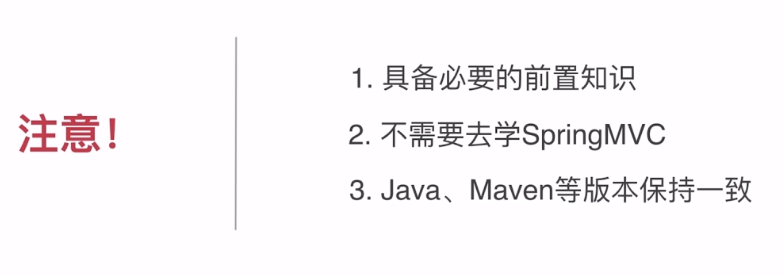
2018年8月31日11:17:12

1. 前面的铺垫文章已经连着写了六篇了，主要是介绍了Spring和SpringMVC框架，小伙伴们在学习的过程中大概也发现了这两个框架需要我们手动配置的地方非常多，不过做JavaEE开发的小伙伴们肯定也听说过“约定大于配置”这样一句话，就是说系统，类库，框架应该假定合理的默认值，而非要求提供不必要的配置，可是使用Spring或者SpringMVC的话依然有许多这样的东西需要我们进行配置，这样不仅徒增工作量而且在跨平台部署时容易出问题。OK，由于这些已经存在的问题，Spring Boot应运而生，使用Spring Boot可以让我们快速创建一个基于Spring的项目，而让这个Spring项目跑起来我们只需要很少的配置就可以了

2018年9月2日10:00:33

1. 学习spring boot的前置知识
2. 利用maven构建项目
3. Spring 注解
4. RESTful API





1. 默认的maven会非常慢，可以使用阿里云的镜像，需要修改maven的settings.xml文件



1. 师兄给的maven 是用了学校的服务器，有时候连不上

我的电脑上安装位置：

安装在E:\apache-maven-3.5.0-with-nexus

本地仓库：F:\Maven Repository

1. 手动启动项目



（1）首先要安装maven，本质上就是配置环境变量。

在<http://maven.apache.org/download.cgi> 下载maven，解压放到本地

在系统环境变量里面新建MAVEN\_HOME,值为解压后的目录

然后在path里面添加%MAVEN\_HOME%\bin 即可

（2）在命令行里面测试maven安装是否成功，输入mvn -version

1. 运行spring-boot项目

在命令行里面进入这个使用idea创建的项目目录，输入命令： mvn spring-boot：run

1. 手动启动项目方法2

使用mvn install方法

(1)首先还是要配置好maven，能在命令行里面使用mvn命令

(2)然后命令行进入项目的目录，输入mvn -install 命令，这样就能在项目的target目录下输出jar包

(3) 使用java -jar来启动项目

在命令行里面输入： java -jar target/demo-0.0.1-SNAPSHOT.jar

然后这个项目就启动了

这个命令后面还可以添加参数 java -jar target/demo-0.0.1-SNAPSHOT.jar –spring.profiles.active=prod

1. 使用配置文件来设置端口和前缀

在前面的项目的基础上修改。可以直接配置application.properities文件，也可以在resources文件夹下新建一application.yml文件，来配置，

内容如下：**server:  
 port:** 8081  
 **servlet:  
 context-path:** /girl

这样访问的URL就是<http://localhost:8081/girl/hello>

尽量使用yml来配置，而不是默认的文件来配置

1. SpringBoot 使用 @Value 从 YAML文件读取属性

在yml文件里增加一个属性：

cupSize: B 注意：者属性是自定义的，并且冒号后面有空格，然后才是值

然后在java代码里面就可以使用@Value注解读取了

@Value(**"${cupSize}"**)  
**private** String **cupSize**;

1. 一定要注意属性的后面要有空格，否则会报IllegalArgumentException异常
2. 在yml的配置文件里面使用配置



1. 在idea里面给变量增加get和set方法

选中变量，使用alt+enter 或者右键generate，然后选中getter或者setter就行了。

1. 对配置进行分组

使用@ConfigurationProperties 来读取配置文件

在yml里面增加 ：

girl:  
 cupSize: B  
 age: 18

注意名字不要写错和空格

增加一个GirlProperities类来保存这个配置

@Component  
@ConfigurationProperties(prefix = **"girl"**)  
**public class** GirlProperities {  
 **private** String **cupSize**;  
 **private** Integer **age**;  
 **public** String getCupSize() {  
 **return cupSize**;  
 }  
 **public** Integer getAge() {  
 **return age**;  
 }  
 **public void** setCupSize(String cupSize) {  
 **this**.**cupSize** = cupSize;  
 }  
 **public void** setAge(Integer age) {  
 **this**.**age** = age;  
 }  
}

注意类前面的两个注解

@Component  
@ConfigurationProperties(prefix = **"girl"**)

然后就可以在代码里面使用了，

@Autowired  
private GirlProperities girlProperities;

注意需要添加注解

调用方法：**return girlProperities**.getAge().toString();

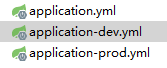
注意还要在pom.xml文件里面添加依赖

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-configuration-processor</**artifactId**>  
 <**optional**>true</**optional**>  
</**dependency**>

1. 多环境配置

使用多个配置文件，一个座位开发时候用的配置，另一个座位生产时的配置，这样可以增加灵活更换配置，增加效率

增加两个新的yml文件



注意后缀

后里面两个yml文件的内容格式一致，只是值不一样

然后在第一个文件application.ym里面修改为如下内容：

**spring:  
 profiles:  
 active:** prod

注意第一行没有空格，第二行两个空格，第三行4个空格

Active的值表明使用哪个配置文件，也可以是dev

这样就能很容易的切换配置了。

1. Controller的使用



1. 使用模板thymeleaf

（1）先增加 依赖

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

注意，使用idea启动项目的时候要确保之前用命令行启动的项目关闭了，否则无法启动。

注意上面的不要写错了。

（2）增加html的模板

在resources文件夹下新建templates文件夹，然后添加一个html文件，内容就是普通html内容

（3） 修改controller类，

将原来的@RestController注解修改为@Controller，其他不变，将返回的代码写为：return ”index“ 这个对应于index.html文件。

然后就能在浏览器里面访问这个html页面了

使用模板这种方式不是常用的方式，一般都是前后端分离。

15． 如果要返回GirlProperities的属性，可以这样修改注解：

@Controller  
@ResponseBody  
public class HelloController {

1. 如果想以多个url访问同一个方法，可以修改注解为：

@RequestMapping(value = {**"/hello"**,**"/hi"**},method = RequestMethod.***GET***)

1. 给类设置url

在类名前增加：@RequestMapping(value = **"/hello"**)

方法前修改为：@RequestMapping(value = **"/say"**,method = RequestMethod.***GET***)

然后方法的URL需要拼接，<http://localhost:8082/hello/say>

2018年9月2日22:21:22





1. 获取url的参数

@RequestMapping(value = **"/say/{id}"**,method = RequestMethod.***GET***)  
 **public** String hello(@PathVariable(**"id"**) Integer id){**return "id:"**+id;}

也可以把id写到前面

@RequestMapping(value = **"/{id}/say/"**,method = RequestMethod.***GET***)

1. 获取传统方式的URL参数

比如:

<http://localhost:8082/hello/say？id=232>

修改方法如下：

@RequestMapping(value = **"/sing"**,method = RequestMethod.***GET***)  
**public** String sing(@RequestParam(**"id"**) Integer myid )  
{  
 **return "id:"**+myid;  
}

访问方法：<http://localhost:8082/hello/sing?id=2324>

@RequestParam(**"id"**)里面的id要和url一致。

1. 设置参数的默认值

@RequestMapping(value = **"/sing"**,method = RequestMethod.***GET***)  
**public** String sing(@RequestParam(value =**"id"**, required = **false**,defaultValue = **"453"**) Integer myid )  
{  
 **return "id:"**+myid;  
}

访问方法：<http://localhost:8082/hello/sing>

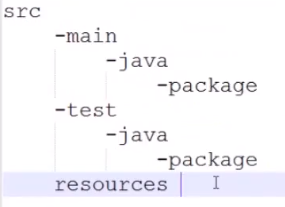
1. 如果嫌@RequestMapping 的写法太长，可以使用 @GetMapping(value = "/sing")

除此之外还有PostMapping，PutMapping等。

2018年9月3日09:30:56

1. 使用maven编写第一个helloworld程序

首先是maven的项目目录结构



然后需要在src的同级，也就是项目根目录下添加pom.xml文件，可以从别的地方复制

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

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.imooc.maven01</groupId>

<artifactId>maven01-model</artifactId>

<version>0.0.1SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.10</version>

