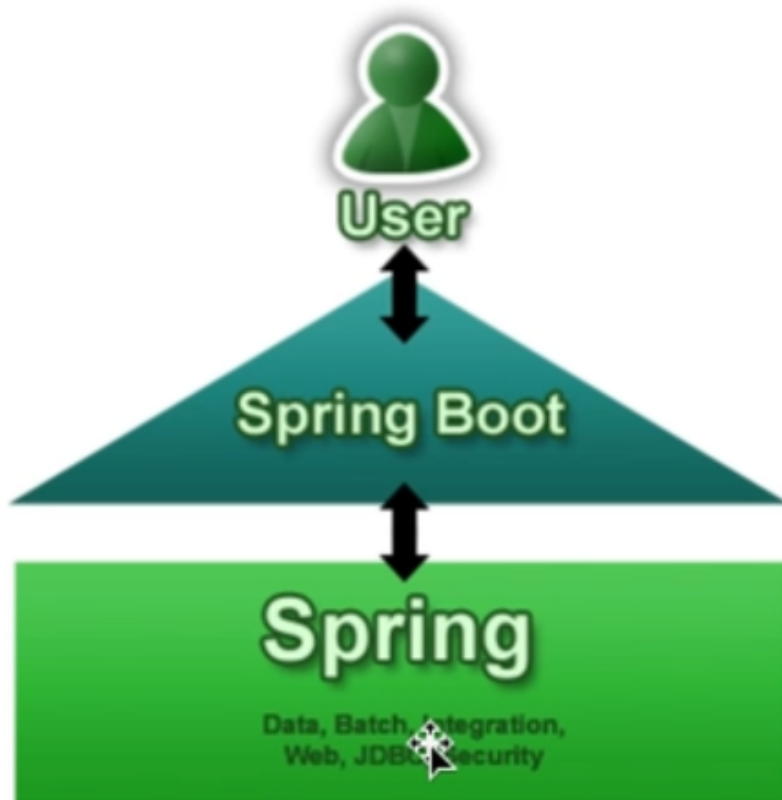


Spring Boot 学习记录和总结

Spring Boot 是一个快速开发框架，它可以迅速搭建出一套基于 Spring 框架体系的应用，是 Spring Cloud 的基础。

Spring Boot 开启了各种自动装配，从而简化了代码的开发，而不需要编写各种配置文件，只需要引入相关的依赖就可以迅速搭建出一个应用。



特点

- 不需要 web.xml
- 不需要 springmvc.xml
- 不需要 tomcat (Spring Boot 内置了 tomcat)
- 不需要配置 JSON 解析
- 支持 REST 架构
- 个性化配置非常简单

一、如何使用 Spring Boot

- 创建 Maven 工程，pom.xml 导入相关依赖

```
<!-- 继承父包-->
<parent>
    <artifactId>spring-boot-starter-
parent</artifactId>
    <groupId>org.springframework.boot</groupId>
    <version>2.3.3.RELEASE</version>
</parent>
<dependencies>
    <!-- web 启动的jar包 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
web</artifactId>
    </dependency>

    <!--其他的引入-->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
</dependencies>
```

- 创建 student 实体类

```
package com.gloryh.entity;

import lombok.AllArgsConstructor;
import lombok.Data;

/**
 * 学生实体类
 *
 * @author 黄光辉
```

```

    * @since 2020/9/24
    **/
@Data
@AllArgsConstructor
public class Student {
    private long id;
    private String name;
    private int age;
}

```

- 创建对应 StudentRepository 接口

```

package com.gloryh.repository;

import com.gloryh.entity.Student;

import java.util.Collection;

/**
 * 学生实体类实现方法接口
 *
 * @author 黄光辉
 * @since 2020/9/24
 **/
public interface StudentRepository {
    /**
     * 查询所有学生信息
     *
     * @return 学生信息列表
     */
    public Collection<Student> findAll();

    /**
     * 按 id 查询学生信息
     *
     * @param id 学生id
     * @return 学生实体类
     */
    public Student findById(long id);

    /**

```

```

        * 按 id 删除学生信息
        *
        * @param id 学生id
        */
    public void deleteById(long id);

    /**
     * 更新或添加学生信息
     *
     * @param student 要更新或添加的学生信息
     */
    public void saveOrUpdate(Student student);
}

```

- 实现 StudentRepository 的接口

```

package com.gloryh.repository.impl;

import com.gloryh.entity.Student;
import com.gloryh.repository.StudentRepository;
import org.springframework.stereotype.Repository;

import java.util.Collection;
import java.util.HashMap;
import java.util.Map;

/**
 * 学生实体类实现方法
 *
 * @author 黄光辉
 * @since 2020/9/24
 */
@Repository
public class StudentRepositoryImpl implements
StudentRepository {
    /**
     * Map 代替数据库
     */
    private static Map<Long, Student> studentMap;
    static {
        studentMap =new HashMap<>();
    }
}

```

```
        studentMap.put(1L,new Student(1L,"张三",20));
        studentMap.put(2L,new Student(2L,"李四",21));
        studentMap.put(3L,new Student(3L,"王五",22));
    }
    /**
     * 查询所有学生信息
     *
     * @return 学生信息列表
     */
    @Override
    public Collection<Student> findAll() {
        return studentMap.values();
    }

    /**
     * 按 id 查询学生信息
     *
     * @param id 学生id
     * @return 学生实体类
     */
    @Override
    public Student findById(long id) {
        return studentMap.get(id);
    }

    /**
     * 按 id 删除学生信息
     *
     * @param id 学生id
     */
    @Override
    public void deleteById(long id) {
        studentMap.remove(id);
    }

    /**
     * 更新或添加学生信息
     *
```

```

        * @param student 要更新或添加的学生信息
        */
    @Override
    public void saveOrUpdate(Student student) {
        studentMap.put(student.getId(), student);
    }
}

```

- 前后端交互方法类

```

package com.gloryh.controller;

import com.gloryh.entity.Student;
import com.gloryh.repository.StudentRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Collection;

/**
 * 学生实体类接口方法调用,前后端交互
 *
 * @author 黄光辉
 * @since 2020/9/24
 */
@RestController
@RequestMapping("/student")
public class StudentHandler {
    @Autowired
    private StudentRepository studentRepository;

    @GetMapping("/findAll")
    public Collection<Student> findAll() {
        return studentRepository.findAll();
    }

    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id")
    long id) {

```

```

        return studentRepository.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student)
    {
        studentRepository.saveOrUpdate(student);
    }

    @PutMapping("/update")
    public void update(@RequestBody Student
student) {
        studentRepository.saveOrUpdate(student);
    }

    @DeleteMapping("/delete/{id}")
    public void deleteById(@PathVariable("id") long
id) {
        studentRepository.deleteById(id);
    }
}

```

注：我们在 SpringBoot 中进行 `@Autowired` 自动注入，就不需要再在配置文件中配置来匹配对应的地方类来完成注入了，SpringBoot 会自动帮我们完成这个任务。我们只需要在对应的类使用 `@Repository` 注解即可。

- 创建 Application 类 启动 Spring Boot 内置的 tomcat

```

package com.gloryh;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * 启动内置tomcat,这个方法也是整个项目的入口
 *
 * @author 黄光辉
 * @since 2020/9/24

```

```

    /**/
    @SpringBootApplication
    public class Application {
        public static void main(String[] args) {

            SpringApplication.run(Application.class, args);
        }
    }

```

- 运行 Application 的 main 方法，观察控制台信息

```

main] com.gloryh.Application : Starting Application on HuangGunghui with PID 19356 (E:\Prog
main] com.gloryh.Application : No active profile set, falling back to default profiles: def
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.37]
main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 891
main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
main] com.gloryh.Application : Started Application in 1.724 seconds (JVM running for 3.509)

```

控制台会显示 已经进行的一些服务的信息，包括 Tomcat 的启动，项目的启动时间等等。。。

- 启动后，使用浏览器访问 Tomcat，测试查询所有 Student 的信息：

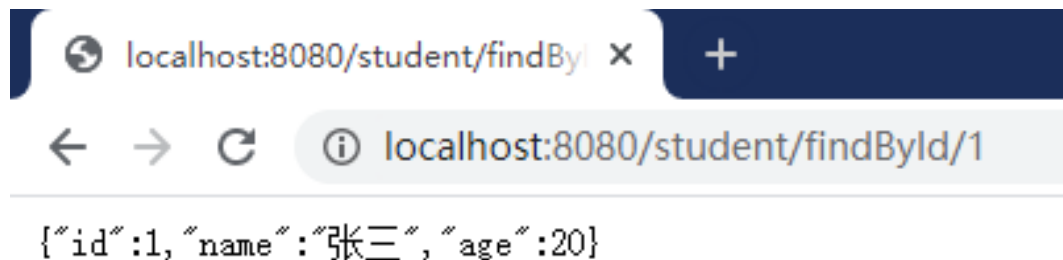


```

[{"id":1,"name":"张三","age":20}, {"id":2,"name":"李四","age":21}, {"id":3,"name":"王五","age":22}]

```

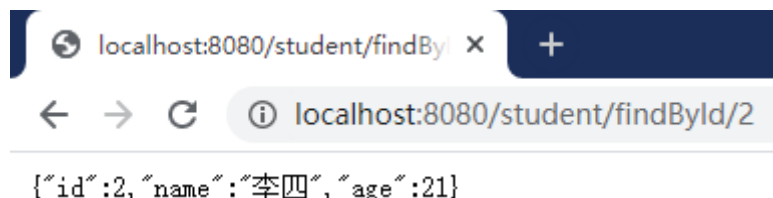
- 测试按 id 查询学生信息



```

{"id":1,"name":"张三","age":20}

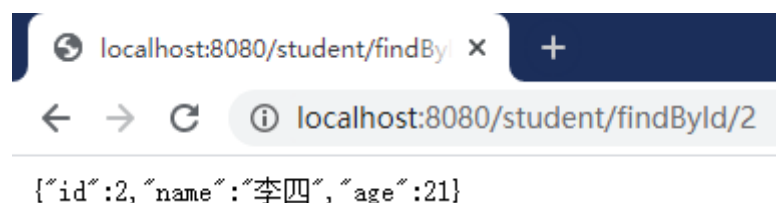
```



```

{"id":2,"name":"李四","age":21}

