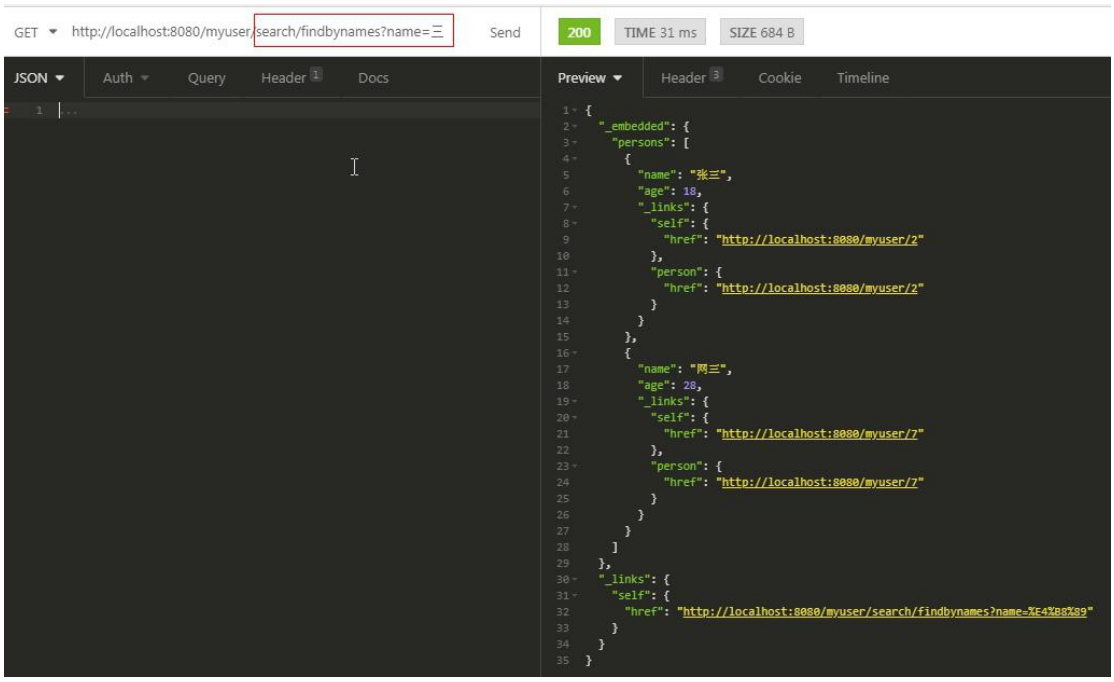
根据用户名精确查询，请求地址：

http://localhost:8080/myuser/search/findbyname?name=王五



根据用户名模糊查询，请求地址：

http://localhost:8080/myuser/search/findbynames?name=三



（8）、隐藏方法

默认情况下，凡是继承了 Respository 接口的的类都会被暴露出来，如果开发者继承了 Respository 接口

又不想暴露出来，在方法上加如下注解配置即可

@RepositoryRestResource(exported=false),加上该注解后整个类里面的全部方法都会隐藏

如果仅仅是向隐藏其中的某些方法，直接在具体方法上使用注解

@RestResource(exported=false)即可

（9）、配置跨域

直接在实现了 Respository 接口的类上增加跨域注解@CrossOrigin 即可

二、SpringBoot 使用 Thymeleaf 模板引擎

1、Thymeleaf 模板介绍

Thymeleaf 是一个 Java 类库，它是一个 xml/xhtml/html5 的模板引擎，可以作为 MVC 的 Web 应用的 View 层。Thymeleaf 还提供了额外的模块与 Spring MVC 集成，因此我们可以使用 Thymeleaf 完全替代 JSP。Thymeleaf 是面向 Web 和独立环境的现代服务器端 Java 模板引擎。Thymeleaf 的主要目标是为您的开发工作流程带来优雅的自然模板,可以在浏览器中正确显示 HTML，还可以作为静态原型工作，从而在开发团队中进行更强大的协作。使用 Spring Framework 的模块，与您最喜爱的工具进行大量集成，以及插入自己的功能的能力，Thymeleaf 是现代 HTML5 JVM Web 开发的理想选择。

2、Spring boot 集成 Thymeleaf 模板入门实例

Thymeleaf 模板是 SpringBoot 官方推荐的模板引擎，下面我们就带领大家来体验使用。

（1）、修改 maven 依赖，增加 Thymeleaf 所需依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

（2）、修改 application.yml 增加 Thymeleaf 相关配置

```
spring:
  #开始 thymeleaf 设置
  thymeleaf:
  #禁用模板缓存
    cache: false
```