</dependency>

</dependencies>

</project>

在两个java目录下新建java源码，注意package是一层层的文件夹。

然后在命令行里面进入项目根目录，使用mvn compile来编译项目，mvn test来运行单元测试。

输入mvn compile后，会在项目根目录下生成target目录，里面会有生成的class文件等其他文件。

mvn package是打包工具

1. MVN 的常用构建命令

mvn -v 查看版本

compile 编译项目

test 测试

package 打包

clean 删除target目录

install 将jar包到本地仓库中

1. 使用archetype插件自动创建目录

之前都是手动创建目录的，非常麻烦，可以使用archetype插件创建符合maven规定的目录

命令行直接进入新项目的空白根目录，输入**mvn archetype:generate** 命令，第一次会下载一些依赖

然后会输入版本号，groupId，artifactId，version,package 信息。

这样就会创建相应的目录和java文件了，并且包含了对应的test测试文件。

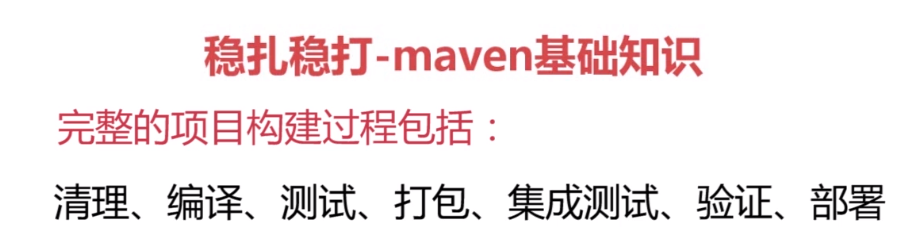
groupId 一般是公司网址的反写+项目名

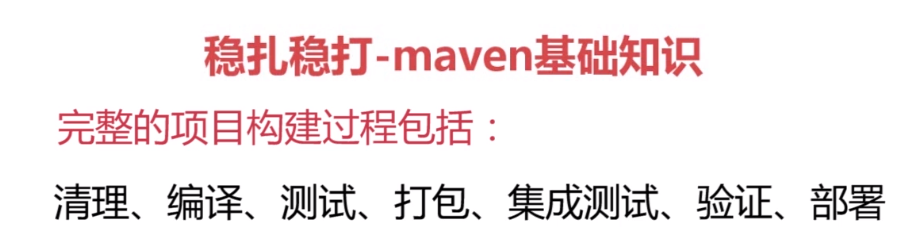
artifactId 一般是项目名+模块名

1. Eclipse中安装maven插件

如果eclipse是4.0及以上的版本或者是myeclipse就不用再安装maven插件了，因为自带了。

可以在preferences里面查看是否有maven插件





1. pom 中的常见元素

project modelVersion

groupId 反写公司网址+项目名

artifactId 项目名+ 模块名

version 第一个0表示大版本号 第二个0表示分支版本号 第三个0表示小版本号

0.0.1snapshot 快照

alpha 内部测试

beta 公测

Release 稳定

GA 正式发布

Packaging 打包方式，默认是jar包，也可以是war，zip，pom

<**groupId**>com.example</**groupId**>  
<**artifactId**>demo</**artifactId**>  
<**version**>0.0.1-SNAPSHOT</**version**>  
<**packaging**>jar</**packaging**>

其他标签：

name

url

description

developers

licenses

organization

<dependencies> 依赖

<dependency>

<**dependency**>  
 <**groupId**></**groupId**>  
 <**artifactId**></**artifactId**>

<version></version>

</**dependency**>  
<type></type>  
 <**scope**>test</**scope**>

<optional></optional> 设置依赖是否可选

<exclusions> 排除依赖的列表

<exclusion></exclusion>

</exclusions>

<dependencyManagement></dependencyManagement> 依赖管理

<**build**>  
 <**plugins**> 插件列表  
 <**plugin**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-maven-plugin</**artifactId**>  
 </**plugin**>  
 </**plugins**>  
</**build**>

<**parent**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-parent</**artifactId**>  
 <**version**>2.0.4.RELEASE</**version**>  
 <**relativePath**/> *<!-- lookup parent from repository -->*</**parent**>

<modules></modules>

1. maven的依赖范围

主要是dependency中的scope标签

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

使用依赖就会将相应的jar包放到项目的classpath中，maven提供了三种classpath：

1. 编译
2. 测试
3. 运行

Junit也是test的范围

Scope的值有六种：

1. Compile 编译测试运行都有效
2. provided 在编译和测试的时候有效，运行无效，比如编译测试时需要，而运行环境已经存在了，就不再需要了，就可以使用provided
3. runtime 比如JDBC
4. test ，比如junit
5. system 编译和测试有效，使用本地javahome
6. import 只使用在dependencymanagement中，表示从其他的pom中导入dependency的配置

默认的范围是compile，编译测试运行都有效

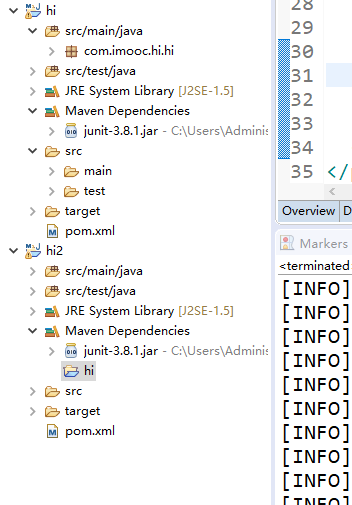
1. maven依赖的传递

一个项目若依赖另一个项目的依赖，只需将后者的坐标复制到前者pom的dependency中。

坐标就是groupid artifact version

但是需要先将后者编译，打包，导入仓库，依次使用maven 命令： compile clean package install

我建立了两个maven项目

，

hi2 项目依赖hi项目，先将第一个项目编译、打包、安装。然后将坐标直接复制到第二个项目即可，然后编译第二个项目，可以看到Maven Dependency里面有hi这个项目。

如果A 依赖B,B依赖C，那么A也依赖C。

在上面的关系下，如果A不想依赖C，那么可以使用exclusion来排除依赖

<exclusions>

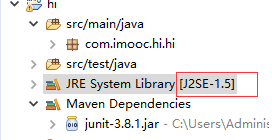
<exclusion>

C的坐标

</exclusion>

</exclusions>

1. 在eclipse中执行maven命令的方法： 右击maven项目，选中run as，就可以看到 maven build 命令，有两个，选后者，然后在出来的对话框中的Golas里面输入命令，点击run按钮即可。第一个maven build命令是选择最近使用过的maven命令。
2. 新建的项目的JRE默认是1.5 实际上可能是1.7或1.8



可以右键JRE Ssystem Library ，选择properties来修改。这样效率不高，每次新建项目后都要重新修改。可以在maven的settings.xml里面修改。

在porfile标签下修改activation标签里面的JDK版本

<activation>

<jdk>1.4</jdk>

</activation>

1. JavaEE 与 JavaSE区别

Java SE（Java Platform，Standard Edition）。Java SE 以前称为 J2SE。它允许开发和部署在桌面、服务器、嵌入式环境和实时环境中使用的 Java 应用程序。Java SE 包含了支持 Java Web 服务开发的类，并为 Java Platform，Enterprise Edition（Java EE）提供基础。

Java EE（Java Platform，Enterprise Edition）。这个版本以前称为 J2EE。企业版本帮助开发和部署可移植、健壮、可伸缩且安全的服务器端 Java 应用程序。Java EE 是在 Java SE 的基础上构建的，它提供 Web 服务、组件模型、管理和通信 API，可以用来实现企业级的面向服务体系结构（service-oriented architecture，SOA）和 Web 2.0 应用程序。

简单来说的话就是：

Java SE 就是做电脑上的运行软件

Java EE 就是用来做网站的 （例如JSP、Html5、CSS3）

1. Mavne 依赖的查询

这maven官网的这个地址<http://mvnrepository.com/> 搜索想要的依赖，会出现dependency格式，直接复制即可。

<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->

<dependency>

<groupId>commons-io</groupId>

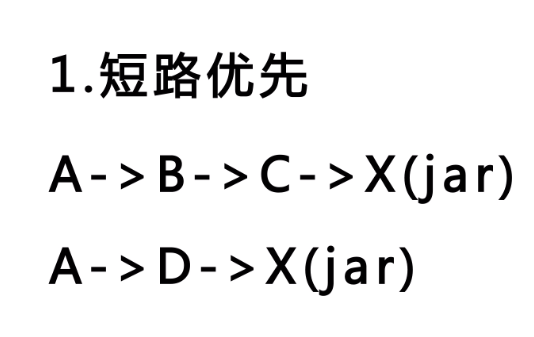
<artifactId>commons-io</artifactId>

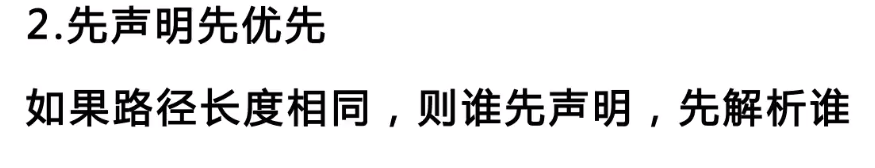
<version>2.6</version>

</dependency>

在eclipse里面修改了pom，直接保存就可以下载新的依赖包，而在idea里面要右键pom文件，然后选择mavne->reimport命令

1. maven 的依赖冲突
2. 短路优先





A 同时依赖B和C，B和C之间不相互依赖，B和C之间都有commons-io的依赖，但是版本不一样。在A中，依赖B的dependency在依赖C的dependency前面，那么A先依赖B,使用B中commons-io的版本。