```

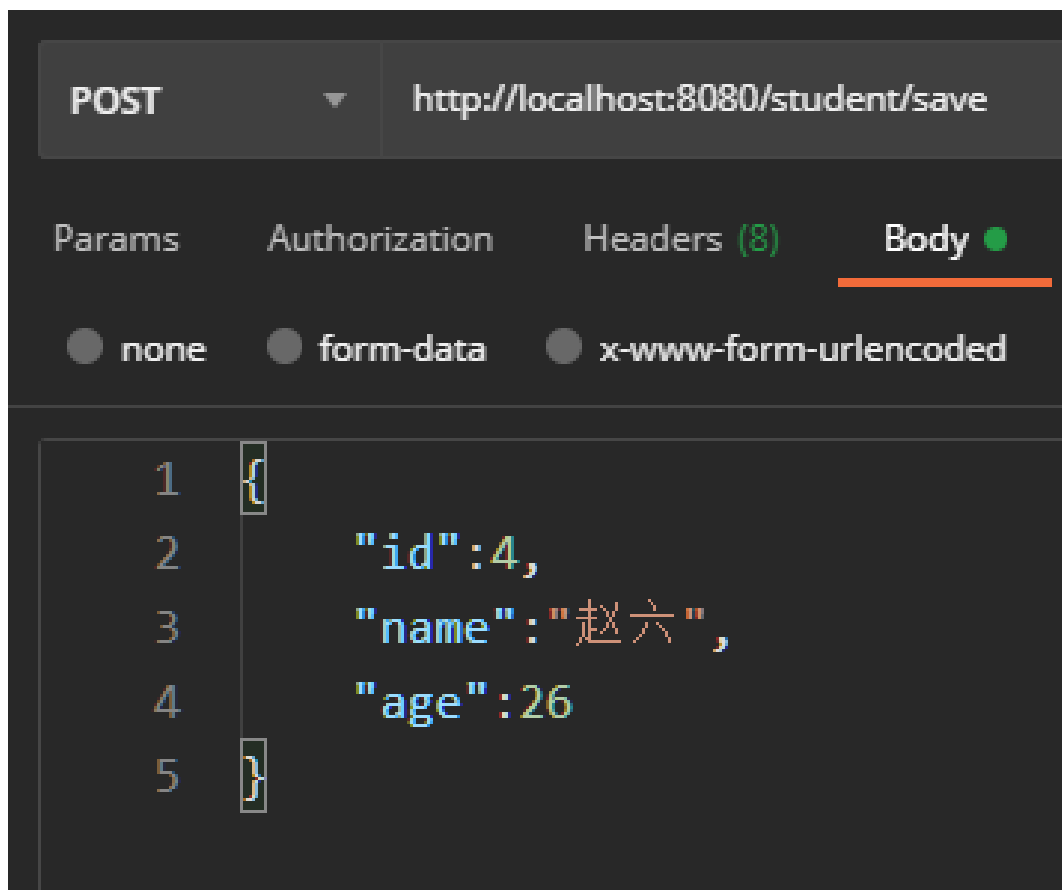


```

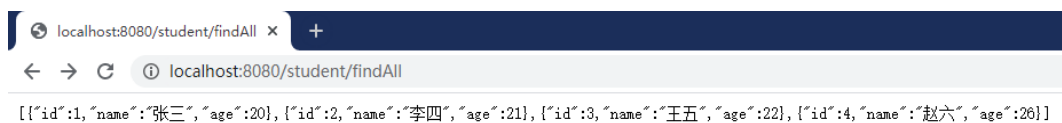
{"id":2,"name":"李四","age":21}

```

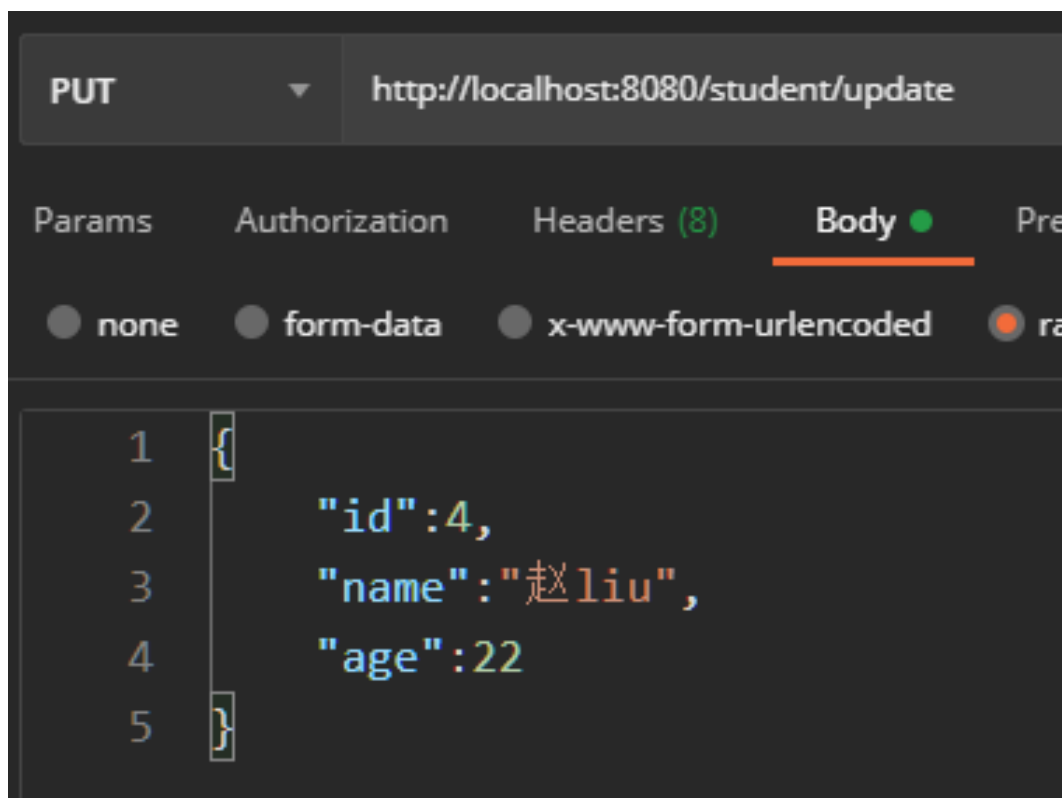
- 使用 PostMan 发送 JSON 数据，测试添加学生信息。



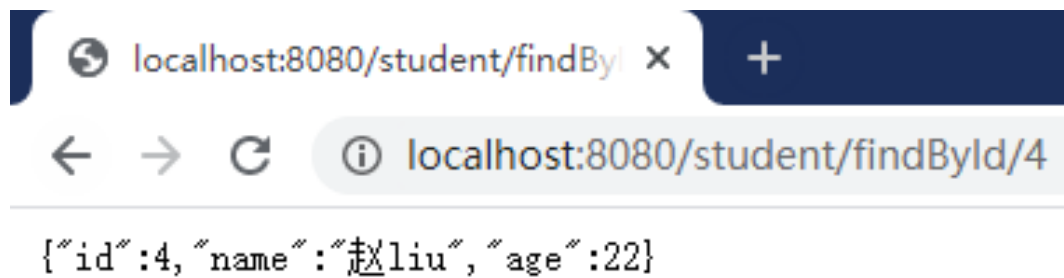
运行后查询所有学生信息：



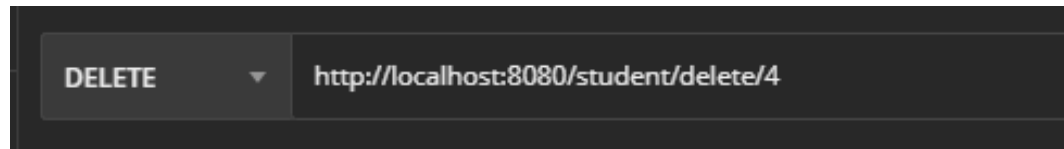
- 使用 PostMan 发送 JSON 数据，测试修改学生信息。



运行后查询该 学生信息：



- 测试按 id 删除 学生信息



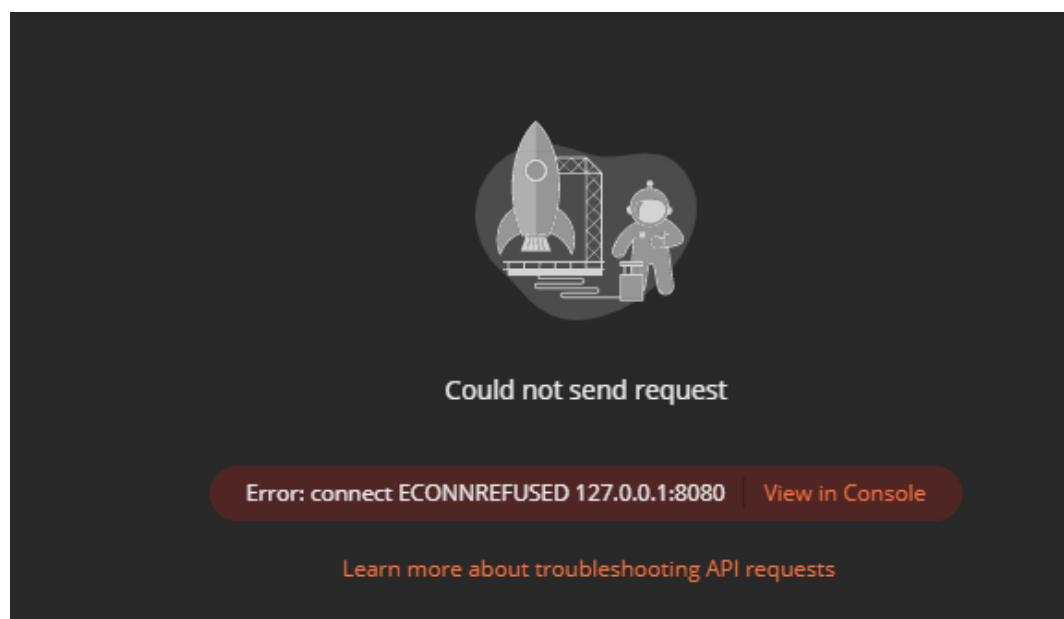
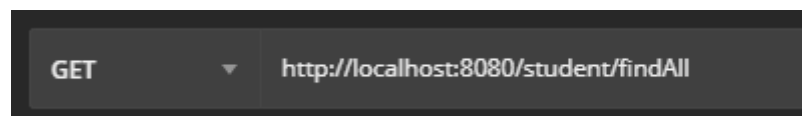
运行后查询所有学生信息;



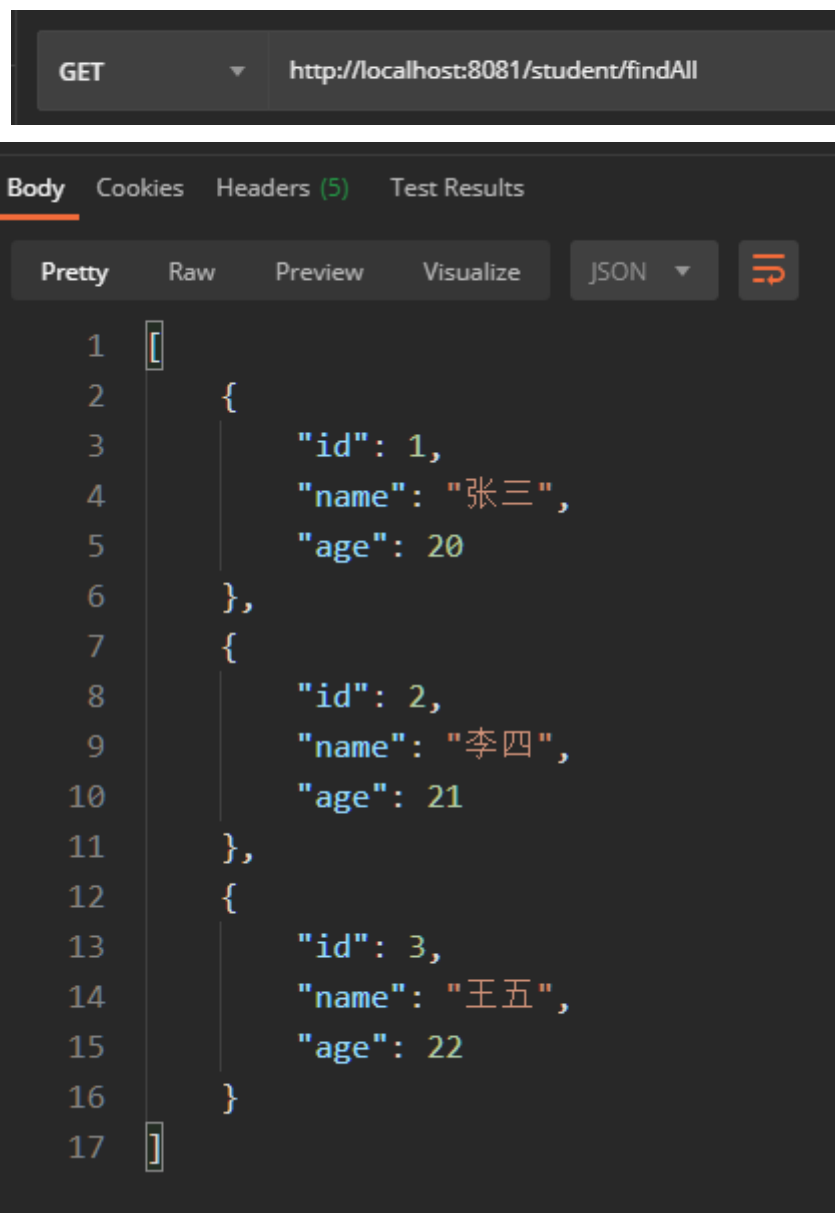
- 如果我们需要修改例如 tomcat 的端口，就需要在 resources 文件夹中 新建一个 application.yml 文件用于修改 Spring Boot 的配置信息（例如：修改为8081）。

```
server:  
  port: 8081
```