（3）、编写 Controller 文件

```
package com.offcn.demo.controller;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

@Controller

public class FirstThymeleafController {

  /**

       * 访问 http://localhost:8080/first

       * 将数据 message 填充到 templates/index.html

    * @param model

    * @return

    */

    @GetMapping("/first")

    public String indexPage(Model model) {

        String message = "Hello, Thymeleaf!";

        model.addAttribute("message", message);

        return "index";

    }

}
```

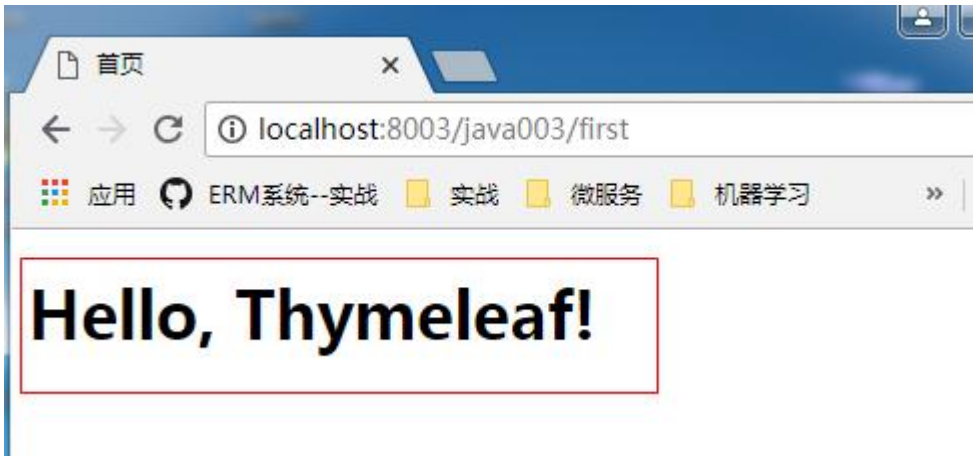
（4）、编写模板文件

在 resource/templates 下新建 index.html

```
<!doctype html>

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:th="http://www.thymeleaf.org">

<head>

    <meta charset="UTF-8">

    <title>首页</title>

</head>

<body>

    <h1 th:text="${message}"></h1>

</body>

</html>
```

通过 类似 EL 表达式将 Model 中的数据填充到 h1 标签中

（5）、运行访问地址 http://localhost:8080/first



3、Spring boot 集成 Thymeleaf 常量文字读取

（1）、修改 application.yml，增加国际化消息属性设置

```
spring:
 #设置文字消息
   messages:
        encoding: UTF-8
        basename: message_zh_CN
```

（2）、增加中文消息设置属性文件 message_zh_CN.properties 【注意文件编码是 UTF-8】

```
title=这是标题
message1=这是消息 2
message2=这是消息 2
```

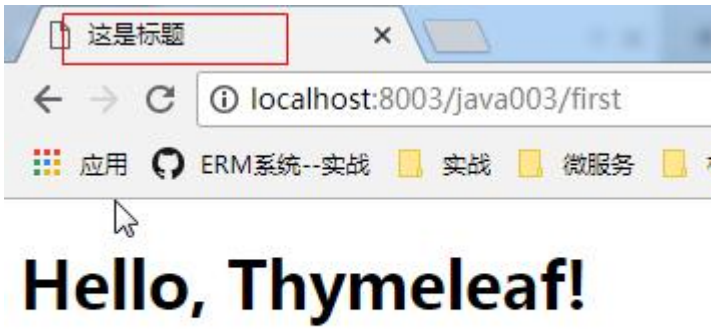
（3）、编写模板文件

在 resource/templates 下编辑 index.html

```
<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title th:text="#{title}"></title>
</head>
<body>
    <h1 th:text="${message}"></h1>
```

```
</body>

</html>
```

（4）、访问运行地址 http://localhost:8080/java003/first，测试结果如下：



4、Spring boot 集成 Thymeleaf 打印对象属性

（1）、新建一个实体 bean　User，内容如下：

```
package com.offcn.demo.bean;

public class User {

private Integer id;

private String name;

private int age;

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 int getAge() {

    return age;

}

public void setAge(int age) {

    this.age = age;

}




}
```