1. Maven的聚合和继承
2. 聚合：将多个maven项目一起install到仓库中

新建一个新的maven项目，将packaging修改为pom，

增加一个modules标签，添加module

<modules>

<module></module>

<module></module>

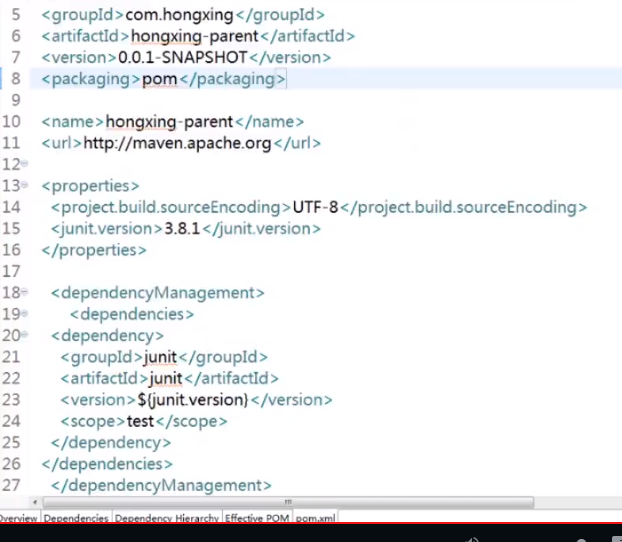
<module></module>

</modules>

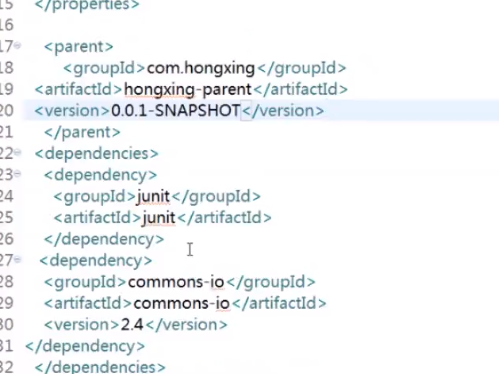
然后再编译打包安装这个新的项目即可。

1. 继承: 多个项目都包含了junit依赖，重复了很多内容，可以提取到一个新的项目，让别的子项目都继承它

新建一个maven项目，修改pom，增加dependencyManagement标签，将packaging修改为pom。

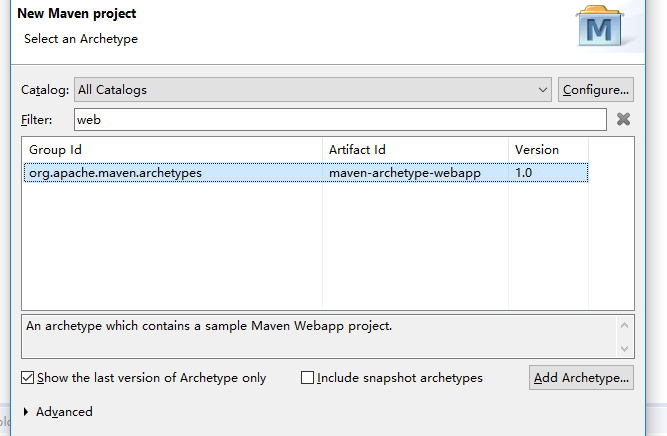


在子项目的pom中，增加parent标签，里面存父项目的坐标。

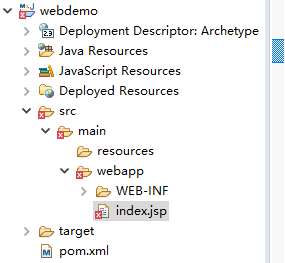


2018年9月3日21:03:31

1. 使用mavne创建一个maven项目，并发布到jetty中
2. 新建一个maven项目，选择web项目



新项目的目录结构如下：



其中index.jsp有红色的叉，是因为没有servelet api的缘故。

在maven的网站搜索servlet，选择想要的版本，将的生成的dependency加入项目的pom中。

<dependency>

<groupId>javax.servlet</groupId>

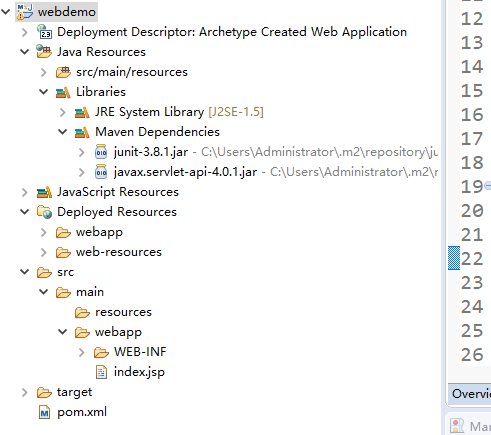
<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

注意scope为provided，表明只在编译和测试时运行

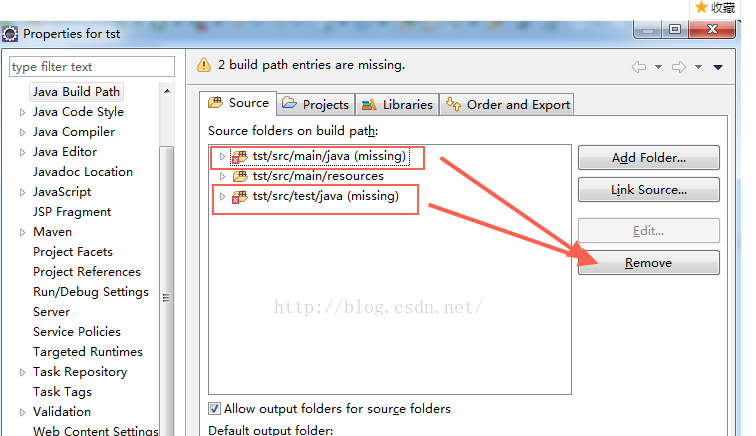


解决jsp红色错误的另一个方法是：添加tomcat的jar包。找到项目的build path，在libraries下选择add library，选择server runtime，然后选择tomcat，前提是已经安装配置过tomcat了。

1. 这时创建的项目，只创建了Resources目录，还没有创建java代码的目录，不符合maven的目录结构。需要手动创建。

视频教程上的方法是：右键项目的Java Resources选择new ->source folder,选择项目名称，和目录名称，目录名称要写src/main/java

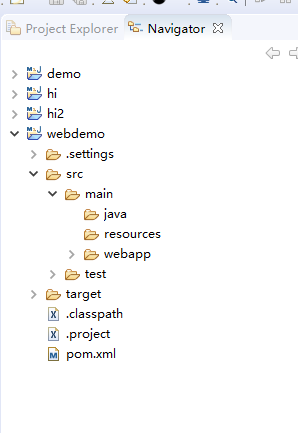
但是我却无法创建这样的目录。网上说的是默认的classpath已经有这个目录了，右击项目名，选择properties，然后选择java build path，可以看到src/main/java是缺失的，



解决方法是直接在这里选择则两项，点Remove删除即可。然后使用上面的方法创建source folder

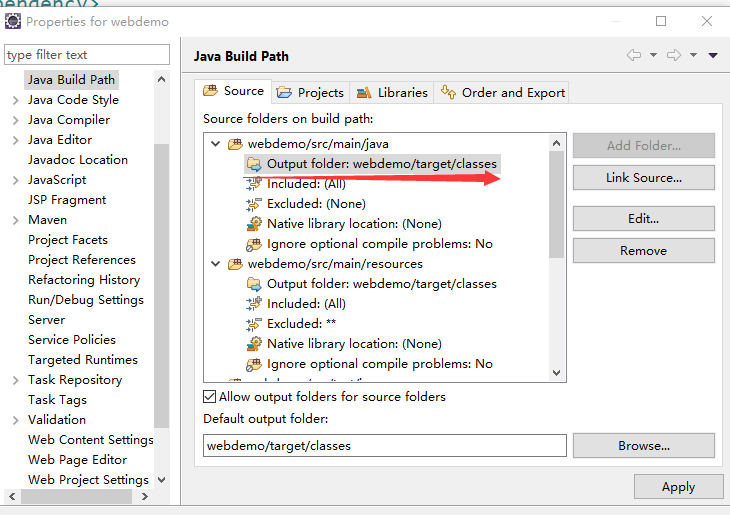
然后创建测试的目录，基本和前面一样，不一样的是目录名为src/test/java

无法创建目录的另一种解决方法是，打开navigator视图，然后创建。



创建好目录后，右键项目，选择maven->update project

然后找到项目属性的build path，去报输出路径是target目录下

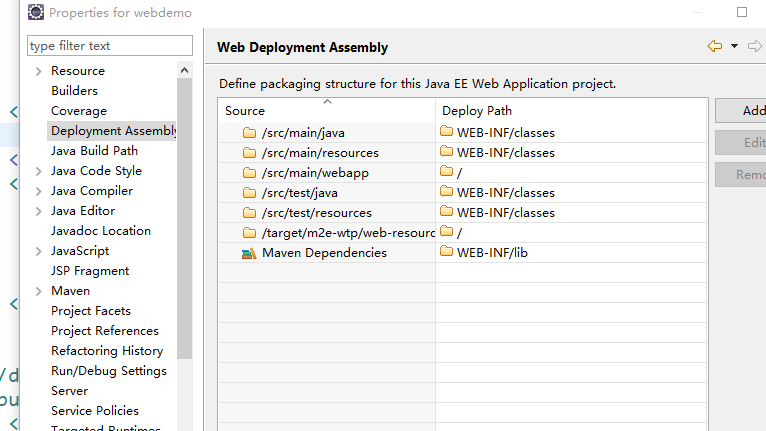


1. 接下来将项目转化为web项目。

右键项目属性，选择project facets，将右边的dynamic web module 勾上。

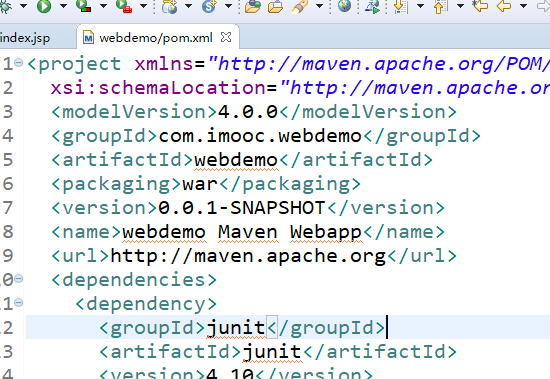
1. 接下来修改部署时的默认配置，

默认如下：



发布时不需要测试，因此需要将测试的删掉

1. 然后用package命令将其打包成war，放到支持的web容器，比如jetty，就可以访问了。



1. 在maven网站搜索 jetty插件，搜索jetty maven plugin

网上依然是dependency，但只需要将里面的坐标复制出来，放到build下的plugins的plugin里面。

<build>

<finalName>webdemo</finalName>

<plugins>

<plugin>

<groupId>org.eclipse.jetty</groupId>

<artifactId>jetty-maven-plugin</artifactId>

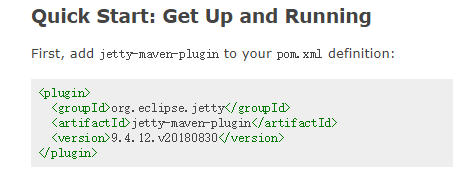
<version>9.4.12.RC2</version>

</plugin>

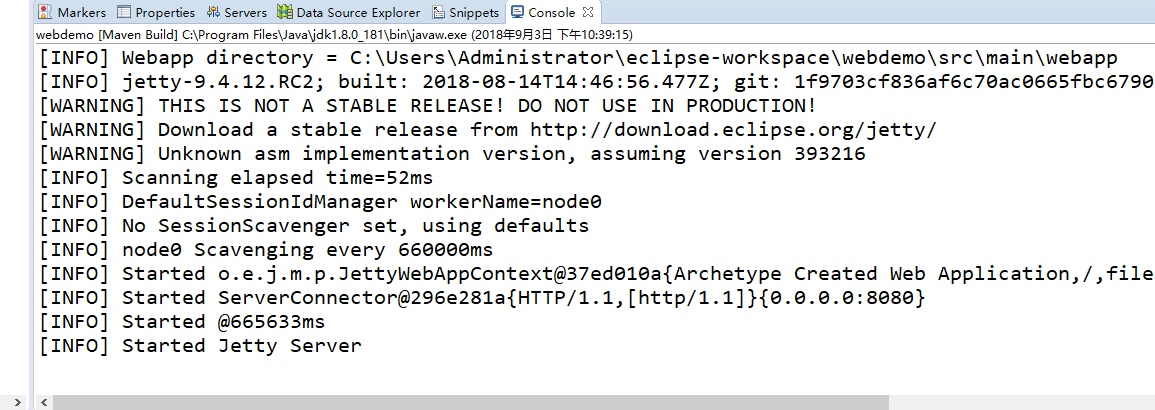
</plugins>

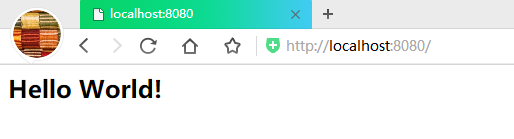
</build>

Eclipse官网也有jetty插件的写法：



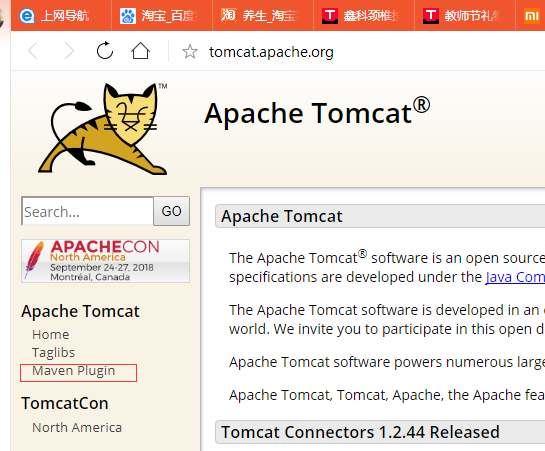
然后在maven命令里面输入 jetty:run 即可运行。第一次会下载很多东西。

