配置文件：

application.properties（默认配置文件，对访问端口和访问路径进行配置）：

server.port = 8081

server.context-path=/hjs

或：application.yml（默认配置文件，使用分组格式，:后必须加空格，子内容前为tab键）

server:

port: 8081

context-path: /hjs

yml/xml文件可以进行配置键值对后在java代码层进行注入，如：

在yml中定义：键:值，然后用@Value(“${键}”)的方式在Java代码层进行注入

yml文件也可以在进行配置键值对后在xml层进行注入，如：

server:

port: 8081

context-path: /hjs

@Component = @Service/@Controller/@Repository，将类纳入Spring容器中管理

@ConfigurationProperties (prefix = “girl”) 指定前缀内容

@RestController = @ResponseBody + @Controller

@RequestMapping：

（1）可以指定多个value：

@RequestMapping(value = {“/first”, “/second”})

（2）Get请求：

方式1：url:http://localhost:8080/hjs/16041321/say

@RequestMapping(value = ”/{id}/first”, method = RequestMethod.GET)

public String say(@PathVariable(“id”) Integer id) {

return “Spring Boot:” + id;

}

方式2：url：http://localhost:8080/hjs/say?id=110

@RequestMapping(value=”/say”, method = RequestMethod.GET)

public String say(@RequestParam(“id”) Integer id) {

return “Spring Boot:” + id;

}

添加默认值：

public String say(@RequestParam(value=”id”, required=false, defaultValue=”0”) Integer id) {

return “Spring Boot:” + id;//不传id默认为0

}

方式3：

@GetMapping(value = “/say”)

public String say(@RequestParam(”id”) Integer id){

return “Spring Boot:” + id;

}

@Transactional:事务操作（需要在innodb引擎下事务才能工作）

数据库操作

1、pom.xml添加依赖

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

2、application.yml配置数据库连接

spring:

datasource:

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://127.0.0.1:3306/test

username: root

password: 557523

jpa:

hibernate:

ddl-auto: create #每次都重新创建数据库，表若已经存在，保存数据得用update

show-sql: true #打印sql语句

3、编写pojo对象，添加@Entity注释，标注Id@Id，自增长@GeneratedValue，运行之后数据库就会自动生成对应表：

@Entity

public class Student {

@Id

@GeneratedValue

private Integer id;

private String name;

...

}

4、JPA实现增删改查

（1）查询：

创建接口继承JpaRepository <Student, Integer>

public interface StuRepository extends JpaRepository <Student, Integer> {

//在这里可以自定义扩展方法

//根据姓名查询

public List<Student> findByName(String name);

}

注入接口，调用CRUD：

@Autowired

private StuRepository stuRepository;

//查询所有

@GetMapping(value = “/findAllStus”)

public List<Student> findStus() {

return stuRepository.findAll();

}

//根据id查询

@GetMapping(value = “/findStuById/{id}”)

public Student findStuById(@PathVariable(“id”) Integer id) {

Optional<Student> result = stuRepository.findById(id);

return result.get();

}

//添加

@PostMapping(value=”/SaveStu”)

public Student SaveStu(@RequestParam(“name”) String name) {

Student student = new Student();

student.setName(name);

return stuRepository.save(student);

}

//更新

@PutMapping(value=“/UpdateStu/{id}”)

public Student UpdateStu(@PathVariable(“id”) Integer id, @RequestParam(“name”) String name) {

Student student = new Student();

student.setId(id);

student.setName(name);

return stuRepository.save(student);

}

//删除

@DeleteMapping(value = “/delStu/{id}”)

public void delStu(@PathVariable(“id”) Integer id) {

Student student = new Student();

student.setId(id);

stuRepository.delete(student);

}

//自定义方法：通过姓名查询

@GetMapping(value = “/findStuByName/{name}”)

public List<Student> findStuByName(@PathVariable(“name”) String name) {

return stuRepository.findByName(name);

}

后端进行表单验证（防止黑客绕过前端验证控制）

1、controller层传值改为传对象，然后在对象前面添加@Valid注解对参数进行限制和报错。并添加参数BindingResult来存储报错信息。

@PostMapping(value = “/”)

public Student SaveStu(@Valid Student student, BindingResult bindingResult) {

if(bingdingResult.hasErrors()){

//如果存在报错，输出报错信息

System.out.println(bindingResult.getFieldError().getDefaultMessage());

return null;

}

return stuRepository.save(student);

}

2、@Valid注解对象所对应的Entity层，对属性值进行限制。

@Entity

public class Student {

@Id

@GeneratedValue

private Integer id;

private String name;

@Min(value = 18, message = “18岁以下不得报考”)

private Integer age; //@Min限定最小值。如果小于value，message将被写入BindingResult中

…

}

AOP使用（利用切面技术，在方法执行前和后添加日志打印）

@Aspect //把当前类标识为一个切面供Spring容器读取，指定这是一个切面

@Configuration //代替Spring配置文件，是一个@Component组件，代表这个类是一个配置类

public class HttpAspect {

private final static Logger logger = LoggerFactory.getLogger(HttpAspect.class);

//指定切面的对象，当执行切面对象的代码时，切面代码就会切入到指定对象代码中，实现日志记录的功能

@Pointcut(“execution(public \* com.hjs.controller.StudentController.\*(..))”)

public void log(){

}

//方法执行前执行

@Before(“log()”)

public void doBefore(){

logger.info(“进入切入点”);

}

//方法执行后执行

@After(“log()”)

public void doAfter(){

logger.info(“退出切入点”);

}

}

接口规范（调用接口，每次返回的内容都不一样，）

public class ResultUtil {

public static Result success(Object object) {

Result result = new Result();

result.setCode(0);

result.setMsg(“成功”);

result.setData(object);

return result;

}

public static Result error(Integer code, String msg) {

Result result = new Result();

result.setCode(code);

result.setMsg(msg);

return result;

}

}

Controller层：

@PostMaping(value = “/SaveStu”)

public Result<Student> SaveStu(@Valid Student student, BindingResult bindingResult) {

if(bindingResult.hasErrors()) {

return ResultUtil.error(1, bindingResult.getFieldError().getDefaultMessage());

}

return ResultUtil.success(studentRepository.save(student));

}

测试Service层

@RunWith(SpringRunner.class)

@SpringBootTest

public class StudentServiceTest {

@Autowired

private StudentService studentService;

@Test

public void findOne() {

Student student = studentService.studentFindOne(16041321);

Assert.assertEquals(new String(“黄继升”), student.getName())

}

}

测试Controller层

@RunWith(SpringRunner.class)

@SpringBootTest

@AutoConfigureMockMvc #使用mvc对象进行接口测试

public class StudentControllerTest {

@Autowired

private StudentService studentService;

@Autowired

private MockMvc mockMvc;

@Test

public void findBestStudent throws Exception {

//返回状态码正确

mockMvc.perform(MockMvcRequestBuilders.post(“/students/findBestStudent”))

.andExpect(MockMvcResultMatchers.status().isOk());

//也可以对返回的内容进行测试

mockMvc.perform(MockMvcRequestBuilders.post(“/students/findBestStudent”))

.andExpect(MockMvcResultMatchers.status().isOk())

.andExpect(MockMvcResultMatchers.content().string(“hjs”));

}

}