（2）、新建一个 Controller，内容如下：

```
package com.offcn.demo.controller;
```

```
import java.util.HashMap;

import java.util.Map;



import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;



import com.offcn.demo.bean.User;



@Controller

public class SecondThymeleafController {



    /**

     * 访问 localhost:8080 页面

     * 将数据 message 填充到 templates/index2.html

  * @param model

  * @return

  */

  @GetMapping("/second")

  public String indexPage(Model model) {

      String message = "Hello, Thymeleaf!";



      User u = new User();
```

```
        u.setId(1);

        u.setName("优就业");

        u.setAge(18);



        Map<String,Object> map=new HashMap<>();

        map.put("src1","1.jpg");

        map.put("src2","2.jpg");




        model.addAttribute("message", message);

        model.addAttribute("user", u);

        model.addAttribute("src", map);

        return "index2";

    }

}
```

（3）、在 resource/templates 下,新增模板文件 index2.html

```
<!doctype html>

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:th="http://www.thymeleaf.org">

<head>

    <meta charset="UTF-8">
```

```
    <title>首页</title>
</head>
<body>
    <h1 th:text="${message}"></h1>


    <img th:src="${src.src1}"/>
    </br>
    <img th:src="${src.src2}"/>
    </br>
    <span th:text="${user.id}"></span>
    <span th:text="${user.name}"></span>
    <span th:text="${user.age}"></span>



</body>
</html>
```

（4）、访问地址：http://localhost:8080/second 运行结果如下

5、Spring boot 集成 Thymeleaf 循环遍历集合

（1）、新建一个 Controller，内容如下：

```
package com.offcn.demo.controller;



import java.util.ArrayList;

import java.util.List;



import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;



import com.offcn.demo.bean.User;
```

```
@Controller
public class ThreeThymeleafController {


    /**
     * 访问 localhost:8080/java003 页面
     * 将数据 message 填充到 templates/index3.html
  * @param model
  * @return
  */
  @GetMapping("/three")
  public String indexPage(Model model) {
      List<User> list=new ArrayList<User>();
      User u1 = new User();
      u1.setId(1);
      u1.setName("优就业");
      u1.setAge(18);
      list.add(u1);


      User u2 = new User();
      u2.setId(2);
      u2.setName("中公教育");
      u2.setAge(28);
```

```
        list.add(u2);

        User u3 = new User();

        u3.setId(3);

        u3.setName("IT 先锋");

        u3.setAge(88);

        list.add(u3);



        User u4 = new User();

        u4.setId(4);

        u4.setName("JAVA 第一");

        u4.setAge(888);

        list.add(u4);



        model.addAttribute("userList", list);

        return "index3";

    }

}
```

(2) 、在 resource/templates 下,新增模板文件 index3.html

```
<!doctype html>

<html xmlns="http://www.w3.org/1999/xhtml"

      xmlns:th="http://www.thymeleaf.org">
```

```
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
   <table width="200" style="text-align: center;">
        <tr>
            <th>编号</th>
            <th>姓名</th>
            <th>年龄</th>
            <th>index</th>
        </tr>
        <tr th:each="user,iterStat : ${userList}">
            <td th:text="${user.id}"></td>
            <td th:text="${user.name}"></td>
            <td th:text="${user.age}"></td>
            <td th:text="${iterStat.index}">index</td>
        </tr>
    </table>
</body>
</html>
```

iterStat 称作状态变量，属性有：

index：当前迭代对象的 index（从 0 开始计算）

count：当前迭代对象的 index（从 1 开始计算）

size：被迭代对象的大小

current：当前迭代变量

even/odd：布尔值，当前循环是否是偶数/奇数（从 0 开始计算）

first：布尔值，当前循环是否是第一个

last：布尔值，当前循环是否是最后一个

(3)、访问地址：http://localhost:8080/java003/three 运行结果如下

6、Spring boot 集成 Thymeleaf 赋值、字符串拼接

（1）、新建一个 Controller，内容如下：

```
package com.offcn.demo.controller;



import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;
```

```
@Controller
public class FourThymeleafController {

    /**
     * 访问 localhost:8080/java003 页面
     * 将数据 message 填充到 templates/index4.html
  * @param model
  * @return
  */
  @GetMapping("/four")
  public String indexPage(Model model) {

      model.addAttribute("userName", "优就业");
      model.addAttribute("href", "http://www.ujiuye.com");
      return "index4";
  }
}
```

（2）、在 resource/templates 下,新增模板文件 index4.html

```
<!doctype html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
<!-- 给标签赋值 th:text -->
    <h1 th:text="${userName}"></h1>
<!-- 给属性赋值 th:value、th:属性名称 -->
     <input type="text" name="names" th:value="${userName}"/>
     </br>
     <em th:size="${userName}"></em>
<!-- 字符串拼接 -->
    <span th:text="'欢迎来:'+${userName}+'学习!'"></span>
    </br>
<!-- 字符串拼接,方式 2 -->
    <span th:text="|欢迎来:${userName}学习!|"></span>
</body>
</html>
```