结果如下：



1. 如果要使用Tomcat作为容器呢，需要增加tomcat的插件

进入tomcat的官网，找到maven plugin



先将jetty的插件注释掉

<plugin>

<groupId>org.apache.tomcat.maven</groupId>

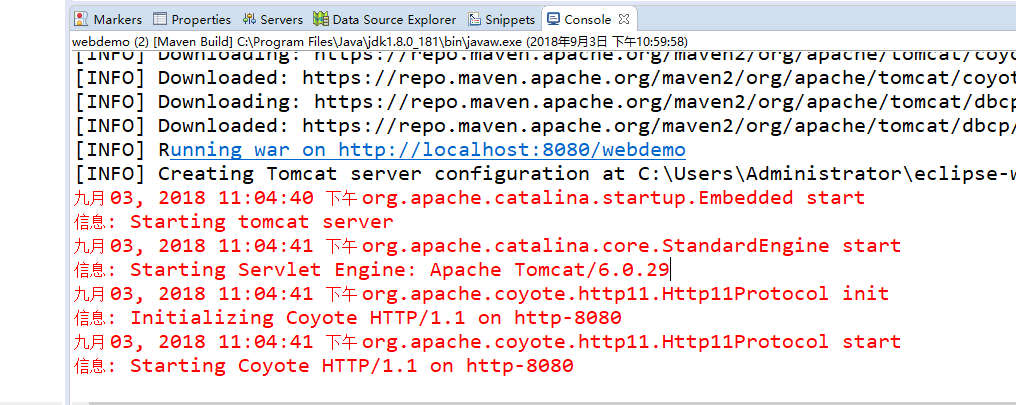
<artifactId>tomcat7-maven-plugin</artifactId>

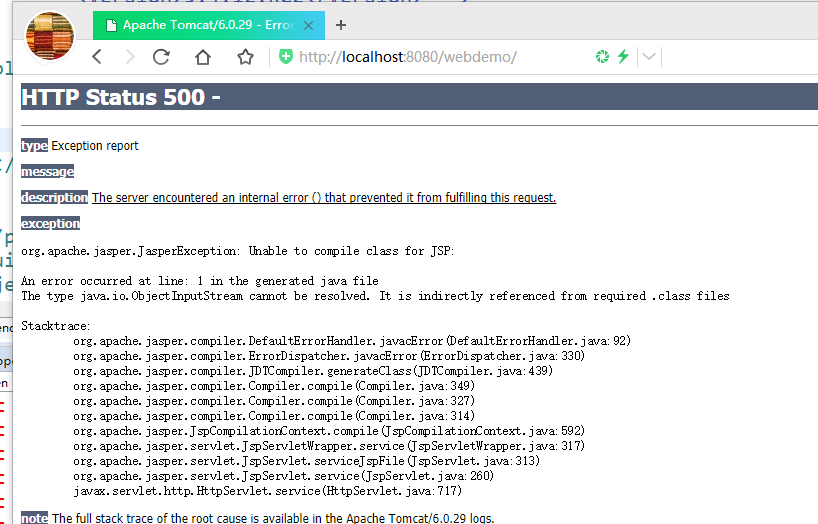
<version>2.2</version>

</plugin>

输入maven命令：tomcat：run 即可运行。第一次也会下载很多东西

运行结果

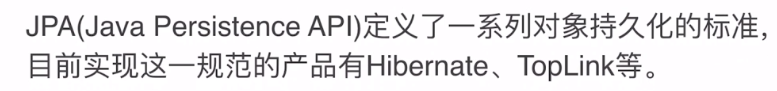


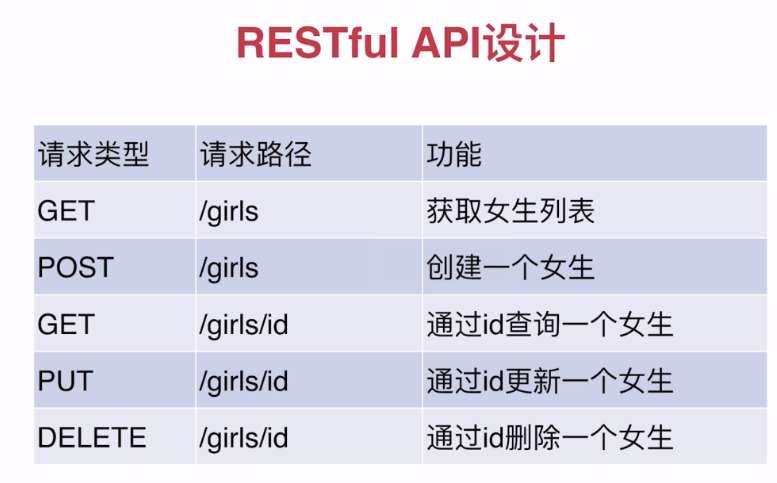


输出结果还是有问题

2018年9月4日09:20:17

1. spring-data-jpa





导入依赖：

~~<dependency>~~

~~<groupId>org.springframework.data</groupId>~~

~~<artifactId>spring-data-jpa</artifactId>~~

~~<version>2.0.9.RELEASE</version>~~

~~</dependency>~~

上面是错的，应该是 starter，否则一些注解无效

<**dependency**>  
 <**groupId**>org.springframework.boot</**groupId**>  
 <**artifactId**>spring-boot-starter-data-jpa</**artifactId**>  
 <**version**>2.0.4.RELEASE</**version**>  
</**dependency**>

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.12</version>

</dependency>

1. 1 int与Integer的基本使用对比

（1）Integer是int的包装类；int是基本数据类型；

（2）Integer变量必须实例化后才能使用；int变量不需要；

（3）Integer实际是对象的引用，指向此new的Integer对象；int是直接存储数据值 ；

（4）Integer的默认值是null；int的默认值是0。

什么时候用Integer :

如果该属性所对应的数据库的字段是主键或者是外键时，用Integer；因为Integer的默认值为null，数据库的主键或者外键不能为空，但是可以为null

什么时候用int :

如果表示数量用int，因为int的默认值为0，它不能为null或者空

1. 依赖注入（Dependency Injection）和控制反转（Inversion of Control，IOC）含义是完全相同的，都是变主动为被动。

Spring 将被依赖对象注入给调用者，所以调用者无须主动获取被依赖对象，只要被动等待spring容器注入即可。

程序完全无需理会被依赖对象的实现，也无需主动定位工厂，这是一种优秀的解耦方式。实例之间的依赖关系由IoC容器负责管理。

Spring框架之后的两个主要改变是：

1. 程序无需使用new调用构造器去创建对象。所有的java对象都可以交给spring容器去创建。
2. 当调用者需要调用被依赖对象的方法时，调用者无需主动获取被依赖对象，只要等待spring容器注入即可。

依赖注入通常有两种：

1. 设值注入：IoC容器使用成员变量的setter方法来注入被依赖对象。
2. 构造注入：IoC容器使用构造器来注入被依赖的方法。
3. Java Persistence API定义了一种定义，可以将常规的普通Java对象（有时被称作POJO）映射到数据库。  
   这些普通Java对象被称作Entity Bean。

2018年9月4日15:21:05

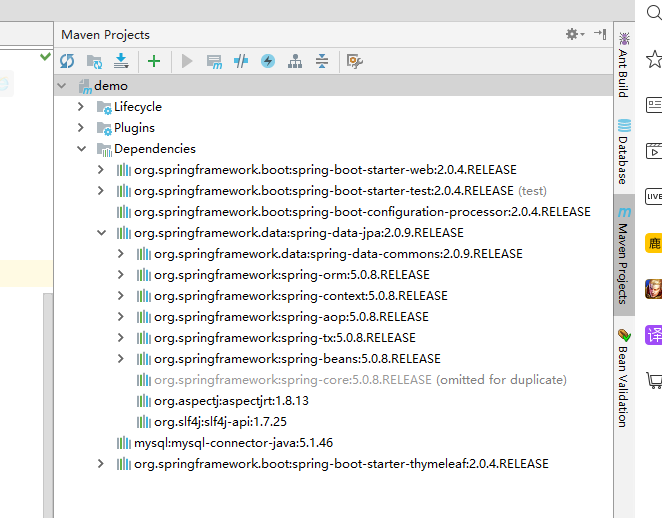
1. ORM 对象关系映射 object Relation Mapping

完成面向对象的编程语言到关系数据库的映射。

自己理解就是处理数据库的一种规范。

当ORM框架完成映射后，既可以利用面向对象程序设计语言的简单易用性，又可以利用关系数据库的技术优势。 因此可以把ORM框架当成应用程序和数据库的桥梁。

1. JAVA EE 5 规范抛弃了传统的Entity EJB，改为使用JPA作为持久层解决方案。
2. 在idea的右边可以查看maven的依赖



原来idea可以同时生成get和set方法，选择要生成的变量，可以多个，然后右键，选择generate或者alt+insert，然后选择getter and setter，