重启项目，使用8080端口访问：



报错，使用8081端口访问：



成功。

二、Spring Boot & JSP 整合

- pom.xml 中导入相关依赖

```
<!-- 继承父包-->
<parent>
  <artifactId>spring-boot-starter-
parent</artifactId>
  <groupId>org.springframework.boot</groupId>
  <version>2.3.3.RELEASE</version>
</parent>
<dependencies>
  <!-- web 启动的jar包 -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
```

```

        <artifactId>spring-boot-starter-
web</artifactId>
    </dependency>

    <!-- JSP 整合 -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
tomcat</artifactId>
    </dependency>
    <dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
    </dependency>

    <!-- JSTL -->
    <dependency>
        <groupId>jstl</groupId>
        <artifactId>jstl</artifactId>
        <version>1.2</version>
    </dependency>

    <!-- 其他的引入 -->
    <dependency>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.11</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
</dependencies>

```

- 创建 application.yml 配置文件（配置端口和 JSP 文件转化格式）

```
server:
  port: 8080
spring:
  mvc:
    view:
      prefix: /
      suffix: .jsp
```

- 创建 Handler

```
package com.gloryh.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestMapping;

/**
 * 测试框架是否搭建成功
 *
 * @author 黄光辉
 * @since 2020/9/26
 */
@Controller
@RequestMapping("/hello")
public class HelloHandler {

    @GetMapping("/index")
    public String index() {
        System.out.println("index...");
        return "index";
    }
}
```

- 创建 Application类作为Spring Boot 项目的启动入口

```
package com.gloryh;

import org.springframework.boot.SpringApplication;
```

```
import
org.springframework.boot.autoconfigure.SpringBootApplication;

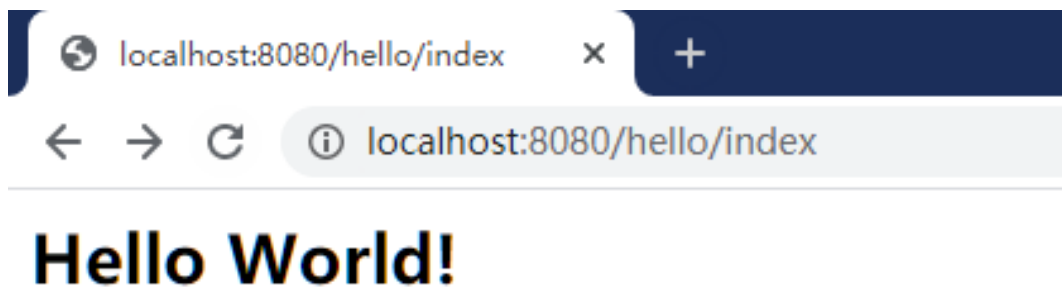
/**
 * 启动内置tomcat,这个方法也是整个项目的入口
 *
 * @author 黄光辉
 * @since 2020/9/24
 */
@SpringBootApplication
public class Application {
    public static void main(String[] args) {

        SpringApplication.run(Application.class,args);
    }
}
```

- 被调用的 index.jsp 页面源码

```
<html>
<body>
<h2>Hello world!</h2>
</body>
</html>
```

- 测试运行:



控制台观察是否进入方法:

```
2020-09-26 14:08:44.020 IN
2020-09-26 14:09:11.464 IN
2020-09-26 14:09:11.464 IN
2020-09-26 14:09:11.471 IN
index...
```

- 调用之前的 StudentRepository , 在JSP 页面展示对应的信息

```
package com.gloryh.controller;

import com.gloryh.repository.StudentRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.servlet.ModelAndView;

/**
 * 测试框架是否搭建成功
 *
 * @author 黄光辉
 * @since 2020/9/26
 */
@Controller
@RequestMapping("/hello")
public class HelloHandler {
    @Autowired
    private StudentRepository studentRepository;

    @GetMapping("/findAll")
    public ModelAndView index() {
        System.out.println("index...");
    }
}
```

```

        ModelAndView modelAndView = new
ModelAndView();
        modelAndView.setViewName("index");
        modelAndView.addObject("list",
studentRepository.findAll());
        return modelAndView;
    }

}

```

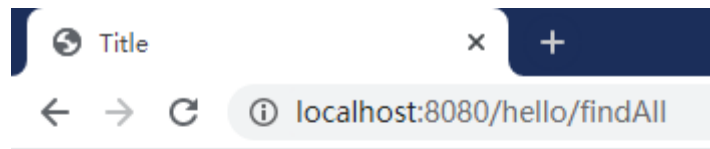
- JSP 源码

```

<%@ page contentType="text/html; charset=UTF-8"
language="java" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c"
uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<h1>学生信息</h1>
<table>
    <tr>
        <th>编号</th>
        <th>姓名</th>
        <th>年龄</th>
    </tr>
    <c:forEach items="${list}" var="studentList">
        <tr>
            <th>${studentList.id}</th>
            <th>${studentList.name}</th>
            <th>${studentList.age}</th>
        </tr>
    </c:forEach>
</table>
</body>
</html>

```

- 运行结果



学生信息

编号 姓名 年龄

1 张三 20

2 李四 21

3 王五 22

- 加入编辑和删除功能

删除功能:

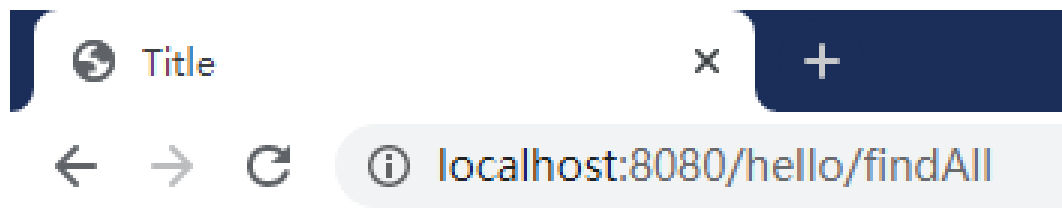
```
@GetMapping("/deleteById/{id}")
public String delete(@PathVariable("id") long id){
    studentRepository.deleteById(id);
    //删除后重定向到
    http://localhost:8080/hello/findAll 完成刷新
    return "redirect:/hello/findAll";
}
```

JSP代码:

```
<%--
    Created by IntelliJ IDEA.
    User: admin
    Date: 2020/9/26
    Time: 14:13
    To change this template use File | Settings |
    File Templates.
--%>
<%@ page contentType="text/html; charset=UTF-8"
    language="java" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
    <title>Title</title>
```

[illegible]

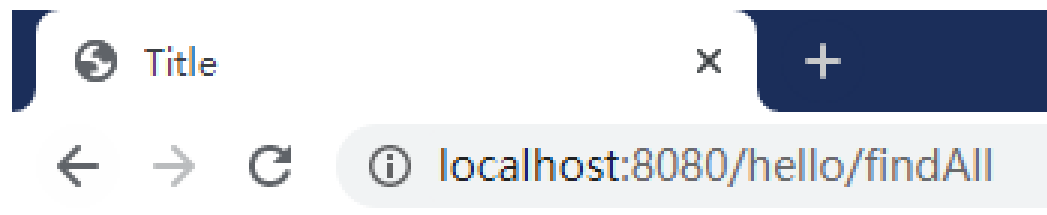
测试删除:



学生信息

编号	姓名	年龄	功 能	
1	张三	20	编辑	删除
2	李四	21	编辑	删除
3	王五	22	编辑	删除

删除王五：



学生信息

编号	姓名	年龄	功 能	
1	张三	20	编辑	删除
2	李四	21	编辑	删除

编辑功能（先按id查找信息，显示在编辑页面，然后完成修改后重定向到列表）：

```

@GetMapping("/findById/{id}")
public ModelAndView findById(@PathVariable("id")
    Long id){
    ModelAndView modelAndView =new ModelAndView();
    modelAndView.setViewName("edit");

    modelAndView.addObject("student",studentRepository
        .findById(id));
    return modelAndView;
}
@PostMapping("/update")
public String update(Student student){
    studentRepository.saveOrUpdate(student);
    //修改后重定向到
    http://localhost:8080/hello/findAll 完成刷新
    return "redirect:/hello/findAll";
}

```

JSP (edit.jsp) 代码:

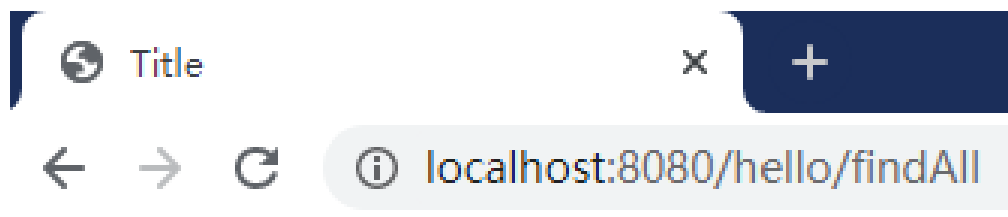
```

<!--
    Created by IntelliJ IDEA.
    User: admin
    Date: 2020/9/26
    Time: 14:45
    To change this template use File | Settings |
    File Templates.
-->
<%@ page contentType="text/html; charset=UTF-8"
    language="java" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<h1>学生信息</h1>
    <form action="/hello/update" method="post">

```

```
<input name="id" value="${student.id}"
readonly="readonly" type="hidden"/><br/>
    姓名: <input type="text" name="name"
value="${student.name}"/><br/>
    年龄: <input type="text" name="age"
value="${student.age}"/><br/>
    <input type="submit" value="确认修改"/>
</form>
</body>
</html>
```

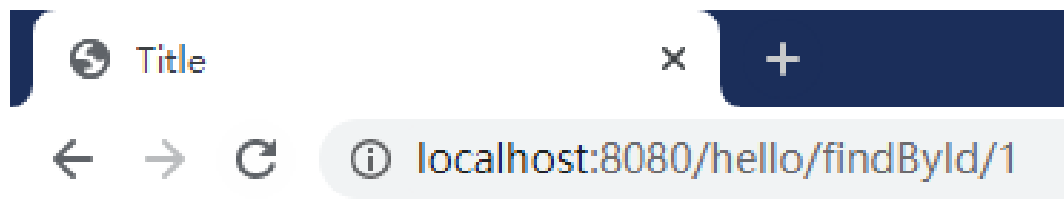
测试:



学生信息

编号	姓名	年龄	功	能
1	张三	20	编辑	删除
2	李四	21	编辑	删除
3	王五	22	编辑	删除

编辑张三的信息:

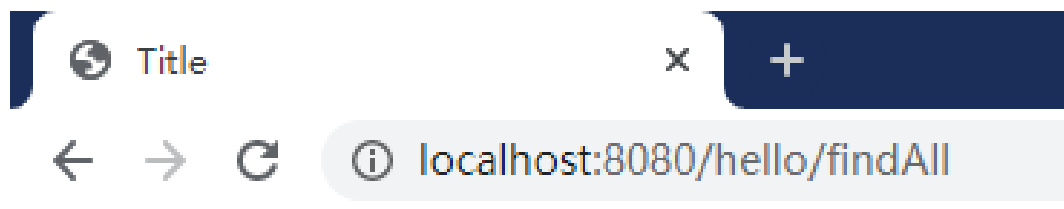


学生信息

姓名:

年龄:

姓名修改为张四，年龄修改为21，点击确认修改，返回列表查看结果：



学生信息

编号	姓名	年龄	功	能
1	张四	21	编辑	删除
2	李四	21	编辑	删除
3	王五	22	编辑	删除

- 添加学生

在 index.jsp 中加入添加学生页面跳转超链接：

```
<br/>
<a href="/add.jsp">添加学生</a>
```

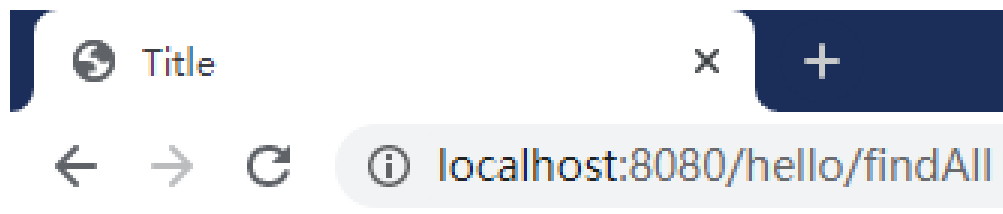
JSP(add.jsp)代码:

```
<%--
    Created by IntelliJ IDEA.
    User: admin
    Date: 2020/9/26
    Time: 14:45
    To change this template use File | Settings |
    File Templates.
--%>
<%@ page contentType="text/html; charset=UTF-8"
    language="java" %>
<%@ page isELIgnored="false" %>
<%@ taglib prefix="c"
    uri="http://java.sun.com/jstl/core_rt" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<h1>学生信息</h1>
<form action="/hello/save" method="post">
    编号: <input name="id" type="text"/><br/>
    姓名: <input type="text" name="name" /><br/>
    年龄: <input type="text" name="age" /><br/>
    <input type="submit" value="添加"/>
</form>
</body>
</html>
```

对应方法:

```
@PostMapping("/save")
public String save(Student student) {
    studentRepository.saveOrUpdate(student);
    //添加后重定向到
    http://localhost:8080/hello/findAll 完成刷新
    return "redirect:/hello/findAll";
}
```

测试:

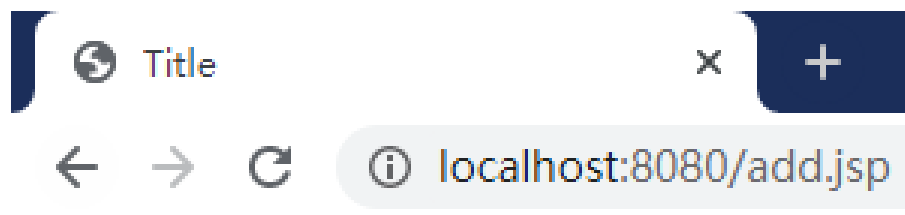


学生信息

编号	姓名	年龄	功 能	
1	张三	20	编辑	删除
2	李四	21	编辑	删除
3	王五	22	编辑	删除

[添加学生](#)

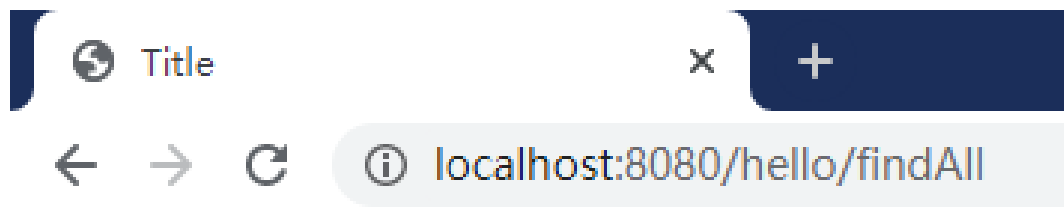
点击添加学生，跳转至添加页面：



学生信息

编号:	<input type="text"/>
姓名:	<input type="text"/>
年龄:	<input type="text"/>
<input type="button" value="添加"/>	

依次输入编号（4），姓名（赵六），年龄（29），点击添加查看结果：



学生信息

编号	姓名	年龄	功能	
1	张三	20	编辑	删除
2	李四	21	编辑	删除
3	王五	22	编辑	删除
4	赵六	29	编辑	删除

[添加学生](#)

三、Spring Boot 整合 HTML

1、结合 Thymeleaf 完成和 HTML 的整合

Spring Boot 可以结合 Thymeleaf 模板来整合 HTML，使用原生的 HTML 作为视图。

Thymeleaf

这个模板是面向Web 和 独立环境的 Java 模板引擎，能够处理绝大多数的静态资源，包括 HTML、XML、JavaScript、CSS 等等。

具体实现

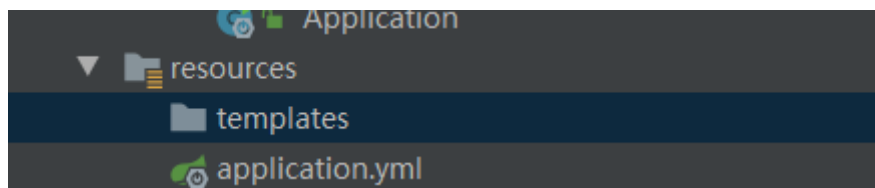
- pom.xml 中添加相关的依赖

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

- application.yml 中 配置视图解析器

```
server:
  port: 8080
spring:
  thymeleaf:
    prefix: classpath:/templates/
    suffix: .html
    mode: HTML5
    encoding: UTF-8
```

注：prefix 前缀（文件夹位置如图），suffix 后缀，mode 模板类型，encoding 编码格式



- 前后端交互逻辑方法

```
package com.gloryh.controller;

import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestMapping;

/**
 * 与 HTML 的整合 测试方法
 *
 * @author 黄光辉
 * @since 2020/9/27
```

```

    /**
    @Controller
    @RequestMapping("/index")
    public class IndexHandler {

        @GetMapping("/index")
        public String index(){
            System.out.println("index...");
            return "index";
        }

    }

```

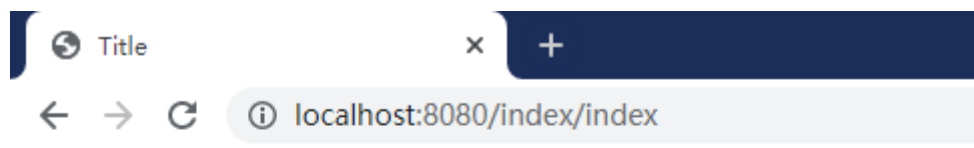
- 对应 HTML 文件

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Hello word!</h1>
<h2>进入该页面，说明与 HTML 整合成功</h2>
</body>
</html>

```

- 测试



Hello Word!

进入该页面，说明与 HTML 整合成功

- 控制台查看方法是否被调用

2020-09-27 09:54:
2020-09-27 09:54:
index...

- 解析 Student 集合

```
package com.gloryh.controller;

import com.gloryh.entity.Student;
import com.gloryh.repository.StudentRepository;
import
org.springframework.beans.factory.annotation.Autowired;
red;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.servlet.ModelAndView;

import java.util.List;

/**
 * 与 HTML 的整合 测试方法
 *
 * @author 黄光辉
 * @since 2020/9/27
 */
@Controller
@RequestMapping("/index")
public class IndexHandler {

    @Autowired
    private StudentRepository studentRepository;

    @GetMapping("/index")
```

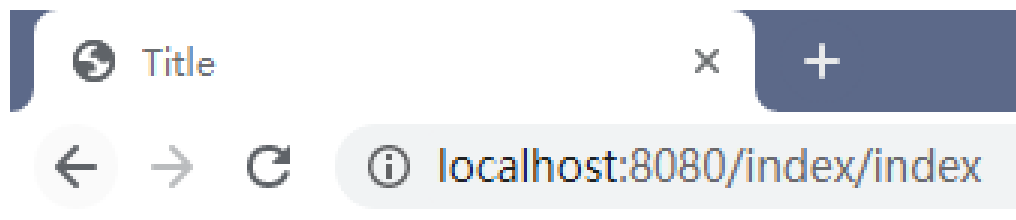
```
public ModelAndView index(){
    System.out.println("index...");
    ModelAndView modelAndView =new
ModelAndView();
    modelAndView.setViewName("index");

    modelAndView.addObject("list",studentRepository.findAll());
    return modelAndView;
}
```

- 前端代码

[illegible]

- 显示结果:



学生信息

编号 姓名 年龄 功 能

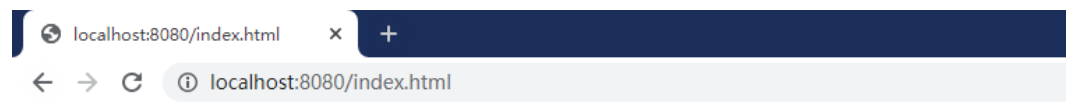
1 张三 20

2 李四 21

3 王五 22

注：resources 下的文件夹，除了被命名为 static 的文件夹外，Spring Boot 一律不会对其进行扫描。例如：

直接访问：index.html



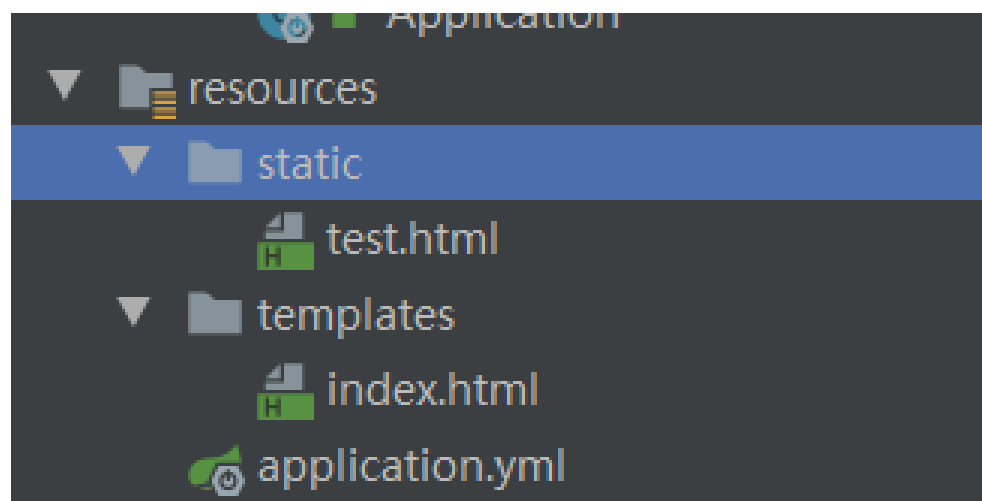
Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

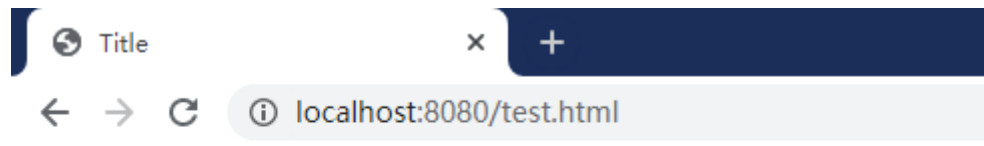
Sun Sep 27 10:19:31 AWST 2020

There was an unexpected error (type=Not Found, status=404).

创建 static 文件夹，编写测试网页



直接访问 test.html:



这是 static 写的 test.html

2、Thymeleaf的常用语法

- 赋值、拼接

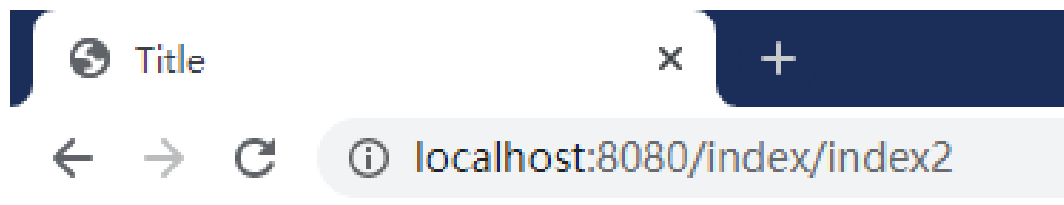
前端:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    赋值: <p th:text="${name}"></p>
    拼接1: <p th:text="'学生姓名是'+${name}+'。没错，就是'+${name}'"></p>
    拼接2: <p th:text="|学生姓名是${name}|" >
</body>
</html>
```