（3）、访问地址：http://localhost:8080/four 运行结果如下

7、Spring boot 集成 Thymeleaf 条件判断、选择语句

（1）、新建一个 Controller，内容如下：

```
package com.offcn.demo.controller;



import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;



@Controller

public class FiveThymeleafController {

    /**

     * 访问 localhost:8080/ 页面

     * 将数据 message 填充到 templates/index4.html

  * @param model

  * @return
```

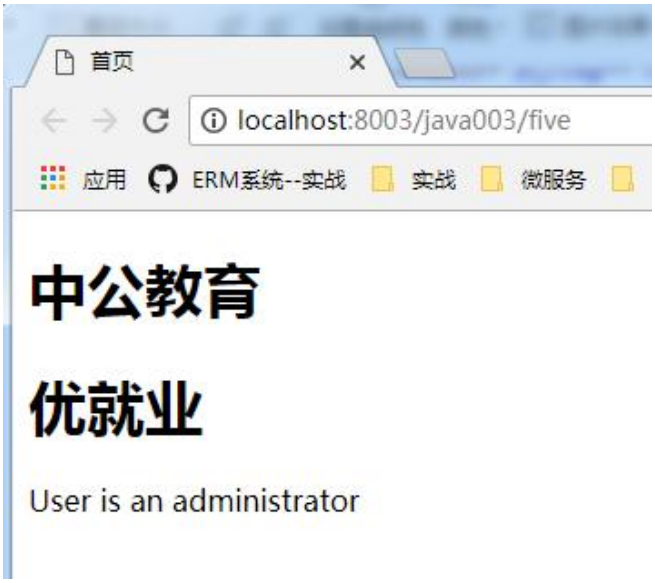
```
 */
 @GetMapping("/five")
 public String indexPage(Model model) {


     model.addAttribute("flag", "yes");
     model.addAttribute("menu", "admin");
     model.addAttribute("manager", "manager");
     return "index5";
 }
}
```

（2）、在 resource/templates 下,新增模板文件 index5.html

```
<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>首页</title>
</head>
<body>
    <!-- th:if 条件成立就显示 -->
    <h1 th:if="${flag=='yes'}" >中公教育</h1>
```

```
    <!-- th:unless 条件不成立就显示 -->
    <h1 th:unless="${flag=='no'}" >优就业</h1>
    <!-- switch 选择语句 -->
  <div th:switch="${menu}">
 <p th:case="'admin'">User is an administrator</p>
 <p th:case="${manager}">User is a manager</p>
</body>
</html>
```

（3）、访问地址：http://localhost:8080/five 运行结果如下



8、Spring boot 集成 Thymeleaf 静态资源加载

我们知道一个网页中加载的静态文件通常有一个十分尴尬的问题，比如对于 bootstrap.css，就是如果我们能让 IDE 识别这个文件，那么我们得用相对路径来引入这个文件。这样我们的 IDE 才能加载到这个文件，并且给予我们相应的提示。但是如果我们想要在发布后服务器能够加载这个文件，我们就必须用相对于 resources 或者 static 的位置来引入静态文件。显然，一般情况下我们不能兼顾这两个问题，只能要么在编写的时候用相对自己的路径，然后在发布的时候用相对于项目资源文件夹的路径，要么就只能放弃 IDE 的提示，非常尴尬。

而在 Thymeleaf 中，我们可很好的处理这一点。在引入资源的时候，我们可以写类似下面的代码:

```
<link rel="stylesheet" type="text/css" media="all"  href="../../css/gtvg.css"
th:href="@{/css/gtvg.css}" />
```

当我们在没有后台渲染的情况下，浏览器会认得 href，但是不认得 th:href，这样它就会选择以相对与本文件的相对路径去加载静态文件。而且我们的 IDE 也能识别这样的加载方式，从而给我们提示。

当我们在有后台渲染的情况下，后台会把这个标签渲染为这样:

```
<link rel="stylesheet" type="text/css" media="all" href="/css/gtvg.css"  />
```

原来的 href 标签会被替换成相对于项目的路径，因此服务器就能找到正确的资源，从而正确渲染。

非常的智能而且方便。

这里需要注意到所有的路径我们是用”@{}”来引用，而不是”${}”，因为后者是用来引用变量名的，而前者是引用路径的，因此我们在这里用的是前者。可是如果我们是把路径写在变量里，那么就要用后者来引用了