1. 消息中间件

中间件   
非底层操作系统软件，非业务应用软件，不是直接给最终用户使用的，不能直接给客户带来价值的软件统称为中间件。   
消息中间件   
管制关注于数据的发送和接收，利用高效可靠的异步消息传递机制集成分布式系统。

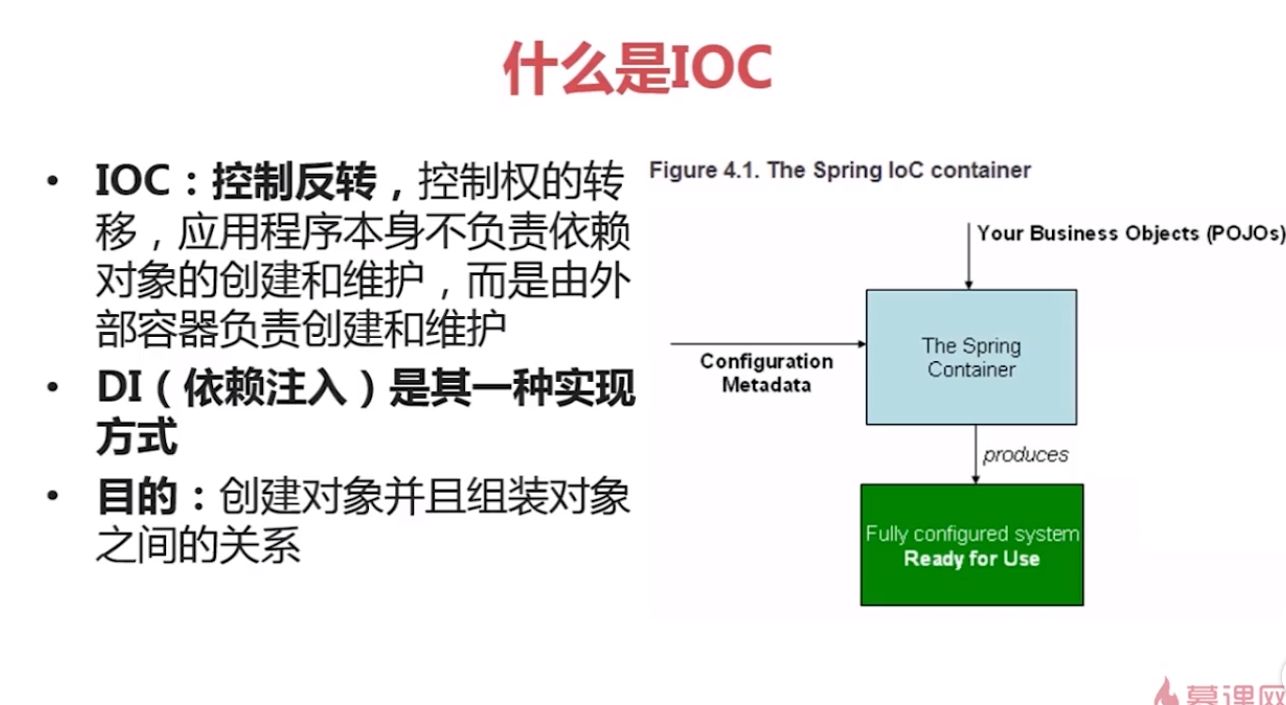
常见消息中间件   
activeMQ（支持多语言，实现jms1.1，j2ee1.4规范），RabbitMQ（支持更多语言，基于AMQP规范），kafka(高吞吐量，分布式，分区,O(1)磁盘顺序提供消息持久化

1. Spring为企业应用提供了一个轻量级的解决方案。包括：基于依赖注入的核心机制、基于AOP的声明式事物管理、与多种持久层技术整合，以及优秀的web MVC框架等。

Spring 贯穿表现层、业务层、持久层。

Spring包容器中的所有对象都当成bean。和java bean不一样。

1. Spring 是一个轻量级的控制反转IoC和面向切面AOP的容器框架



获得依赖对象的过程被反转了。取名为：依赖注入

2018年9月5日09:46:50

1. 下载spring framework

进入网站<http://repo.springsource.org/libs-release-local/>

是一个目录结构，沿着路径org->springframework->spring 看到各个版本的spring框架，下载dist的压缩包。

1. 运行JAVA EE书上第七章的第一个spring 例子。书上只有源码，没有依赖的jar包，需要自己下载。

用eclipse打开7.2节的myspring项目，这只是一个简单的java application，也就是控制台程序。按照前面的方法下载spring framework jar包，解压到本地。

Eclipse打开项目的build path，选择libraries，然后点击add external jars，加载框架下libs文件夹里面的所有jar文件即可。这样项目就不会报错了。最后右键项目，选择run as java application，这样控制台就能输出信息了。

1. Spring 的bean中可能有property元素。Spring只要看到property元素，框架就会在底层以反射方式执行一次setter方法。也就是说这个类里面有个set方法。

<bean id="person" class="org.crazyit.app.service.Person">

<!-- 控制调用setAxe()方法，将容器中axe Bean作为传入参数 -->

<property name="axe" ref="axe"/>

</bean>

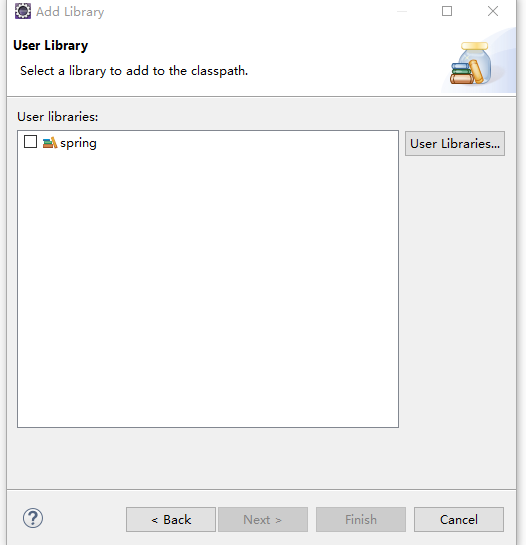
1. 在eclipse中使用user library添加spring或者hibernate库的另一种方法，

一种是直接在build path里面添加jar包

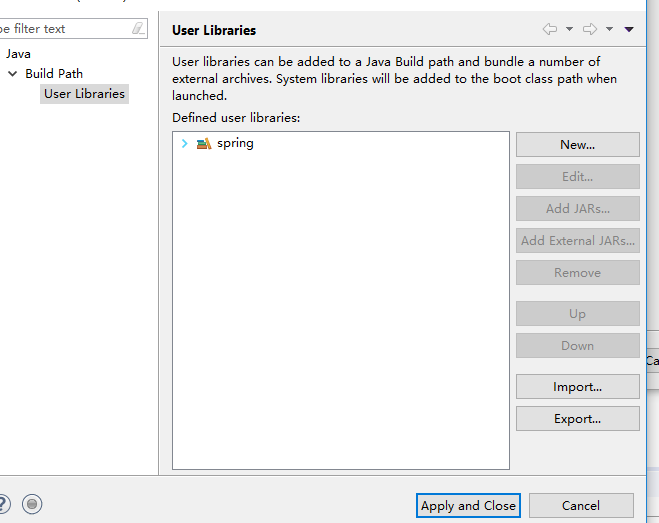
另一种是添加user library

右击项目，选择build path，然后选择add libraries，然后选择user library，

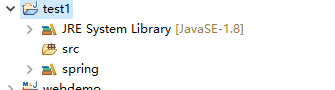
然后会出现以来已经安装的库的列表，如果是空白的点击右边的user libraries。



在新出来的对话框点击new，输入新的库的名字，然后点击add external jars，导入所需的jar包即可。



选好库后，项目结构如下：



这个spring就是之前的user library名字，是可以自定义的。

2018年10月8日14:31:33

1. 反向代理（Reverse Proxy）方式是指接收Internet上的连接请求，然后将请求转发给内部网络上的服务器，并将从服务器上得到的结果返回给Internet上请求连接的客户端，此时代理服务器对外就表现为一个反向代理服务器。

正向代理服务器：局域网内通过一个正向代理服务器访问外网。

一般使用Nginx作为反向代理服务器。

1. 会话管理

单个应用的会话由tomcat来管理，会话信息与tomcat存放在一起。如果部署多个应用，对于同一个用户请求，即使请求通过Nginx派发到不同的服务器上，也能共享会话信息。

1. 复制会话：web服务器通常都支持session复制，一台应用的会话信息改变将立刻复制到其他集群的web服务器上。
2. 集中式会话：所有的web服务器都共享一个会话，会话信息通常存放在一台服务器上，比如使用Redis服务器来存放会话。

2018年10月18日19:19:42

1. 使用@Controller必须要配合一个模板来使用

（1）在maven中加入依赖

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

1. 在resources目录下新建一个目录templates

然后在templates下新建一个html文件，比如“index.html”

这样在controller的方法里面就可以返回值这个html页面了

return “index”；

1. 视频教程上说现在不推荐使用模板了。因为现在的java web开发都是前后端分离的，后端提供一些rest接口即可。使用模板会降低系统性能。
2. 一个controller配置多个url



请求方法，一般是get和post，如果requestmapping，里面的method什么都不写，就任何方式都能使用。

1. url里面的参数

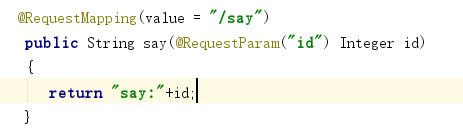


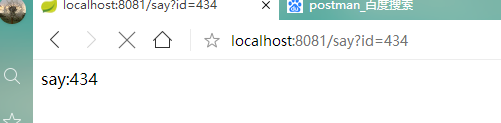


注意要写成RestController

上面的方式是直接在原来的 url上用斜线拼接字符串。使用的是@PathVariable注解

如果是使用传统的问号方式的提交参数的话，要用到@RequesParam注解





2018年10月18日20:11:21

1. 
2. 使用数据库，需要先引用spring boot data jpa

<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->

<dependency>

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

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

<version>2.0.6.RELEASE</version>

</dependency>

然后添加mysql

<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->

<dependency>

<groupId>mysql</groupId>

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

<version>8.0.12</version>

</dependency>

然后在yml里面配置数据库的地址

1. 看视频教程里面的springboot的jpa教程

出现异常

org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'entityManagerFactory' defined in class path resource [org/springframew