后端:

```
@GetMapping("/index2")
public String index2(Map<String, String> map) {
    map.put("name", "张三");
    return "index2";
}
```

运行结果:



赋值:

张三

拼接1:

学生姓名是张三。没错，就是张三

拼接2:

学生姓名是张三

- 条件判断: if / unless

th:if 表示条件成立时显示内容, 而 th:unless 表示条件不成立时显示内容

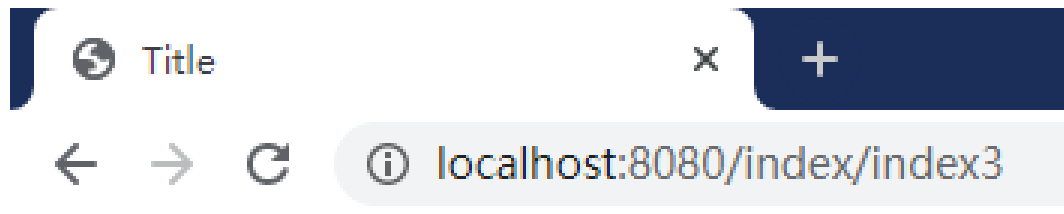
前端:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
</head>
<body>
  <p th:if="${flag == true}" th:text="if判断成立">
</p>
  <p th:unless="${flag != true}" th:text="unless判断成立"></p>
</body>
</html>
```

后端:


```
@GetMapping("/index3")
public String index3(Map<String, Boolean> map) {
    map.put("flag", true);
    return "index2";
}
```

显示:



if判断成立

unless判断成立

- 循环 th:each

之前测试 整合 的时候用过了，可以回头看一下

注意:

该循环方法提供一个状态变量 stat

index 值 代表 集合中元素的 index 从 0 开始

count 值 代表 集合中元素的 count 从1开始

size 值 代表 集合的大小

current 值 代表 当前迭代的变量

even 值 代表 当前迭代是否为奇数

odd 值 代表 当前迭代是否为偶数

first 值 代表 当前迭代的元素是否为第一个

last 值 代表 当前迭代的元素是否为最后一个

除此之外，stat 还提供多个参数值，可以自己去试试

前端:

```
<tr>
  <th>index</th>
```


当你的URL 需要动态地形成时，需要使用 {name} (name=\${name}) 这种格式，例如<http://localhost:8080/index/>。最后追加一个动态name。

```
<a th:href="@{http://localhost:8080/index/{name}
(name=${name})}"></a>
```

- 链接 src

用法（图片链接：/imgs/ddd.png,或后台传过来的src，命名为src）：

```


```

- 三元运算

```
<input th:value="${age gt 30?'中年':'青年'}">
```

解释，当传过来的值为 大于 30 时，input 的 value 值为中年，否则为青年。

后台传一个 age，值为30：

```
@GetMapping("/eq")
public String eq(Model model){
    model.addAttribute("age",30);
    return "test";
}
```

显示：

学生信息

index count 编号 姓名 年龄 功 能

青年

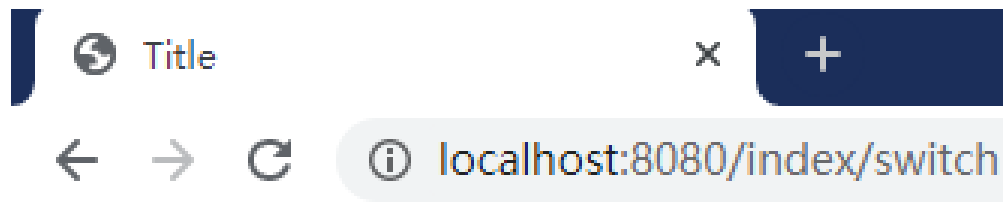
- 条件判断 switch

```
<div th:switch="${gender}">
    <p th:case="女">女</p>
    <p th:case="男">男</p>
    <p th:case="*">未知</p>
</div>
```

后台传过来一个 gender，值为 女：

```
@GetMapping("/switch")
public String switchTest(Model model) {
    model.addAttribute("gender", "女");
    return "index";
}
```

结果：



女

- 基本对象

`#ctx` 上下文对象

`#vars` 上下文变量

`#locale` 区域对象

`#request` `HttpRequest` 对象

`#response` `HttpResponse` 对象

`#session` `HttpSession` 对象

`#servletContext` `ServletContext` 对象

- 内嵌对象

内嵌对象可以直接通过 `#` 访问

`dates` : `java.util.Date` 的功能方法

`calendars` : `java.util.Calendar` 的功能方法

numbers : 格式化数字

strings : java.lang.String 的功能方法

objects : Object 的功能方法

booleans : 对布尔求值的方法

arrays : 操作数组的功能方法

lists : 操作集合的功能方法

sets : 操作集合的功能方法

maps : 操作集合的功能方法

四、Spring Boot 的数据校验

- 添加 依赖

```
<dependency>
  <groupId>javax.validation</groupId>
  <artifactId>validation-api</artifactId>
  <version>2.0.1.Final</version>
</dependency>
<dependency>
  <groupId>org.hibernate.validator</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>6.0.14.Final</version>
</dependency>
```

- 在实体类中直接只用注解进行数据校验约束和信息的返回

```
package com.gloryh.entity;

import lombok.Data;
import org.hibernate.validator.constraints.Length;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

/**
 * 数据校验 测试 实体类
 */
```

```

* @author 黄光辉
* @since 2020/9/29
**/
@Data
public class User {
    @NotNull(message = "id不能为空")
    private Long id;
    @NotEmpty(message = "姓名不能为空")
    @Length(min = 2,message = "姓名长度不得小于2")
    private String name;
    @Min(value =16,message = "年龄不得小于16")
    private int age;
}

```

- 业务逻辑层进行交互 (@Valid注解负责验证, BindingResult 负责收集错误信息)

```

@GetMapping("/validate")
public void validate(@Valid User user,
BindingResult bindingResult) {
    //打印 user 信息
    System.out.println(user);
    //判断 BindingResult 是否收集到错误信息
    if (bindingResult.hasErrors()) {
        //收集到, 打印错误代码+错误信息
        List<ObjectError> objectErrors =
bindingResult.getAllErrors();
        for (ObjectError objectError:objectErrors)
        {

            System.out.println(objectError.getCode()+":"+objectError.getDefaultMessage());
        }
    }
}

```

- 不传参数直接访问 (控制台信息)

```
2020-07-27 14:07:07:707 INFO 17112
User(id=null, name=null, age=0)
NotEmpty:姓名不能为空
NotNull:id不能为空
Min:年龄不得小于16
```

五、Spring Boot 整合 JDBC

- pom.xml 引入相关依赖

```
<!-- Spring Data JDBC -->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
jdbc</artifactId>
</dependency>

<!-- MySQL 驱动 -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.20</version>
</dependency>
```

- application.yml 进行相关配置

```
server:
  port: 8080
spring:
  mvc:
    view:
      prefix: /
      suffix: .jsp
  thymeleaf:
    prefix: classpath:/templates/
    suffix: .html
    mode: HTML5
    encoding: UTF-8
  datasource:
```

```
url: jdbc:mysql://localhost:3305/mybatis_demo?
useUnicode=true&characterEncoding=utf8&serverTimezone=UTC&allowMultiQueries=true
username: admin
password: 123
driver-class-name: com.mysql.cj.jdbc.Driver
```

- 用到的实体类

```
package com.gloryh.entity;

import lombok.Data;
import org.hibernate.validator.constraints.Length;

import javax.validation.constraints.Min;
import javax.validation.constraints.NotEmpty;
import javax.validation.constraints.NotNull;

/**
 * 数据校验 测试 实体类
 *
 * @author 黄光辉
 * @since 2020/9/29
 */
@Data
public class User {
    @NotNull(message = "id不能为空")
    private Long id;
    @NotEmpty(message = "姓名不能为空")
    @Length(min = 2,message = "姓名长度不得小于2")
    private String username;
    @Min(value = 16,message = "年龄不得小于16")
    private int age;
    @NotEmpty(message = "密码不得为空")
    private String password;
}
```

- 用到的数据库

对象		user @mybatis_demo (8.0) -...		
开始事务		文本	筛选	排序
id	username	password	age	
1	张三	111	21	
2	李四	222	22	
3	王五	333	23	
4	赵六	444	24	

- 新建一个 UserRepository 接口类

```
package com.gloryh.repository;

import com.gloryh.entity.User;

import java.util.List;

/**
 * 用户接口类
 *
 * @author 黄光辉
 * @since 2020/9/29
 */
public interface UserRepository {
    public List<User> findAll();
    public User findById(long id);
    public void save(User user);
    public void update(User user);
    public void deleteById(long id);
}
```

- 新建接口对应的实现类

```
package com.gloryh.repository.impl;

import com.gloryh.entity.User;
import com.gloryh.repository.UserRepository;
import
org.springframework.beans.factory.annotation.Autowired;
red;
```

```
import
org.springframework.jdbc.core.BeanPropertyRowMapper
;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import java.util.List;

/**
 * 用户接口实现类
 *
 * @author 黄光辉
 * @since 2020/9/29
 */
@Repository
public class UserRepositoryImpl implements
UserRepository {
    /* JdbcTemplate 注入即可实现连接数据库*/
    @Autowired
    private JdbcTemplate jdbcTemplate;
    @Override
    public List<User> findAll() {
        return jdbcTemplate.query("SELECT * FROM
user",new BeanPropertyRowMapper<>(User.class));
    }