9、Spring boot 集成 Thymeleaf 片段 fragment 定义使用

thymeleaf 也提供了类似 import 的东西，可以将很多代码块抽象成模块，然后在需要的时候引用，非常方便。

fragment 介绍

fragment 类似于 JSP 的 tag，在 html 中文件中，可以将多个地方出现的元素块用 fragment 包起来使用。

fragment 使用

定义 fragment

所有的 fragment 可以写在一个文件里面，也可以单独存在，例如：

（1）、在 resource/templates 下,新增模板文件 footer.html

```
<body>
    <h1 th:fragment="copy">
      &copy; 1999-2018 Offcn.All Rights Reserved
    </h1>
</body>
```

在 Springboot 中，默认读取 thymeleaf 文件的路径是：src/main/resource/templates

（2）、编写 Controller

```
package com.offcn.demo.controller;


import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;


@Controller
public class SixThymeleafController {
    /**
     * 访问 localhost:8080/java003 页面
     * 将数据 message 填充到 templates/index6.html
  * @param model
  * @return
  */
```

```
  @GetMapping("/six")
  public String indexPage(Model model) {


      return "index6";
  }
}
```

（3）、在 resource/templates 下,新增视图文件 index6.html

```
<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>首页</title>


</head>
<body>
<!-- 把片段的内容插入到当前位置 -->
  <div th:insert="~{footer :: copy}"></div>


  </br>
  <!-- 使用片段的内容替换当前标签 -->
```

```
  <div th:replace="~{footer :: copy}"></div>
  </br>
  <!-- 保留自己的主标签，不要片段的主标签 -->
   <div th:include="~{footer :: copy}"></div>
</body>
</html>
```