找了好久，才发现是jar包导入错误了。

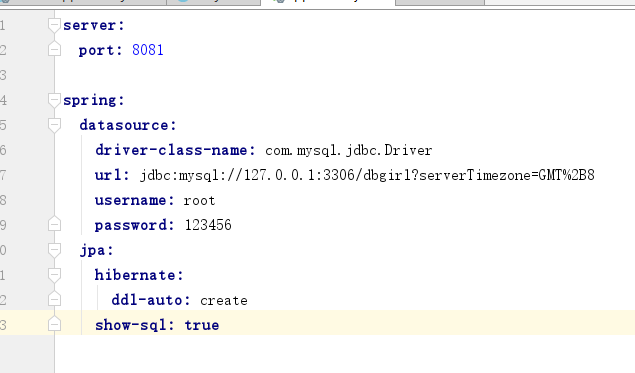
导入包错误出现在@ID上，这个注解是必须的。表示主键。如果没有做这个注解，则会出现上面的错误。

然后这个注解需要导入包是：**import** javax.persistence.Id;

1. 实现目的，自动在mysql里面创建一个表girl

使用jpa，就不需要自己写sql语句来创建表了。直接创建一个实体类即可。系统会自动操作数据库。

配置如下：



ddl-auto: create表示是创建一个表，也可以是updata等

然后新建一个Girl类，类前需要添加@Entity注解

Girl就表示一个实体，最后会自动在mysql中创建一个gril表，并且表格式和这个girl类的结构一样。

package com.example.demo6;

**import javax.persistence.Entity;**

**import javax.persistence.GeneratedValue;**

**import javax.persistence.Id;**

@Entity

public class Girl {

@Id

@GeneratedValue

private Integer id;

private String cupSize;

private Integer age;

public Girl()

{

}

public Integer getId() {

return id;

}

public String getCupSize() {

return cupSize;

}

public Integer getAge() {

return age;

}

public void setId(Integer id) {

this.id = id;

}

public void setCupSize(String cupSize) {

this.cupSize = cupSize;

}

public void setAge(Integer age) {

this.age = age;

}

}

有两点需要注意的：

1. 需要添加一个Girl的无参构造方法，并且需要实现各个变量的getter和setter方法。
2. 添加@ID注解，导入的包名，不能是springframework的包，应该是import javax.persistence.Id;

很容易就导入错误。

1. 查询数据库

查询数据库非常简单，不需要自己写一行sql语句

为了演示效果，先在girl表中手动增加几行数据。

然后yml的配置里面要修改成：**ddl-auto:** update

如果是create，那么系统一启动就会先删除girl表。

接下来创建一个GirlController类，和GirlRepository接口

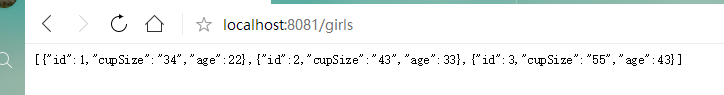
**package** com.example.demo6;  
  
**import** org.springframework.data.jpa.repository.JpaRepository;  
  
**public interface** GirlRepository **extends** JpaRepository<Girl,Integer> {  
  
}

这个接口，不需要自己去实现，spring会自动实现的。

**package** com.example.demo6;  
  
**import** org.springframework.beans.factory.annotation.Autowired;  
**import** org.springframework.web.bind.annotation.GetMapping;  
**import** org.springframework.web.bind.annotation.RestController;  
  
**import** java.util.List;  
@RestController  
**public class** GirlController {  
  
 @Autowired  
 **private** GirlRepository **girlRepository**;  
  
 @GetMapping(**"/girls"**)  
 **public** List<Girl> girlList(){  
  
 **return girlRepository**.**findAll**();  
 }  
  
}

这样就实现了查询girl表中所有数据的功能。

浏览器返回：



2018年10月19日10:20:27

1. @RestController是@Controller和@Responssbody的组合。

@Responsbody注解表面，返回的内容是文字，而不是html界面。

1. Springboot要是实现判断用户是否登录，如果没有登录就重定向到登录界面。要用到的技术是springboot的拦截器功能。
2. 日志打印功能。简单的可以使用system.out.println()来在控制台输出。

但是推荐使用logger来输出日志。

在要使用日志的类下定义一个变量。

private static final Logger log = LoggerFactory.getLogger(SessionHandlerInterceptor.class);

括号里面是当前类名。

注意导入的包名是slf4j的包。其他的不行。

**import** org.slf4j.Logger;  
**import** org.slf4j.LoggerFactory;

然后就可以使用log.info()等输出日志来了。

1. 记录下使用拦截器实现用户登录验证的具体实现过程。也就是当用户请求某些方法的时候，如果没有登录，则重定向到登录页面。

实现过程比较简单，记得以前使用play framework也有过类似的功能，只需要在方法前添加一个注解即可。

在springboot中判断用户是否登录主要用到tomcat的session。Session是保存用户信息的工具。一个浏览器只保存一个session对象，不管这个浏览器打开了多少个页面。

这里使用HttpSession，获取它的对象的方法如下：

HttpSession session = request.getSession();

然后就可以使用setAttribute（）或者getAttribute（）方法来设置或者获取数据。实际上就和键值对一样。

判断用户登录情况就用，增加一个userName键就可以了。

1. 要实现拦截器功能，要实现两个自定义类。

第一个是要实现WebMvcConfiguration接口，或者实现WebMvcConfigurationSupport类。

WebMvcConfigurationSupport类是在springboot2.0里面使用的。如果是之前的版本要用WebMvcConfigurerAdapter。

然后主要重写addInterceptors方法。在里面添加拦截路径。

@Configuration  
**public class** WebAppConfig **extends** WebMvcConfigurationSupport {  
 @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 registry.addInterceptor(**new SessionHandlerInterceptor**()).addPathPatterns(**"/\*"**).excludePathPatterns(**"/login"**);  
 }  
  
 @Override  
 **public void** addCorsMappings(CorsRegistry registry) {  
  
 }  
  
 @Override  
 **public void** addFormatters(FormatterRegistry registry) {  
  
 }  
  
 @Override  
 **public void** addViewControllers(ViewControllerRegistry registry) {  
  
 }  
}

需要特别注意的是类名前需要有个@Configuration注解。否则拦截无效。

接下来要实现真正的拦截类。实现接口HandlerInterceptor即可。这个接口的实现类传递到前面的类的addInterceptors方法即可。

**public class** SessionHandlerInterceptor **implements** HandlerInterceptor {  
  
 **private static final** Logger ***log*** = LoggerFactory.*getLogger*(SessionHandlerInterceptor.**class**);  
  
  
 @Override  
 **public boolean** preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) {  
 ***log***.info(**"---------------------开始进入请求地址拦截----------------------------"**);  
 HttpSession session = request.getSession();  
 **if**(!StringUtils.*isEmpty*(session.getAttribute(**"userName"**))){  
 **return true**;  
 }  
 **else**{  
*// PrintWriter printWriter = null;  
// try {  
// printWriter = response.getWriter();  
// } catch (IOException e) {  
// e.printStackTrace();  
// }  
// printWriter.write("{code:0,message:\"session is invalid,please login again!\"}");* **try** {  
 response.sendRedirect(**"/login"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return false**;  
 }  
  
 }  
  
 @Override  
 **public void** postHandle(HttpServletRequest request, HttpServletResponse response, Object handler, ModelAndView modelAndView) {  
 ***log***.info(**"--------------处理请求完成后视图渲染之前的处理操作---------------"**);  
 }  
  
 @Override  
 **public void** afterCompletion(HttpServletRequest request, HttpServletResponse response, Object handler, Exception ex) {  
 ***log***.info(**"---------------视图渲染之后的操作-------------------------0"**);  
 }  
}

在preHandler方法里面就可以判断用户是否登录了。如果登录了，返回true，才能继续向下执行。

2018年10月19日15:58:16

1. 新的问题。按照前面的配置的拦截器。拦截器的功能是实现了。但是静态资源也给拦截了。也就是static下的样式文件，js文件等都不能访问了。

@Configuration  
**public class** WebAppConfig **extends** WebMvcConfigurationSupport {

解决方法有很多。其中一种是：

前面继承了WebMvcConfigurationSupport

不继承它，而是修改成实现WebMvcConfigurer接口就可以了。

如下：

@Configuration  
**public class** WebAppConfig **implements** WebMvcConfigurer {  
 @Override  
 **public void** addInterceptors(InterceptorRegistry registry) {  
 registry.addInterceptor(**new** SessionHandlerInterceptor())  
 . addPathPatterns (**"/\*"**)  
 .excludePathPatterns(**"/login"**);  
 }  
  
}

注意. addPathPatterns (**"/\*"**)方法后面是一个星号。有的教程上是两个。暂时还不知道一个星号和两个星号的区别。

2018年10月19日23:13:02

1. 真的是要崩溃了，写了一下午的表单提交，都没有成功。显示写了个最原始，最简单的，就成功了。也不知道问题出在哪里。

我的登录界面为：

<html>

<head>

<title>表单提交</title>

</head>

<body>

<form action="/login" method="post">

<input type="text" name="userName"/> <br/>

<input type="text" name="passWord"/> <br/>

<input type="submit" value="提交"/>

</form>

</body>

</html>

里面有个很简单的form表单。

Controller为：

*//提交登录信息*@PostMapping(value = **"/login"**)  
**public** @ResponseBody String loginSub(@ModelAttribute Administrator administrator)  
{  
 ***log***.info(**"开始"**);  
 **if**(administrator != **null**)  
 {  
 ***log***.info(administrator.getUserName());  
 ***log***.info(administrator.getPassWord());  
 }  
  
 **return** administrator.toString();  
}

一开始用的别人写的很好看的前端模板。是基于jQuery的。然后按钮实际上是个div标签。使用jquery的$来写点击事件，总是不成功。后来发现没有引用jquery的js文件。

然后在点击事件里面用alert输出信息。终于能输出了。可是表单提交一点效果都没有。使用postman工具测试，表明后台结束表单数据是的没问题的。

真是好难啊。

然后就写了上面的最简单的form提交界面。一开始也不行。后来把拦截器关了，url换了。各种尝试终于成功了。问题也不出在拦截器，url也没问题。真的很奇怪。

接着换回原来好看的界面。提交还是不行。那只好把提交的按钮从div换成input了。

*<!--<div class="login-btn" type="submit">马上登录</div>-->*<**input type="submit" class="login-btn" value="马上登录"**/>

2018年10月20日21:43:19

1. 使用JPA查询一条数据。

视频教程上使用findOne（）来查询一条数据，参是主键。可是这个方法已经不好用了。在springboot 一点几版本可以，二点几的版本，用法已经变了。

可以使用findById（）来查询，参数是主键的数据类型。本身是泛型，根据主键来决定真实类型。

可是问题来了，findById返回的并不是实体类型。而是Optional<T>,

Optional 是一个包含或着不包含一个非空值的容器对象。如果一个值存在，isPresent()将会返回true 并且 get() 将会返回所对应的值.

