## Mysql 客观题

2 4 3 2 2 2

## MySQL编程题

## <https://github.com/huyuanixn325/SQl-Base-2019-5-17-2-0-39-612.git>

## Orcale测验题

2 4 3 4 2

## Oracle编程题

## <https://github.com/huyuanixn325/SQl-Base-2019-5-17-2-0-39-612.git>

## DDL和DCL 测验

4 3 2 1

## DDL-DML编程题

## <https://github.com/huyuanixn325/DDL-DML-Base-2019-5-17-7-33-10-495.git>

## 索引和多表查询-测验题

3 1 3 3

索引和多表查询-编程题

<https://github.com/huyuanixn325/multi-table-query-2019-5-21-3-10-2-585.git>

Spring IoC基础

* IOC:Inversion of Control控制反转，也叫（Dependency Injection）依赖注入。IoC 不是一种技术，只是一种思想。它能指导我们如何设计出松耦合、更优良的程序。比如在程序中，依赖注入就是利用某种工具，将依赖注入到需要的位置。Spring IOC 负责创建对象，管理对象（通过依赖注入（DI），装配对象，配置对象，并且管理这些对象的整个生命周期。 依赖注入还有另一层意思：就是依赖第三方工具完成注入的操作。依赖注入的核心原理是注解和反射。
* 优点是 内存控制：统一管理对象，避免对象乱创建导致额外的内存开销。便于内存的优化。 降低耦合度：便于项目的扩展、易于维护。如果IoC+接口情况下，删除任意实现类都不会导致程序编译出错。虽然运行到特定得代码会报错，但是其他代码在使用时不会有问题-----从侧面也反应出是松耦合。

## SpringIoC-practice

<https://github.com/huyuanixn325/SpringIoC-practice-2019-5-23-6-38-50-100.git>

## SpringBoot中常见的auto-configuration注解

* @ConditionalOnClass（某个class位于类路径上，才会实例化一个Bean）
* @ConditionalOnMissingClass（某个class类路径上不存在的时候，才会实例化一个Bean）
* @ConditionalOnBean（仅仅在当前上下文中存在某个对象时，才会实例化一个Bean）
* @ConditionalOnMissingBean（仅仅在当前上下文中不存在某个对象时，才会实例化一个Bean）
* @ConditionalOnProperty Spring Boot通过@ConditionalOnProperty来控制Configuration是否生效，通过其两个属性name以及havingValue来实现的，其中name用来从application.properties中读取某个属性值。如果该值为空，则返回false;如果值不为空，则将该值与havingValue指定的值进行比较，如果一样则返回true;否则返回false。如果返回值为false，则该configuration不生效；为true则生效。
* @ConditionalOnResource 类路径是否有指定的值
* @ConditionalOnWebApplication当前项目是Web项目的条件
* @ConditionalOnNotWebApplication 当前项目不是Web项目的条件
* @ConditionalOnExpression 基于SpEL表达式作为判断条件

## SpringBoot自动配置基础

3 2 4 spring.profiles.active=prod

## 属性配置文件练习

<https://github.com/huyuanixn325/Spring-AutoConfiguration-2019-5-23-7-44-30-444.git>

## 创建SpringBoot项目

# 1.HelloworldApplication 类

**package** com.example.helloworld;

**import** org.springframework.boot.SpringApplication;

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

**@SpringBootApplication**

**public** **class** **HelloworldApplication** {

**public** **static** **void** **main**(String[] args) {

SpringApplication.run(HelloworldApplication.class, args);

}

}

2.HelloController 类

**package** **com**.example.helloworld;

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

**import** **org**.springframework.web.bind.annotation.RequestMethod;

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

@**RestController**

public class HelloController {

@**RequestMapping**(**value** = "",method = RequestMethod.GET)

public String test(){

**return** "**Hello** **SpringBoot**";

}

}

3.HelloworldApplicationTests类

**package** com.example.helloworld;

**import** org.junit.Assert;

**import** org.junit.Before;

**import** org.junit.Test;

**import** org.junit.runner.RunWith;

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

**import** org.springframework.boot.test.context.SpringBootTest;

**import** org.springframework.http.MediaType;

**import** org.springframework.test.context.junit4.SpringRunner;

**import** org.springframework.test.web.servlet.MockMvc;

**import** org.springframework.test.web.servlet.MvcResult;

**import** org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

**import** org.springframework.test.web.servlet.result.MockMvcResultHandlers;

**import** org.springframework.test.web.servlet.setup.MockMvcBuilders;

**import** org.springframework.web.context.WebApplicationContext;

**import** java.awt.\*;

**import** java.io.UnsupportedEncodingException;

**@RunWith**(SpringRunner.class)

**@SpringBootTest**

**public** **class** **HelloworldApplicationTests** {

**@Autowired**

**private** WebApplicationContext webApplicationContext;

**private** MockMvc mockMvc;

**@Before**

**public** **void** **setUp**() **throws** Exception{

mockMvc = MockMvcBuilders.webAppContextSetup(webApplicationContext).build();