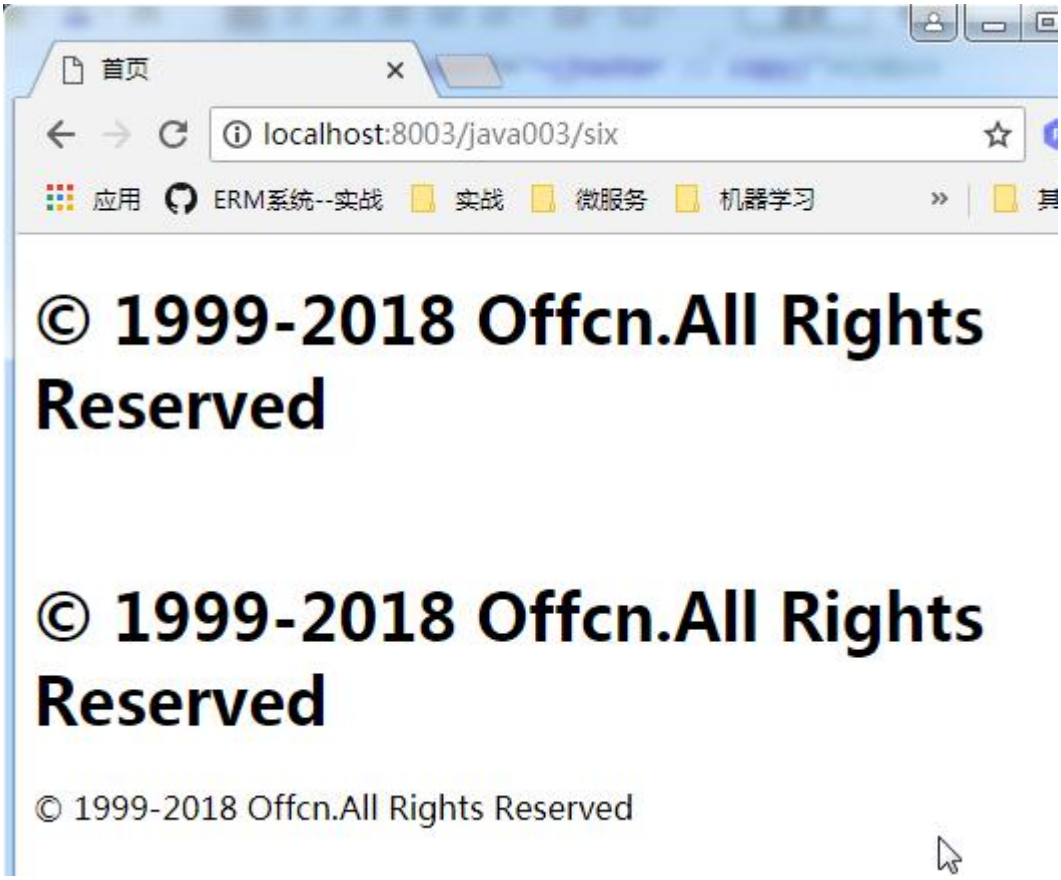
fragment 的引用

th:insert:保留自己的主标签，保留 th:fragment 的主标签。

th:replace:不要自己的主标签，保留 th:fragment 的主标签。

th:include:保留自己的主标签，不要 th:fragment 的主标签。（官方 3.0 后不推荐）

（4）、访问地址 http://localhost:8080/six 运行



10、Spring boot 集成 Thymeleaf 表达式内置对象使用

（1）、常见内置工具对象如下：

#dates 与 java.util.Date 对象的方法对应，格式化、日期组件抽取等等

#numbers 格式化数字对象的工具方法

#strings 与 java.lang.String 对应的工具方法

（2）、编写 Controller

```
package com.offcn.demo.controller;
```

```
import java.util.Date;


import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;


@Controller
public class SevenThymeleafController {
    /**
     * 访问 localhost:8080/java003 页面
     * 将数据 message 填充到 templates/index7.html
  * @param model
  * @return
  */
  @GetMapping("/seven")
  public String indexPage(Model model) {
    //日期时间
      Date date = new Date();
      model.addAttribute("date", date);
      //小数的金额
      double price=128.5678D;
```

```
        model.addAttribute("price", price);


        //定义大文本数据
        String str="Thymeleaf 是 Web 和独立环境的现代服务器端 Java 模板引擎，能够处理 HTML，XML，JavaScript，CSS 甚至纯文本。\r\n" +
                "Thymeleaf 的主要目标是提供一种优雅和高度可维护的创建模板的方式。为了实现这一点，它建立在自然模板的概念上，将其逻辑注入到模板文件中，不会影响模板被用作设计原型。这改善了设计的沟通，弥补了设计和开发团队之间的差距。\r\n" +
                "Thymeleaf 也从一开始就设计了 Web 标准 - 特别是 HTML5 - 允许您创建完全验证的模板，如果这是您需要的\r\n" ;


        model.addAttribute("strText", str);


    //定义字符串
    String str2="JAVA-offcn";
    model.addAttribute("str2", str2);


        return "index7";
  }
}
```

（3）、resource/templates 下,新增模板文件 index7.html

```
<!doctype html>
<html xmlns="http://www.w3.org/1999/xhtml"
```

```
      xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>首页</title>


</head>
<body>
时间：<span th:text="${#dates.format(date,'yyyy-MM-dd
HH:mm:ss')}">4564546</span></br>
金额：<span th:text="'￥'+${#numbers.formatDecimal(price, 1, 2)}">180</span>
</br>
<!-- # 这里的含义是 如果 atc.text 这个变量多余 200 个字符，后面显示... -->
<p th:text="${#strings.abbreviate(strText,60)}">内容内容内容</p>


<!-- 判断字符串是否为空 -->
<span th:if="${!#strings.isEmpty(str2)}">字符串 str2 不为空</span></br>
<!-- 截取字符串，指定长度 -->
<span th:text="${#strings.substring(str2,0,4)}">字符串 str2 的值</span>
</body>
</html>
```

（4）、运行访问地址 http://localhost:8080/java003/seven 效果如下

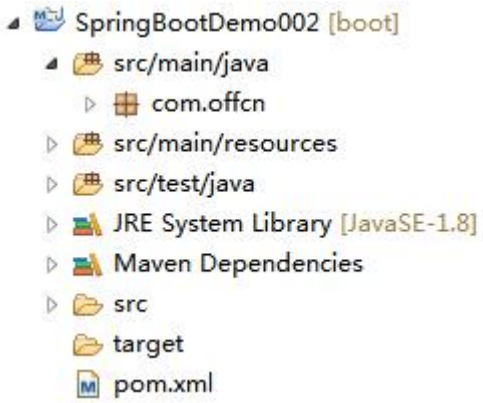
三、SpringBoot 使用 JPA、Thymeleaf 集成开发应用

使用 SpringBoot 集成 Jpa 、Thymeleaf 实现一个 web 版的用户增删改查基本功能。

1、创建一个新的 SpringBoot 工程

创建 SpringBoot 项目，模块选择：Web、Thymeleaf 、JPA 、MySQL

项目名称：SpringBootDemo002



pom.xml 核心依赖库如下：

```
<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-thymeleaf</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
    </dependency>
    <dependency>
        <groupId>com.alibaba</groupId>
        <artifactId>druid</artifactId>
        <version>1.1.10</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>

    <dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
        <version>4.2.1</version>
    </dependency>
</dependencies>
```