也就是返回optional<T>后，用ispresent来判断值是否存在，如果存在用get方法获取值，这时得到的才是真实实体数据。

*//提交登录信息*@PostMapping(value = **"/login"**)  
**public** @ResponseBody String loginSub(@ModelAttribute Administrator administrator, HttpServletRequest request, HttpServletResponse response) {  
 HttpSession session = request.getSession();  
 Administrator result = **null**;  
  
 **log**.info(**"开始:"**+ administrator.getUserName()+**" "**+administrator.getPassWord());  
 **if** (administrator != **null**) {  
 Optional<Administrator> queryresult = **administratorRepository**.findById(administrator.getUserName());  
 **if** (queryresult.isPresent()) {  
 result = queryresult.get();  
 }  
 }  
 **if** (result != **null** && result.getPassWord().equals(administrator.getPassWord())) {  
 *//用户名和密码正确，在session中增加内容，然后转到主页* session.setAttribute(**"userName"**,result.getUserName());  
 **return** result.toString();  
  
 } **else** { *//用户名或者密码错误，重定向到登录页面* **try** {  
 response.sendRedirect(**"/login"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return null** ;  
 }  
  
}

2018年10月21日17:03:52

1. 记录下使用EntityManager执行原生SQL语句的过程。虽然原理几乎看不懂，网上讲解的都是乱七八糟的 。但是还是实现了，做一下简单记录。
2. 定义一个接口用于返回所有值

**public interface** BaseDao<T> {  
  
 List<T> findAll();  
  
}

（2）建立一个接口的实现类。

@Repository  
**public class** BaseDaoImp **implements** BaseDao<Administrator> {  
  
 @PersistenceContext  
 **private** EntityManager **entityManager**;  
  
 @Override  
 **public** List<Administrator> findAll() {  
 **return this**.**entityManager** .createQuery(**"select t from Administrator t"**,Administrator.**class**)  
 .getResultList();  
  
 }  
}

注意：类前面要添加@Repository注解

（2）调用方法

@Autowired  
**private** BaseDaoImp **baseDaoImp**;  
  
*//测试*@GetMapping(**"/test3"**)  
**public** @ResponseBody List<Administrator> getAdmin3()  
{  
 **return this**.**baseDaoImp**.findAll();  
}

注意：使用@Autowired注解，不需要new一个对象了。

2018年10月21日21:44:20

1. 研究了一天的springboot创建数据库表的问题都没有成功，知道现在才发现用JdbcTemplate可以了。

网上的教程都真TM的扯淡。全是springboot jpa自动建表。我建你个大爷啊。我不需要自动建表，要动态建表。根本找不到相关教程。自动建表，需要创建一个Entity类。系统会自动根据这个实体类建表。也就是说表的结构是事先已经定死了的。我现需要根据我的业务来随时创建新的表。比如新来一个用户注册了，就要给他创建一个用户表。不可能每次都在一个表里面操作啊。

也就是说我需要执行原始的SQL语句。网上有个EntityManage可以执行createQuery来执行sql语句。可是我测试了，只能执行查询语句。我要执行create table就报错。真是的非常无语。搞了一天都没有搞出来。

直到现在，看到springboot 2精髓这本书上提到了Spring JDBC Template 这个东西，立马试了，终于成功了。

2018年10月22日22:37:32

1. Android和springboot使用json传递数据

这里记录两者都使用阿里巴巴的fastjson工具的方法。

Android端的使用方法见Android的笔记。

主要是 String str= JSON.toJSONString(list); 将集合转换成json字符串。然后使用okhttp的post方法提交数据。

后台springboot要使用fastjson，需要添加maven依赖

<dependency>

<groupId>com.alibaba</groupId>

<artifactId>fastjson</artifactId>

<version>1.2.49</version>

</dependency>

然后解析数据，本质就是讲请求体的string转换成转换成对象。所以请求体是@RequestBody String str

然后使用JSONArray jsonArray= JSON.parseArray(str); 来转换数据。和android上是一样的。

*/\*  
使用fastjson来解析数据  
 \*/*@RequestMapping(**"/json"**)  
**public** @ResponseBody String getJsonByFastJson(@RequestBody String str)  
{  
 System.***out***.println(str);  
  
 JSONArray jsonArray= JSON.*parseArray*(str);  
  
 StringBuilder sb= **new** StringBuilder();  
 **for**(**int** i=-0;i<jsonArray.size();i++)  
 {  
 JSONObject jsonObject= jsonArray.getJSONObject(i);  
 sb.append(jsonObject.getString(**"name"**)).append(**" "**);  
 }  
  
 **return "接收到："**+sb.toString();  
  
}

1. 刚刚导入了fastjson的依赖，可是怎么也用不了。根部没有它的包。

不管我的maven刷新了多少次都无用。最后我将idea重启了。就没问题了。

1. 后台使用fastjson来解析json。客户端依然使用fastjson。

这种方式更简单，直接将请求体转换成集合对象。

springboot的默认json工具是jackson。所以使用很简单，将请求体设置为@RequestBody List<Contact> list即可。

但是有一点要注意点注意的是android端和后台的Contact对象要一致，也就是格式要一样。否则解析出错。

@RequestMapping(**"/json2"**)  
**public** @ResponseBody String getJsonByJackson(@RequestBody List<Contact> list) *//默认使用jackson解析json，也可以使用fastjson* {  
 StringBuilder sb = **new** StringBuilder();  
  
 **for** (Contact contact : list) {  
 sb.append(contact.getName()).append(**" "**);  
 }  
  
 **return "接收到："** + sb.toString();  
  
 }

2018年10月23日11:29:54

1. 客户端和后台设置请求的ContentType

之前使用okhttp提交json的时候设置了contenttype为json。可以在后台获取到contenttype。

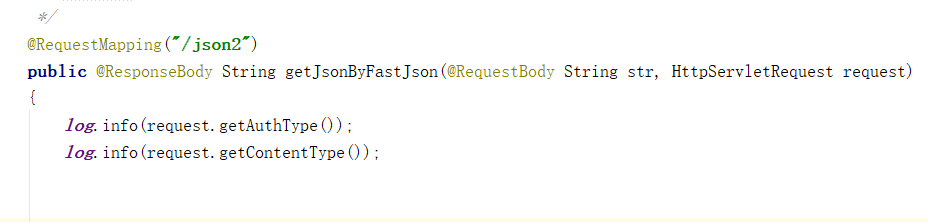
（1）android的写法是：先定义一个字符串：

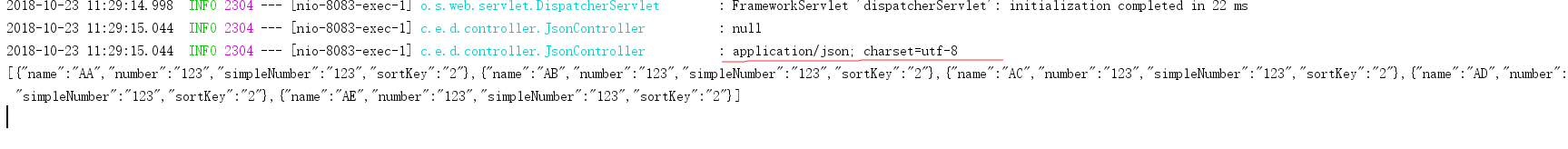
**public static final** MediaType ***JSONTYPE*** = MediaType.*parse*(**"application/json; charset=utf-8"**);

然后在RequestBody 创建时设置类型



1. 后台的写法也很简单，在controller里面实现。





可以看到后台输出的contenttype和前台设置的内容一模一样。

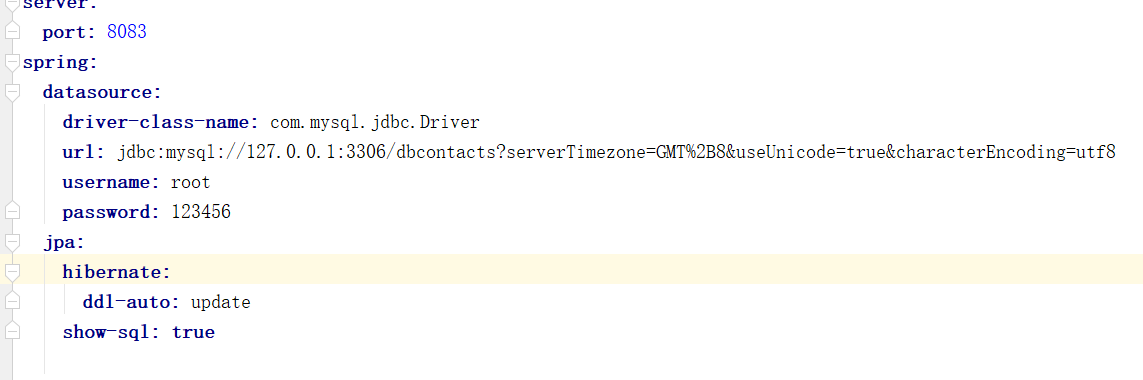
2018年10月23日23:22:37

1. springboot 操作mysql中文乱码。

首先同意字符编码为utf8

然后在springboot中mysql的连接配置里面设置编码格式为utf8

url: jdbc:mysql://127.0.0.1:3306/dbcontacts?serverTimezone=GMT%2B8&useUnicode=true&characterEncoding=utf8



2018年10月24日21:05:26

1. 使用thymeleaf给html传递数据

（1）首先要添加依赖

<dependency>

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

<artifactId>spring-boot-starter-thymeleaf</artifactId>

</dependency>

（2）controller里面给model添加属性

@GetMapping("/person.html")

public String showPersonManager(Model model)