    @Override
    public User findById(long id) {
        //return
jdbcTemplate.queryForObject("SELECT * FROM user
WHERE id ="+id,User.class);
        return jdbcTemplate.queryForObject("SELECT
* FROM user WHERE id = ?",new Object[]
{id},User.class);
    }

    @Override
    public void save(User user) {
```

```

        jdbcTemplate.update("INSERT INTO
user(username ,password,age)
VALUES(?,?,?)",user.getUsername(),user.getPassword(
),user.getAge());
    }

    @Override
    public void update(User user) {
        jdbcTemplate.update("UPDATE user SET
username = ? ,password = ? , age = ? WHERE id =
?",user.getUsername(),user.getPassword(),user.getAge(),user.getId());
    }

    @Override
    public void deleteById(long id) {
        jdbcTemplate.update("DELETE FROM user
WHERE id = ?",id);
    }
}

```

- 新建一个业务逻辑类来调用方法实现类

```

package com.gloryh.controller;

import com.gloryh.entity.User;
import com.gloryh.repository.UserRepository;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
 * 用户方法测试
 *
 * @author 黄光辉
 * @since 2020/9/29
 */
@RestController

```

```

@RequestMapping("/user")
public class UserHandler {
    @Autowired
    private UserRepository userRepository;

    @GetMapping("/findAll")
    public List<User> findAll() {
        return userRepository.findAll();
    }

    @GetMapping("/findById/{id}")
    public User findById(@PathVariable("id") long
id) {
        return userRepository.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody User user) {
        userRepository.save(user);
    }

    @PutMapping("/update")
    public void update(@RequestBody User user) {
        userRepository.update(user);
    }

    @DeleteMapping("/delete/{id}")
    public void deleteById(@PathVariable("id") long
id) {
        userRepository.deleteById(id);
    }
}

```

六、Spring Boot 整合 MyBatis

- 创建 Maven 工程，pom.xml 导入相关依赖

```

<!--添加父依赖包-->
<parent>
    <groupId>org.springframework.boot</groupId>

```

```

    <artifactId>spring-boot-starter-
parent</artifactId>
    <version>2.3.3.RELEASE</version>
</parent>
<dependencies>
    <!--web相关-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
web</artifactId>
    </dependency>
    <!--MyBatis 和 Spring Boot 整合依赖-->
    <dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-
starter</artifactId>
        <version>2.1.3</version>
    </dependency>
    <!-- MySQL 驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-
java</artifactId>
        <version>8.0.20</version>
    </dependency>

    <!--其他的导入-->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
</dependencies>

```

- 创建一个交互用的数据表,添加几个数据

```

use test;
CREATE TABLE student(
    id INT PRIMARY KEY auto_increment,
    name VARCHAR(11),
    score DOUBLE,
    birthday date
)

```

- 新建数据表对应的实体类

```

package com.gloryh.entity;

import lombok.Data;

import java.util.Date;

/**
 * 学生实体类
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@Data
public class Student {
    private Long id;
    private String name;
    private Double score;
    private Date birthday;
}

```

- 创建一个接口，定义基本的CRUD操作

```

package com.gloryh.repository;

import com.gloryh.entity.Student;

import java.util.List;

/**
 * 学生实体类 CRUD 接口
 *

```

```

* @author 黄光辉
* @since 2020/9/30
**/
public interface StudentRepository {
    /**
     * 查询所有学生信息
     * @return 学生列表
     */
    public List<Student> findAll();

    /**
     * 按 id 查询学生信息
     * @param id
     * @return 学生实体类
     */
    public Student findById(Long id);

    /**
     * 添加一条学生信息
     * @param student
     */
    public void save(Student student);

    /**
     * 修改学生信息
     * @param student
     */
    public void update(Student student);

    /**
     * 按 id 删除学生信息
     * @param id
     */
    public void deleteById(Long id);
}

```

- 在 resources/mapping 路径下创建 MyBatis 相关的 Mapper.xml，用于实现 CRUD 接口类的方法

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```

<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper
3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-
mapper.dtd">
<mapper
namespace="com.gloryh.repository.StudentRepository"
>
    <select id="findAll"
resultType="com.gloryh.entity.Student">
        SELECT * FROM student
    </select>
    <select id="findById"
resultType="com.gloryh.entity.Student">
        SELECT * FROM student WHERE id = #{id}
    </select>
    <insert id="save"
parameterType="com.gloryh.entity.Student">
        INSERT INTO student(name,score,birthday)
VALUES (#{name},#{score},#{birthday})
    </insert>
    <update id="update"
parameterType="com.gloryh.entity.Student">
        UPDATE student SET name = #{name}, score =
#{score}, birthday =#{birthday} WHERE id = #{id}
    </update>
    <delete id="deleteById" parameterType="Long">
        DELETE FROM student WHERE id = #{id}
    </delete>
</mapper>

```

- 创建业务逻辑层，应用相关业务

```

package com.gloryh.controller;

import com.gloryh.entity.Student;
import com.gloryh.repository.StudentRepository;
import
org.springframework.beans.factory.annotation.Autowired;
red;
import org.springframework.web.bind.annotation.*;

```



```

import java.util.List;

/**
 * 学生 CRUD 具体操作调用
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@RestController
@RequestMapping("/student")
public class StudentHandler {
    @Autowired
    private StudentRepository studentRepository;

    @GetMapping("/findAll")
    public List<Student> findAll(){
        return studentRepository.findAll();
    }
    @GetMapping("/findById/{id}")
    public Student findById(@PathVariable("id")
Long id){
        return studentRepository.findById(id);
    }

    @PostMapping("/save")
    public void save(@RequestBody Student student){
        studentRepository.save(student);
    }
    @PutMapping("/update")
    public void update(@RequestBody Student
student){
        studentRepository.update(student);
    }
    @DeleteMapping("/delete/{id}")
    public void delete(@PathVariable("id") Long id)
    {
        studentRepository.deleteById(id);
    }
}

```

- 创建配置文件 application.yml

```

spring:
  datasource:
    url: jdbc:mysql://localhost:3305/test?
    useUnicode=true&characterEncoding=utf8&serverTimezone=UTC&allowMultiQueries=true
    username: admin
    password: 123
    driver-class-name: com.mysql.cj.jdbc.Driver
mybatis:
  mapper-locations: classpath:/mapping/*.xml
  type-aliases-package: com.gloryh.entity

```

- 创建 Spring Boot 启动类 Application

```

package com.gloryh;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Spring Boot 启动类
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@SpringBootApplication
@MapperScan("com.gloryh.repository")
public class Application {
    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);
    }
}

```

注：@MapperScan("com.gloryh.repository") 注解负责将包内的接口自动扫描后交给 IoC 容器，方便我们调用。

- 测试（查询所有学生信息，其余自测）



七、Spring Boot 整合 Spring Data JPA

JPA 是一套规范，Hibernate 框架 就是 JPA 的实现。

Spring Data JPA 并不是对 JPA 规范的具体实现，其本身是一个抽象层。

- pom.xml 导入相关依赖

```
<!--添加父依赖包-->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
parent</artifactId>
    <version>2.3.3.RELEASE</version>
</parent>
<dependencies>
    <!--web相关-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
web</artifactId>
    </dependency>
    <!-- MySQL 驱动 -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-
java</artifactId>
        <version>8.0.20</version>
    </dependency>
    <!-- Spring Data JPA-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-
jpa</artifactId>
    </dependency>
```

- 创建实体类（使用之前的Student实体类，但需要使用注解完成和数据库的映射）

```
package com.gloryh.entity;

import lombok.Data;

import javax.persistence.*;
import java.util.Date;

/**
 * 学生实体类
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Column
    private String name;
    @Column
    private Double score;
    @Column
    private Date birthday;
}
```

注：

`@Entity` 注解：代表该实体类 是要和数据库映射的实体类

`@Id` 注解：表示被注解的属性对应数据表中的主键

`@GeneratedValue` 注解：用于指定被注解的主键的生成策略

`@Column` 注解：表示被注解的属性对应数据表中的对应字段

- 创建一个接口（StudentRepository2），用于实现CRUD操作

由于 JPA 的自动化，常规的CRUD操作不需要定义，直接继承对应的方法接口即可

```
package com.gloryh.repository;

import com.gloryh.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 * 学生实体类 CRUD 接口 --JPA
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
public interface StudentRepository2 extends JpaRepository<Student, Long> {
}
```

注：

继承 JpaRepository 接口后 还需要传递两个泛型参数（即 Student 和 Long），分别代表要操作的数据表的对应实体类，以及 对应数据表的主键 数据类型。

- 创建业务逻辑层（StudentHandler2），应用相关业务

```
package com.gloryh.controller;

import com.gloryh.entity.Student;
import com.gloryh.repository.StudentRepository;
import com.gloryh.repository.StudentRepository2;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
```

```

* 学生 CRUD 具体操作调用 --JPA
*
* @author 黄光辉
* @since 2020/9/30
**/
@RestController
@RequestMapping("/student")
public class StudentHandler2 {
    @Autowired
    private StudentRepository2 studentRepository2;

    @GetMapping("/findAll2")
    public List<Student> findAll() {
        return studentRepository2.findAll();
    }

    @GetMapping("/findById2/{id}")
    public Student findById(@PathVariable("id")
Long id) {
        return
studentRepository2.findById(id).get();
    }

    @PostMapping("/save2")
    public void save(@RequestBody Student student)
{
        studentRepository2.save(student);
    }

    @PutMapping("/update2")
    public void update(@RequestBody Student
student) {
        //JPA 的内部 会使用 Hibernate 的方法进行update操作，即查询该主键对应的数据是否存在，不存在则添加，存在则修改
        studentRepository2.save(student);
    }

    @DeleteMapping("/delete2/{id}")
    public void delete(@PathVariable("id") Long id)
{
        studentRepository2.deleteById(id);
    }
}

```

```
}  
}
```

- application.yml 中进行相关配置

```
spring:  
  datasource:  
    url: jdbc:mysql://localhost:3305/test?  
useUnicode=true&characterEncoding=utf8&serverTimezone=UTC&allowMultiQueries=true  
    username: admin  
    password: 123  
    driver-class-name: com.mysql.cj.jdbc.Driver  
  jpa:  
    show-sql: true  
    properties:  
      hibernate:  
        format_sql: true
```

- 创建启动类 (Application2)

```
package com.gloryh;  
  
import org.springframework.boot.SpringApplication;  
import  
org.springframework.boot.autoconfigure.SpringBootApplication;  
  
/**  
 * Spring Boot 启动类  
 *  
 * @author 黄光辉  
 * @since 2020/9/30  
 **/  
@SpringBootApplication  
public class Application2 {  
    public static void main(String[] args) {  
  
        SpringApplication.run(Application2.class,args);  
    }  
}
```

- 测试（查询所有学生信息，其余自测）



八、Spring Boot 整合 Spring Data MongoDB

MongoDB 是一种非关系型数据库，以BSON（类似于JSON）为数据格式存储数据。

- 创建 Maven 工程，pom.xml 导入相关依赖

```
<!--添加父依赖包-->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
parent</artifactId>
    <version>2.3.3.RELEASE</version>
</parent>
<dependencies>
    <!--web相关-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
web</artifactId>
    </dependency>
    <!--Spring Data MongoDB-->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-
mongodb</artifactId>
    </dependency>

    <!--其他的导入-->
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
</dependencies>
```