}

**@Test**

**public** **void** **contextLoads**() **throws** Exception {

MvcResult mvcResult = mockMvc.perform(MockMvcRequestBuilders.get("").accept(MediaType.TEXT\_HTML\_VALUE)).andDo(MockMvcResultHandlers.print())

.andReturn();

**int** status = mvcResult.getResponse().getStatus();

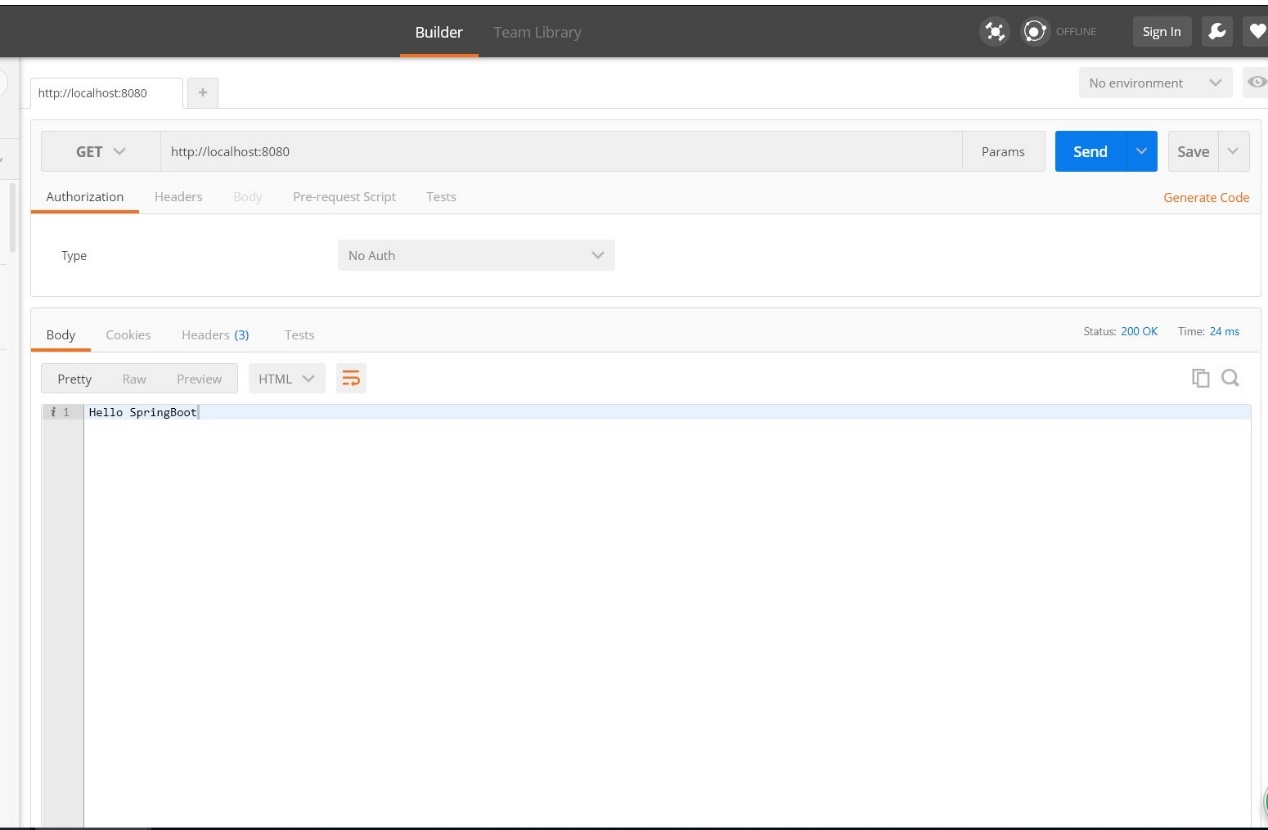
String content = mvcResult.getResponse().getContentAsString();