{

String name = "hello guozhang";

model.addAttribute("name",name);

return "web/person";

}

这样前台就能获取到name的属性了。

1. 前台的写法

<p th:text="'Hello！, ' + ${name} + '!'">3333</p>

这样p标签显示的就是Hello！, hello guozhang

2018年10月24日22:08:04

1. 使用列表标签和thymeleaf来接收前台传递的集合

前台:

<**ul**>  
 <**li th:each="mylist: ${list}" th:text="${mylist}"**></**li**>  
 </**ul**>

后台:

@GetMapping(**"/person.html"**)  
 **public** String showPersonManager(Model model, ModelMap map)  
 {List<String> list=**new** ArrayList<String>();  
 list.add(**"1s"**);  
 list.add(**"2s"**);  
 list.add(**"3s"**);model.addAttribute(**"list"**,list);  
  
 **return "web/person"**;  
 }

总结：只需要将一个list添加到model里面。

前台使用each来迭代。

<li th:each="mylist: ${list}" th:text="${mylist}"></li>

其中mylist是list集合中一个元素。名字自定义。list是和后台model.addAttribute(**"list"**,list);里面的**"list"**保持一致。

2018年10月25日09:28:10

1. css中的类选择器是点。而id选择器是#



2018年10月25日16:04:14

1. 使用jquery的方式，让html在加载时执行JavaScript代码。

首先引入jquery的js库。

<**script type="text/javascript" src="js/jquery.js"**></**script**>

然后编写要执行的方法：

<**script type="text/javascript"**>  
 **$**(***document***).ready(**function** () {  
 *alert*(**"hello world"**);  
 });  
</**script**>

可以在这里执行一些ajax代码，来加载页面初始化数据。这样就能达到前后端分离的目的。也不用springboot的那些模板了。

1. ul列表的点击事件，使用jquery实现

<**script type="text/javascript"**>  
 **$**(***document***).ready(**function** () {  
 **$**(**'ul'**).on(**'click'**,**'li'**,**function** () {  
 *alert*(**$**(**this**).**text**());  
 })  
 });  
</**script**>

注意**$**(**'ul'**).on(**'click'**,**'li'**,**function** () 里面是单引号

1. 设置ul列表里面鼠标的样式。

默认情况下，鼠标在列表上面会呈现选择文字样式。可以改成只有一个箭头的样式，设置cursor为default即可。

.**listcontainer li**:**hover**{  
 **background-color**: **#bdbbbb**;  
 **cursor**:**default**;  
 */\*color: #333;\*/*}

1. 用jquery的方式来执行按钮的点击事件。

比如在html里面有个按钮，并设置其id为testAjax

<button id=”testAjax” type=”button”>按钮</button>

js代码为：

<**script type="text/javascript"**>  
 **$**(**"testAjax"**).click(**function** () {  
 *alert*(**"xxx"**);  
 });  
</**script**>

1. 实现ajax请求。

比如在按钮的点击事件里面，页面加载事件里面实现。

使用$.ajax()的方式来实现。比如：

**$**.**ajax**({  
 **type**:**"POST"**,  
 **url**:**"person"**,  
 **data**:**$**(**this**).**text**(),  
 success:**function** (data) {  
 *alert*(data);  
 }  
});

其中data请求体的内容，在springboot里面用@RequestBody来获取。

也可以加上error，比如我把url故意写错。

1. 在ajax的url里面传递参数。

上面的写法是把参数写到data里面，即请求体里面了。实测的时候发现在原来的数据后面多了个=好，虽然影响不大，但是这样写法不完美。正确的应该将参数匹配到url里面。

**url**:**"person/"**+**$**(**this**).**text**(),

然后后台用PathVariable来获取这个参数，这样，参数就没有等号了。



1. post请求的一样

"Error resolving template [fff], template might not exist or might not be accessible by any of the configured Tem

当出现上面的异常时，实际上是controlle的方法的返回类型没有加@ResponseBody

这个注解表面返回的类型是字符串，而且方法本身的返回类型不一定是String，可以是一个类。而且是用jackson将对象转换成json的。

@PostMapping(**"/users"**)  
**public** @ResponseBody List<UserInfo> getAllusers()  
{  
 List<UserInfo> alluser = **userInfoRepository**.findAll();  
 **return** alluser;  
}

1. 在jquery里面遍历json数组。使用$each()方法。

一开始按照网上有篇博客上的教程写的，可是一直失败，读取不到json里面的每个对象。然后我不用springboot，直接写了简单的html页面来解析json。就没问题，同样的代码，放到springboot里面就不行。

<html>

<head>

<script type="text/javascript" src="jquery.js"></script>

</head>

<body>

<script type="text/javascript">

$(document).ready(function () {

var json3 = [{"name":"bbb","serialNumber":"22222fff2"},{"name":"aaa","serialNumber":"1111"},{"name":"mmm","serialNumber":"866952033252797"}];

$.each(json3, function(idx, obj) {

console.log(idx+'...'+obj.serialNumber);

});

});

</script>

</body>

</html>

json数据内容都是一样的。

后来发现另一篇网页上写了用eval转换string为json对象。这样一下就成功了。

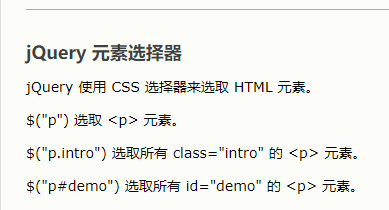
success:**function** (data) {  
 **$**(**"#msg"**).html(data);**var jsonObj = *eval*('(' + data + ')');** ***console***.log(jsonObj);  
 **$**.each(jsonObj,**function**(idx,obj){  
 ***console***.log(idx+**'....'**+obj.serialNumber+**'...'**+obj.**name**);  
 });  
 },

注意eval的写法，比较复杂。

2018年10月25日18:39:50

1. jquery的选择器

jquery的选择器和css的选择器一样，有三种：标签选择器、类选择器、id选择器



也就是说用class的要在前面加点。用id的要在签名加井号。上面表的p不一定要有。

1. jquery给列表动态添加元素

<**ul class="list"**></**ul**>

也是在ajax的返回里面修改ul列表内容。

success:**function** (data) {**$**(**".list"**).remove();  
 **var** jsonObj = *eval*(**'('** + data + **')'**);  
 ***console***.log(jsonObj);  
 **$**.each(jsonObj,**function**(idx,obj){**$**(**".list"**).append(**'<li>'**+obj.serialNumber+**'</li>'**);  
 });  
 },

注意选择器签名的用法：**$**(**".list"**)

对应id的选择器，例子：

<**p id="msg"**>xxxx</**p**>

**$**(**"#msg"**).html(data);

1. 给ul列表设置滚动条

只需要修改ul列表父容器div的overflow属性即可。设置为auto表示自动，即如果宽度显示不够，宽度增加滚动，如果高度不够，高度增加滚动。

如果设置为scroll，则一直显示滚动条。

1. 通过代码给客户端返回错误。

之前写的代码在controller里面除了出现异常，否则都是返回成功。有的时候需要手动返回错误到前台，并且设置错误信息。比如查询一个数据库，如果第一个表查询不到，就没必要进行后面的操作了。这时候就可以返回错误了。如果返回一个string res=“错误”，这样本质上还是返回的成功。要让客户端的接收到正确的错误码，比如404,500等。可以用HttpServletResponse 来实现。

@PostMapping(**"person/{serial}"**)  
**public** @ResponseBody String getPerson(@PathVariable(**"serial"**)String serial, HttpServletResponse response)  
{  
 System.***out***.println(serial);  
  
 String res= **this**.**contactDao**.getContactBySerialNumber(serial);  
 **if**(res.equals(**"error"**))  
 {  
 response.setStatus(500);  
 **try** {  
 response.getWriter().append(**"server 错误"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return "error:"**+serial;  
 }  
  
 **return** res;  
}

这样在即使没有出现异常的情况下，客户单也能接收到错误了。

1. 用jquery给table添加数据的时候，防止重复添加的方法：

**$**(**".tablebody"**).html(**""**);

**其中tablebody的tbody的class。之前使用过remove方法，但是这样也无法添加数据了。表一直是空的。**

2018年10月26日12:03:28

1. 刚刚写了个退出登录的功能。前台是在人家模板上修改的。只需要写个后台controller，在里面移除username的session。并重定向到登录页面即可。

*//从主页面退出到登录页面*@GetMapping(**"/logout"**)  
**public void** redictLogout(HttpServletRequest request, HttpServletResponse response)  
{

System.***out***.println(**"退出"**);  
  
 HttpSession session = request.getSession();  
 session.removeAttribute(**"userName"**);  
  
 **try** {  
 response.sendRedirect(**"/login"**);  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
  
}

后台很简单，关键是前台出现大坑了。前台是用jquery实现的，在点击事件里面弹出对话框，点击确认按钮，退出。

**$**(***document***).on(**'click'**, **'.pf-logout'**, **function**() {  
 **layer**.confirm(**'您确定要退出吗？'**, { *//这部分的代码修改后要清理浏览器缓存，重启浏览器才能生效* **icon**: 4,  
 **title**: **'确定退出'** *//按钮*}, **function**(){  
 ***location***.href= **'logout'**;  
});  
 });

问题出在location.href

上面的是我修改过的，原来的是location.href=’login.html’, 改成logout后，在加上后台controller也写好了。可是怎么也成功不了。点击退出后，网页链接总是localhost:8083/login.html,

无论我的idea和浏览器重启多少次都在这样。难道我改错地方了，然后我吧“**您确定要退出吗”** 这几个字修改了，返现文字确实变了，那么我改的地方肯定正确啊。这是为什么呢？

后来我把浏览器缓存清理了一下。在重启浏览器，一切正常。真的是大坑。

”

2018年10月26日12:13:1

到此时，这个通讯录的系统基本上就已经写完了。除了通讯录功能外，还增加了APP下载的功能。

然后还想增加一些功能，

1. 毕竟通讯录是个人隐私，不能让别人随便看到。应该增加管理员权限。权限不够的用户看不到完整的电话号码。
2. 前端是用的淘宝买的模板，布局只能适应电脑浏览器，手机端就比较难看了。最好能做成两者通用。