2、创建 SpringBoot 整合 JPA 处理数据库配置文件

```
spring:
  datasource:
    url: jdbc:mysql://localhost:3306/springboot-0311-02?serverTimezone=GMT%2B8
    username: root
    password: 123
    type: com.alibaba.druid.pool.DruidDataSource
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
  thymeleaf:
    cache: false
```

3、创建实体类 User.java

```
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "tb_user")
public class User {
    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "name",nullable = false,length = 100)
    private String name;

    @Column(name = "age",nullable = true,length = 4)
    private Integer age;
}
```

4、创建数据访问层接口定义 UserRepository.java

```
public interface UserDao extends JpaRepository<User,Long> {
```

```
    public User findByNameIs(String name);

    public User findByNameAndAge(String name,Integer age);
}
```

5、创建 Service 接口定义 UserService.java

```
public interface UserService {

    //新增用户
    public void add(User user);

    //修改
    public void update(Long id,User user);

    //删除
    public void delete(Long id);

    //根据 id 获取用户信息
    public User findOne(Long id);

    //获取全部用户信息
    public List<User> findAll();
}
```

6、创建 Service 实现类 UserServiceImpl.java

```
@Service
public class UserServiceImpl implements UserService {

    @Autowired
    private UserDao userDao;

    @Override
    public void add(User user) {
      userDao.save(user);
    }

    @Override
```

```
    public void update(Long id, User user) {
        user.setId(id);
     userDao.saveAndFlush(user);
    }

    @Override
    public void delete(Long id) {
     userDao.deleteById(id);
    }

    @Override
    public User findOne(Long id) {
        return userDao.findById(id).get();
    }

    @Override
    public List<User> findAll() {
        return userDao.findAll();
    }
}
```

7、创建 Controller　UserController.java

```
@Controller
public class UserController {

    @Autowired
    private UserService userService;

    @GetMapping("/")
    public String findAll(Model model){
        List<User> userList = userService.findAll();
        model.addAttribute("page",userList);

        return "user/list";
    }

    //用户点击 add 按钮，跳转到新增用户页面
    @RequestMapping("/toAdd")
    public String toAdd(){
        return "user/userAdd";
```

```
    }

    //保存新增用户数据
    @PostMapping("/add")
    public String save(User user){
        userService.add(user);
        //跳转到列表页
        return "redirect:/";
    }

    //跳转到编辑页面
    @RequestMapping("/toEdit/{id}")
    public String toEdit(Model model,@PathVariable("id") Long id){
        //读取对应 id 用户信息
        User user = userService.findOne(id);
        model.addAttribute("user",user);
        //跳转到编辑页面
        return "user/userEdit";
    }

    //保存修改用户数据
    @RequestMapping("/update")
    public String update(User user){
        userService.update(user.getId(),user);
        //跳转到列表页
        return "redirect:/";
    }

    //删除用户数据
    @RequestMapping("/delete/{id}")
    public String delete(@PathVariable("id") Long id){
        userService.delete(id);
        //跳转到列表页
        return "redirect:/";

    }
}
```

8、创建前端页面

修改 pom.xml 引入 bootstrap webJars

```
<dependency>
        <groupId>org.webjars</groupId>
        <artifactId>bootstrap</artifactId>
         <version>4.2.1</version>
</dependency>
```

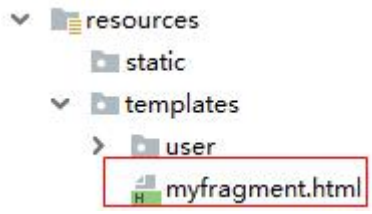
模板文件引用 bootstrap

```
<link rel='stylesheet' th:href="@{/webjars/bootstrap/4.2.1/css/bootstrap.min.css}"
href='webjars/bootstrap/4.2.1/css/bootstrap.min.css'>
```

在 templates 目录下建 user 目录，下面建 list.html,userAdd.html ,userEdit.html ,与 controller 中定义的对应

7.1 myfragment.html:

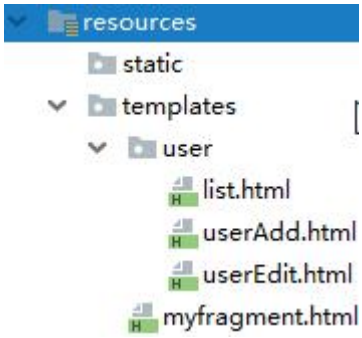
存放目录




```
<h1 th:fragment="head1" align="center">用户管理系统</h1>

<h1 th:fragment="foot1">
     &copy; 1999-2020 Offcn.All Rights Reserved
</h1>
```