- 创建学生实体类(Student3)

```
package com.gloryh.entity;

import lombok.Data;
import org.springframework.data.annotation.Id;
import
org.springframework.data.mongodb.core.mapping.Docum
ent;
import
org.springframework.data.mongodb.core.mapping.Field
;

/**
 * 学生实体类 --MongoDB
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@Data
@Document(collection = "student")
public class Student3 {
    @Id
    private String id;
    @Field(value = "student_age")
    private Integer age;
    @Field(value = "student_name")
    private String name;
}
```

- 对应数据库数据

FILTER

ADD DATA



VIEW



student

	_id ObjectId	student_age String	student_name String
1	5f7412b9c0b4175b91235cbb	"20"	"张三"
2	5f741343c0b4175b91235cbd	"21"	"李四"
3	5f741366c0b4175b91235cbe	"23"	"王五"

- 创建 CRUD 相关操作接口 (StudentRepository3) , 写法类似于 JPA

```
package com.gloryh.repository;

import com.gloryh.entity.Student3;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;

/**
 * 学生实体类 CRUD 接口 --MongoDB
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@Repository
public interface StudentRepository3 extends
MongoRepository<Student3,String> {
}
```

- 创建业务逻辑层 (StudentHandler3) , 应用相关业务

```
package com.gloryh.controller;

import com.gloryh.entity.Student3;
import com.gloryh.repository.StudentRepository3;
```

```
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

/**
 * 学生 CRUD 具体操作调用 --MongoDB
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@RestController
@RequestMapping("/student")
public class StudentHandler3 {

    @Autowired
    private StudentRepository3 studentRepository3;

    @GetMapping("/findAll3")
    public List<Student3> findAll() {
        return studentRepository3.findAll();
    }

    @GetMapping("/findById3/{id}")
    public Student3 findById(@PathVariable("id")
String id) {
        return
studentRepository3.findById(id).get();
    }

    @PostMapping("/save3")
    public void save(@RequestBody Student3 student)
{
        studentRepository3.save(student);
    }

    @PutMapping("/update3")
    public void update(@RequestBody Student3
student) {
```

//MongoDB 的内部 会使用 **Hibernate** 的方法进行 **update**操作，即查询该主键对应的数据是否存在，不存在则添加，存在则修改

```
        studentRepository3.save(student);
    }

    @DeleteMapping("/delete3/{id}")
    public void delete(@PathVariable("id") String
id) {
        studentRepository3.deleteById(id);
    }
}
```

- application.yml 进行相关配置

```
spring:
  data:
    mongodb:
      database: student
      host: 127.0.0.1
      port: 27017
```

- 创建启动类 (Application3)

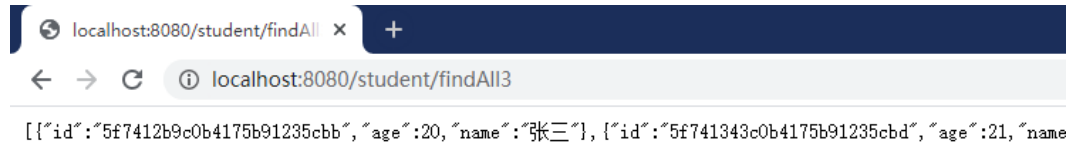
```
package com.gloryh;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Spring Boot 启动类
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@SpringBootApplication
public class Application3 {
    public static void main(String[] args) {
```

```
SpringApplication.run(Application3.class, args);  
    }  
}
```

- 测试（查询所有学生信息，其余自测）



九、Spring Boot 整合 Spring Data Redis

Redis 是一种主流的非关系型数据库，没有表结构，以 键值对 形式存储数据。

- 创建 Maven 工程，pom.xml 导入相关依赖

```
<?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>org.example</groupId>  
  
    <artifactId>spring_boot_mybatis_demo</artifactId>  
    <version>1.0-SNAPSHOT</version>  
    <!-- 添加父依赖包 -->  
    <parent>  
        <groupId>org.springframework.boot</groupId>  
        <artifactId>spring-boot-starter-parent</artifactId>  
        <version>2.3.3.RELEASE</version>  
    </parent>  
    <dependencies>
```

```

        <!--web相关-->
        <dependency>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
web</artifactId>
        </dependency>
        <!--Spring Data Redis-->
        <dependency>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-
redis</artifactId>
        </dependency>
        <!-- Redis 连接所需依赖 -->
        <dependency>
            <groupId>org.apache.commons</groupId>
            <artifactId>commons-pool2</artifactId>
        </dependency>

        <!--其他的导入-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>
</project>

```

- 创建学生实体类，并实现序列化接口

```

package com.gloryh.entity;

import lombok.Data;
import org.springframework.data.annotation.Id;
import
org.springframework.data.mongodb.core.mapping.Docum
ent;
import
org.springframework.data.mongodb.core.mapping.Field
;

```

```

import java.io.Serializable;
import java.util.Date;

/**
 * 学生实体类 --Redis
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@Data
public class Student4 implements Serializable {
    private Long id;
    private String name;
    private Double score;
    private Date birthday;
}

```

- 不需要 Repository 接口类，直接创建业务逻辑层 (StudentHandler4)，应用相关业务（调用 RedisTemplate）

```

package com.gloryh.controller;

import com.gloryh.entity.Student4;
import
org.springframework.beans.factory.annotation.Autowired;
import
org.springframework.data.redis.core.RedisTemplate;
import org.springframework.web.bind.annotation.*;

/**
 * 学生 CRUD 具体操作调用 --redis
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@RestController
@RequestMapping("/student")
public class StudentHandler4 {

    @Autowired

```

```

private RedisTemplate redisTemplate;

@GetMapping("/findByKey4/{key}")
public Student4 findById(@PathVariable("key")
String key) {
    return (Student4)
redisTemplate.opsForValue().get(key);
}

@PostMapping("/save4")
public void save(@RequestBody Student4 student)
{
    redisTemplate.opsForValue().set("stu",
student);
}

@PutMapping("/update4")
public void update(@RequestBody Student4
student) {
    //redisTemplate 的内部 会查询该主键对应的数据是否
存在，不存在则添加，存在则修改
    redisTemplate.opsForValue().set("stu",
student);
}

@DeleteMapping("/delete4/{key}")
public boolean delete(@PathVariable("key")
String key) {
    redisTemplate.delete(key);
    return redisTemplate.hasKey(key);
}
}

```

- application.yml 进行相关配置

```

spring:
  redis:
    database: 0
    host: localhost
    port: 6379

```


- 创建启动类 (Application4)

```
package com.gloryh;

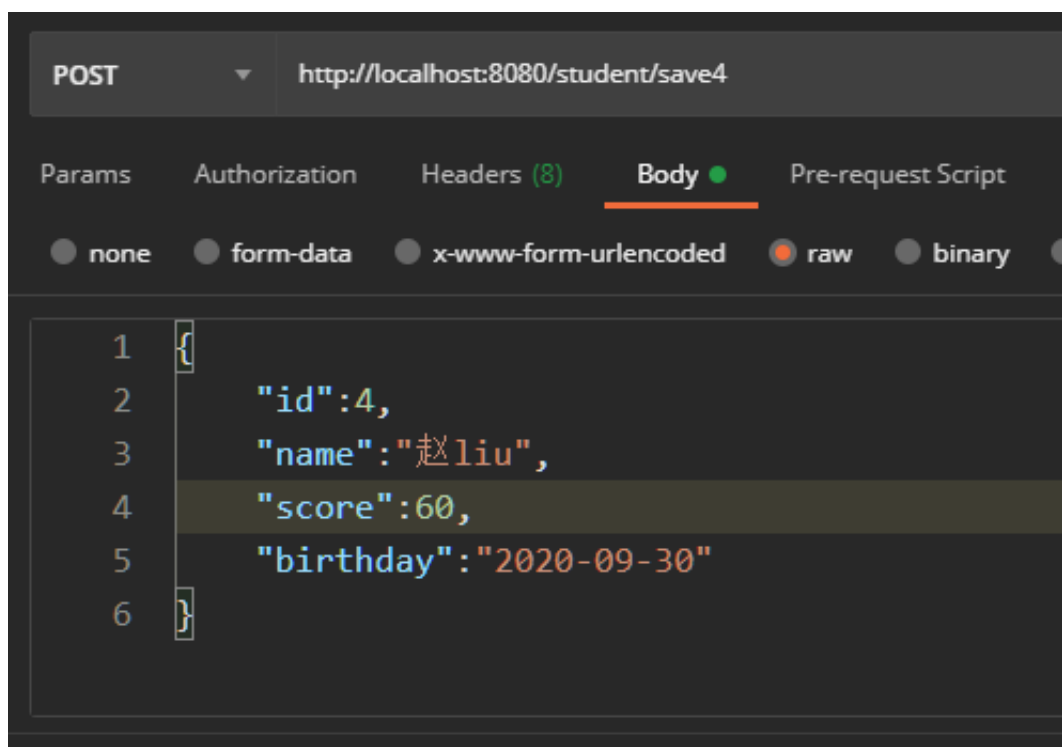
import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Spring Boot 启动类
 *
 * @author 黄光辉
 * @since 2020/9/30
 */
@SpringBootApplication
public class Application4 {
    public static void main(String[] args) {

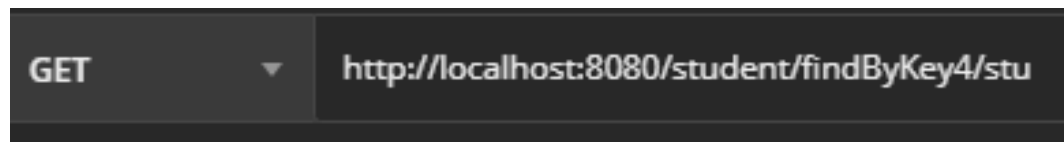
        SpringApplication.run(Application4.class,args);
    }
}
```

- 测试（存储和获取）：

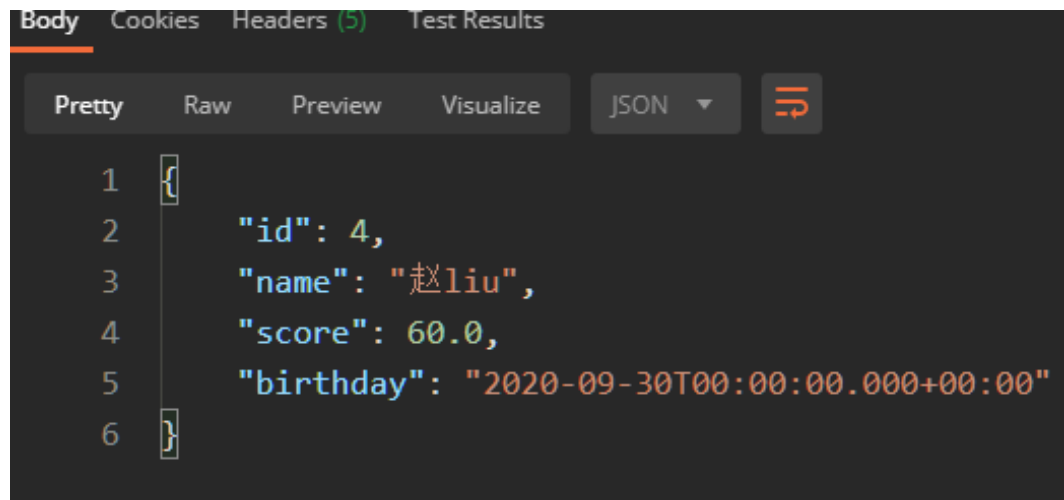
存储：



取出：



结果：



十、Spring Boot 整合 Spring Security

1、快速搭建

- 创建 Maven 工程，pom.xml 导入相关依赖

```
<?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>org.example</groupId>

  <artifactId>spring_boot_mybatis_demo</artifactId>
  <version>1.0-SNAPSHOT</version>
  <!--添加父依赖包-->
  <parent>
    <groupId>org.springframework.boot</groupId>
```

```

        <artifactId>spring-boot-starter-
parent</artifactId>
        <version>2.3.3.RELEASE</version>
    </parent>
    <dependencies>
        <!--web相关-->
        <dependency>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
web</artifactId>
        </dependency>
        <!--Thymeleaf-->
        <dependency>

        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-
thymeleaf</artifactId>
        </dependency>
        <!--Spring Security-->
        <dependency>

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

        <!--其他的导入-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>
</project>