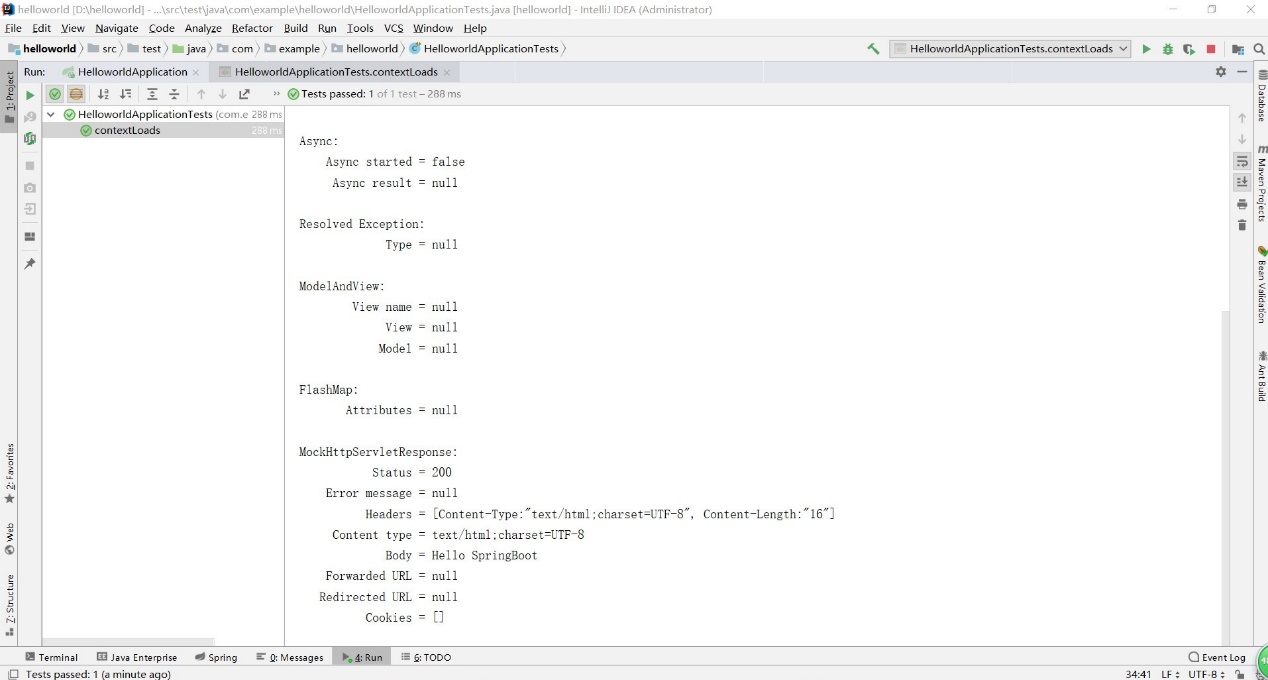
Assert.assertEquals(200,status);

Assert.assertEquals("Hello SpringBoot",content);

}

}





## http

1 3 1 3 2 2

## SpringBoot-Restful

[git@github.com:huyuanixn325/Spring-Boot-Restful-2019-5-7-2-41-28-647.git](mailto:git@github.com:huyuanixn325/Spring-Boot-Restful-2019-5-7-2-41-28-647.git)

## Postman基本使用

<https://github.com/huyuanixn325/JAVA-CRUD-2019-5-7-2-33-44-100.git>

## postman使用基础

application/xml

123

124

## MyBatis基础总结

###### 1.什么是MyBatis？

MyBatis是一个实现了JPA规范的用来连接数据库并对其进行增删改查操作的开源框架 （就和传统的JDBC一样，就是个连接数据库的东西），其实，它底层就是一个JDBC封装的组件。MyBatis的前身是Ibatis，Ibatis创建与2002年最初为Apache下面的一个开源项目，2010迁移到google code下面并改名为MyBatis。

###### 2.优点:

* 基于SQL语法，简单易学。
* 能了解底层组装过程。
* SQL语句封装在配置文件中，便于统一管理与维护，降低了程序的耦合度。
* 程序调试方便。
* 与传统JDBC比较较少了大量的代码量，是最简单的持久化框架。
* sql代码从程序代码中彻底分离，可重用，增强了项目中的分工，增强了移植性。

###### 缺点：

* sql工作量很大，尤其是字段多、关联表多时，更是如此。
* sql依赖于数据库，导致数据库移植性差。

###### 3. Mybatis有两种工作方式，一种是通过Mapper接口的工作方式，另一种是通过SqlSession执行sql的传统Mybatis工作方式。

###### 4. 一种是通过注解绑定，就是在接口的方法上面加上 @Select、@Update等注解，里面包含Sql语句来绑定；

@**Select**("select \* from `tb\_Teacher` where id = #{id}")

**public** teacher selectTeacherByID(int **id**);

另外一种就是通过xml里面写SQL来绑定, 在这种情况下,要指定xml映射文件里面的namespace必须为接口的全路径名。

<**select** id="selectUserByID" parameterType="int" resultType="test.student">

**select** id,name,age,stuCountry stu\_country **from** `tb\_Student` **where** id = **#{id}**

</**select**>

第三种，是第一种的特例，也是通过注解绑定，在接口的方法上面加 上 @SelectProvider、@UpdateProvider、@InsertProvider、@DeleteProvider 注解，通过 Java 代码，生成对应的动态 SQL 。

###### 5. 当Sql语句比较简单时候,用注解绑定, 当SQL语句比较复杂时候,用xml绑定,一般用xml绑定的比较多。

###### 6. MyBatis的动态SQL是基于OGNL表达式的，它可以帮助我们方便的在SQL语句中实现某些逻辑。MyBatis的动态SQL主要就是为了解决手动拼接SQL的麻烦 .

## MyBatis基础练习

<https://github.com/huyuanixn325/Spring-MyBatis-basis-2019-5-7-2-56-28-816.git>