7.2 list.html:

存放在目录




```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>用户管理</title>
    <link rel='stylesheet'
th:href="@{/webjars/bootstrap/4.2.1/css/bootstrap.min.css}"
href='webjars/bootstrap/4.2.1/css/bootstrap.min.css'>
</head>
<body class="container">
<div th:insert="~{myfragment :: head1}"></div>
<br/>
<h1>用户列表</h1>
<br/><br/>
<div class="with:80%">
    <div class="form-group">
        <div class="col-sm-2 control-label">
            <a href="toAdd" th:href="@{/toAdd}" class="btn btn-info">add</a>
        </div>
    </div>
    <table class="table table-hover">
        <thead>
        <tr>
            <th>#</th>
            <th>Name</th>
            <th>Age</th>
            <th>Edit</th>
            <th>Delete</th>
        </tr>
        </thead>
        <tbody>
```

```
        <tr  th:each="user : ${page}">
            <th scope="row" th:text="${user.id}">1</th>
            <td th:text="${user.name}">neo</td>
            <td th:text="${user.age}">6</td>
            <td><a th:href="@{'/toEdit/'+${user.id}}">edit</a></td>
            <td><a th:href="@{'/delete/'+${user.id}}">delete</a></td>
        </tr>
        </tbody>

    </table>

</div>
<div th:include="~{myfragment :: foot1}"></div>
</body>
</html>
```

7.3 userAdd.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>user</title>
    <link rel='stylesheet'
th:href="@{/webjars/bootstrap/4.2.1/css/bootstrap.min.css}"
href='webjars/bootstrap/4.2.1/css/bootstrap.min.css'>
</head>
<body class="container">
<div th:insert="~{myfragment :: head1}"></div>
<br/>
<h1>添加用户</h1>
<br/><br/>
<div class="with:80%">
    <form class="form-horizontal"    th:action="@{/add}"  method="post">
        <div class="form-group">
            <label for="name" class="col-sm-2 control-label">name</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" name="name"  id="name"
```

```
placeholder="name"/>
            </div>
        </div>

        <div class="form-group">
            <label for="age" class="col-sm-2 control-label">age</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" name="age"  id="age"
placeholder="age"/>
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-10">
                <input type="submit" value="Submit" class="btn btn-info" />
                     
                <input type="reset" value="Reset" class="btn btn-info" />
            </div>

        </div>
    </form>
</div>
<div th:include="~{myfragment :: foot1}"></div>
</body>
</html>
```

7.4 userEdit.html

```
<!DOCTYPE html>
<html lang="en" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8"/>
    <title>user</title>
    <link rel='stylesheet'
th:href="@{/webjars/bootstrap/4.2.1/css/bootstrap.min.css}"
href='webjars/bootstrap/4.2.1/css/bootstrap.min.css'>
</head>
<body class="container">
<div th:insert="~{myfragment :: head1}"></div>
<br/>
<h1>修改用户</h1>
```

```
<br/><br/>
<div class="with:80%">
    <form class="form-horizontal"   th:action="@{/update}" th:object="${user}"
method="post">
        <input type="hidden" name="id" th:value="*{id}" />
        <div class="form-group">
            <label for="name" class="col-sm-2 control-label">name</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" name="name"  id="name"
th:value="*{name}" placeholder="name"/>
            </div>
        </div>

        <div class="form-group">
            <label for="age" class="col-sm-2 control-label">age</label>
            <div class="col-sm-10">
                <input type="text" class="form-control" name="age"  id="age"
th:value="*{age}" placeholder="age"/>
            </div>
        </div>
        <div class="form-group">
            <div class="col-sm-offset-2 col-sm-10">
                <input type="submit" value="Submit" class="btn btn-info" />
                     
                <a href="/manageruser/" th:href="@{/manageruser/}" class="btn
btn-info">Back</a>
            </div>

        </div>
    </form>
</div>
<div th:include="~{myfragment :: foot1}"></div>
</body>
</html>
```

9、运行测试


优就业
用户列表
add
Name Age Edit Delete
10 中公教育 200 edit delete
8 优就业 100 edit delete
© 1999-2018 Offcn.All Rights Reserved


优就业
添加用户
name
name
age
age
Submit
Reset
© 1999-2018 Offcn.All Rights Reserved


优就业
修改用户
name
中公教育
age
200
Submit
Back
© 1999-2018 Offcn.All Rights Reserved