```

- 创建与前端交互的逻辑视图层

```

package com.gloryh.controller;

import org.springframework.stereotype.Controller;

```

```

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

/**
 * Spring Security 测试
 *
 * @author 黄光辉
 * @since 2020/10/2
 */
@Controller
@RequestMapping("/hello")
public class HelloHandler {

    @GetMapping("/index")
    public String index(){}
        return "index";
    }
}

```

- 创建前端页面

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>Hello world!</h1>
</body>
</html>

```

- 创建 application.yml 文件

```

spring:
  thymeleaf:
    suffix: .html
    prefix: classpath:/templates/

```

- 创建启动类 Application5

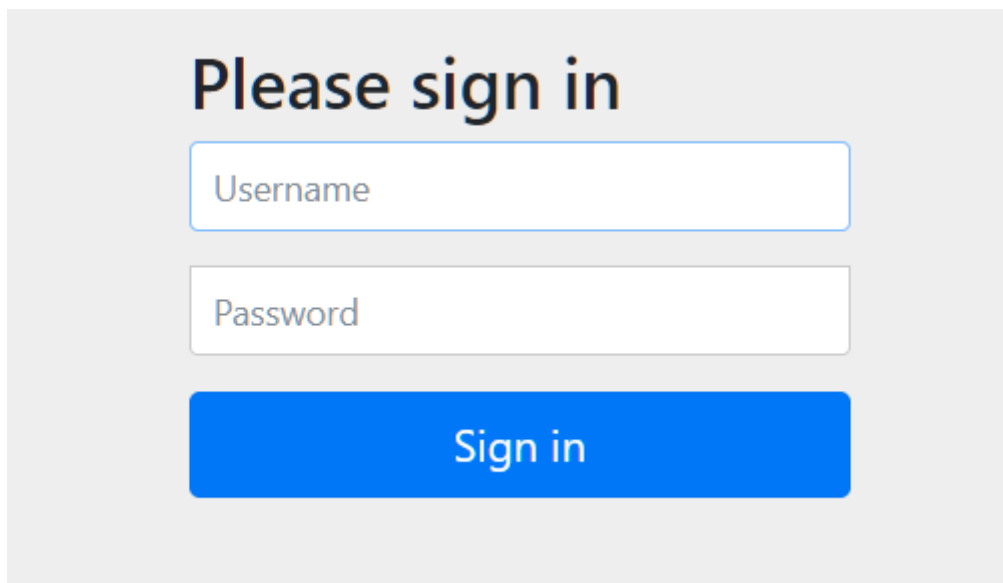
```
package com.gloryh;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Spring Security 整合启动类
 *
 * @author 黄光辉
 * @since 2020/10/2
 */
@SpringBootApplication
public class Application5 {
    public static void main(String[] args) {

        SpringApplication.run(Application5.class, args);
    }
}
```

- 启动测试是否成功 (<http://localhost:8080/hello/index>)



The image shows a web page with a light gray background. At the top, the text 'Please sign in' is displayed in a large, bold, black font. Below this, there are two white input fields with blue borders. The first field is labeled 'Username' and the second is labeled 'Password'. Below these fields is a large blue button with the text 'Sign in' in white. The entire form is centered on the page.

我们在输入测试网址时，会自动跳转到一个登录页面，这是 Security 自带的登录页面，如果我们是未登录状态，就会跳转到登录页面，如果我们没有登录页面，就会跳转到自带的登录页面。

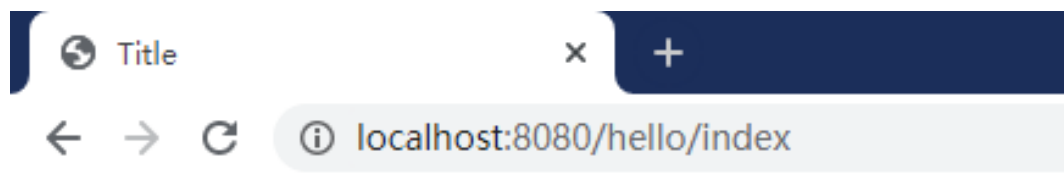
这里的 username 默认为 user，默认密码会打印在控制台：

```
Using generated security password: 1f91e633-aba0-45af-9f8c-70a494287c86
```

输入账户和密码：



点击登录即可访问对应的页面：



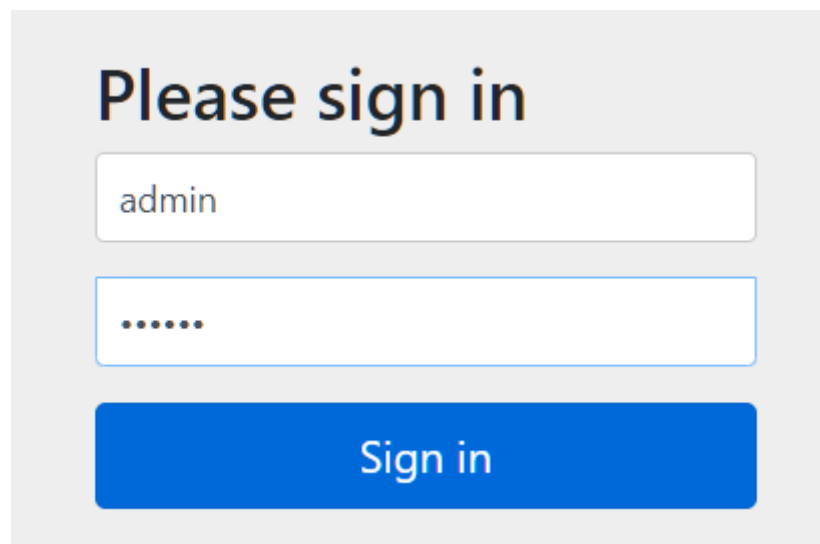
Hello World!

但这些都是Spring Security 自带的，我接下来们需要将这些换成我们自定义的用户名和密码

- 在 application.yml 中可以自定义配置登录的用户名和密码

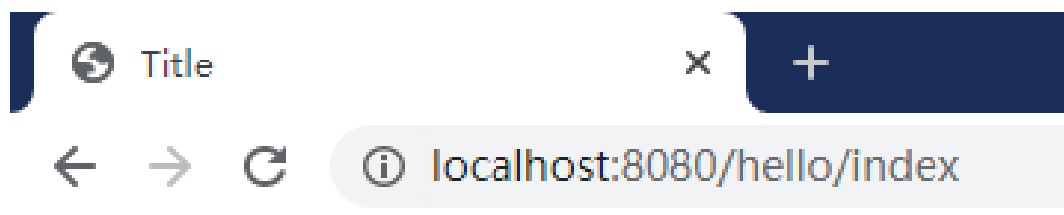
```
spring:
  thymeleaf:
    suffix: .html
    prefix: classpath:/templates/
  security:
    user:
      password: 123123
      name: admin
```

这样我们在验证时输入我们配置的账号和密码即可：



Please sign in

点击登录，完成跳转：



Hello World!

2、权限管理

账号不直接用有权限，账号有对应的角色，角色拥有系统的相应操作权限

定义两个页面：index.html 和 admin.html ,两个角色 ADMIN 和 USER ，拥有 USER 角色的用户 只能访问 index.html 页面，而 拥有 ADMIN 角色的用户 可以访问 index.html 和 admin.html

- 实现密码接口实现类

```
package com.gloryh.config;

import
org.springframework.security.crypto.password.Passwo
rdEncoder;

/**
```

```

* 密码编码口实现类
*
* @author 黄光辉
* @since 2020/10/2
**/
public class MyPasswordEncoder implements
PasswordEncoder {
    @Override
    public String encode(CharSequence charSequence)
    {
        return charSequence.toString();
    }

    @Override
    public boolean matches(CharSequence
charSequence, String s) {
        return s.equals(charSequence.toString());
    }
}

```

- 创建 SecurityConfig 类，继承 Security 提供的 WebSecurityConfigurerAdapter 类，并添加 @Configuration 和 @EnableWebSecurity 表示这是一个 Security 的配置类，重写需要的方法。

```

package com.gloryh.config;

import
org.springframework.context.annotation.Configuratio
n;
import
org.springframework.security.config.annotation.auth
entication.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.web.
builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.
configuration.EnableWebSecurity;

```



```

import
org.springframework.security.config.annotation.web.
configuration.WebSecurityConfigurerAdapter;

/**
 * Spring Security Config
 *
 * @author 黄光辉
 * @since 2020/10/2
 */
@Configuration
@EnableWebSecurity
public class SecurityConfig extends
WebSecurityConfigurerAdapter {
    @Override
    protected void
configure(AuthenticationManagerBuilder auth) throws
Exception {
        //添加账户和角色，设置关系

        auth.inMemoryAuthentication().passwordEncoder(new
MyPasswordEncoder())
            .withUser("user")
            .password(new
MyPasswordEncoder().encode("000"))
            .roles("USER")
            .and()
            .withUser("admin")
            .password(new
MyPasswordEncoder().encode("123"))
            .roles("ADMIN", "USER");
    }

    @Override
    protected void configure(HttpSecurity http)
throws Exception {
        //设置 角色和权限的关系
        http.authorizeRequests()

        .antMatchers("/admin").hasRole("ADMIN")

```

```

.antMatchers("/index").access("hasRole('ADMIN') or
hasRole('USER')")
    .anyRequest().authenticated()
    .and()
    .formLogin()
    .loginPage("/login")
    .permitAll()
    .and()
    .logout()
    .permitAll()
    .and()
    .csrf()
    .disable();
}
}

```

- 修改与前端交互的逻辑视图层

```

package com.gloryh.controller;

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

/**
 * Spring Security 测试
 *
 * @author 黄光辉
 * @since 2020/10/2
 */
@Controller
public class HelloHandler {

    @GetMapping("/index")
    public String index(){
        return "index";
    }

    @GetMapping("/admin")
    public String admin(){

```

```

        return "admin";
    }

    @GetMapping("/login")
    public String login(){
        return "login";
    }
}

```

- 对应的前段页面

index.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>这里是 index 页面</h1>
<form action="/logout" method="post">
    <input type="submit" value="退出"/>
</form>
</body>
</html>

```

admin.html:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
<h1>这里是 admin 页面</h1>
<form action="/logout" method="post">
    <input type="submit" value="退出"/>
</form>
</body>
</html>

```

login.html:

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <form th:action="@{/login}" method="post">
        用户名: <input name="username" type="text"/>
    <br/>
        密码: <input name="password" type="text"/>
    <br/>
        <input type="submit" value="登录"/>
    </form>
</body>
</html>
```

- 启动测试（自测）