# 一：SpringBoot入门

## 1.1:SpringBoot简介

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。

## 1.2:SpringBoot特性

1. SpringBoot并不是对Spring功能上的增强，而是提供了一种快速创建独立的Spring应用程序的框架

2. 嵌入的Tomcat，无需部署WAR文件

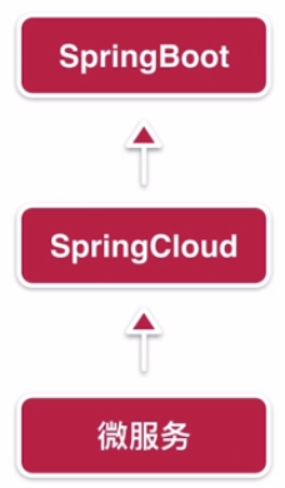
3. 简化Maven配置

4. 自动配置Spring

5. 绝对没有代码生成和对XML没有要求配置

6.备受关注，是下一代框架，已经是不争的事实，不需要学习springmvc

7.微服务的入门级微框架，springboot是springcloud的基础



## 1.3:SpringBoot开发环境准备

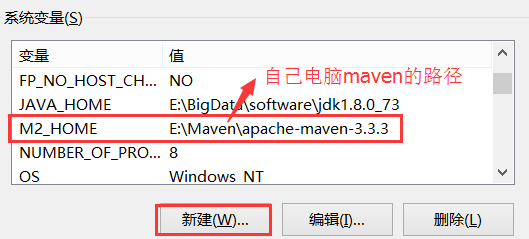
1.开发环境JDK1.8 Tomcat7.0(这里不演示配置)

2.开发工具Eclipse或者是Idea

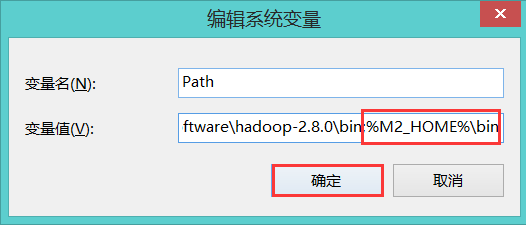
3.项目管理工具Maven

### ）电脑中配置maven环境

1. 下载maven的地址 <http://maven.apache.org/download.cgi>
2. 配置环境变量
3. 右键我的电脑==》高级系统设置==》环境变量==》新建环境变量



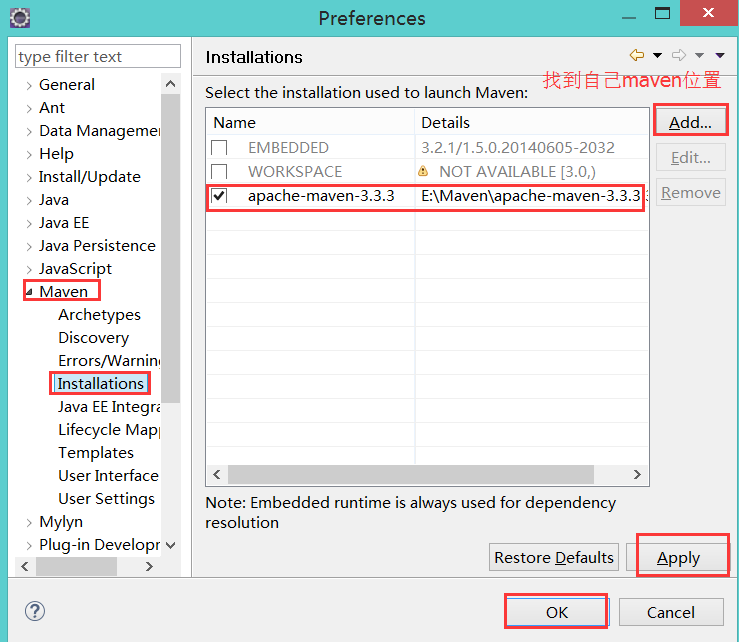
1. 在系统变量中的path中新增maven的bin目录

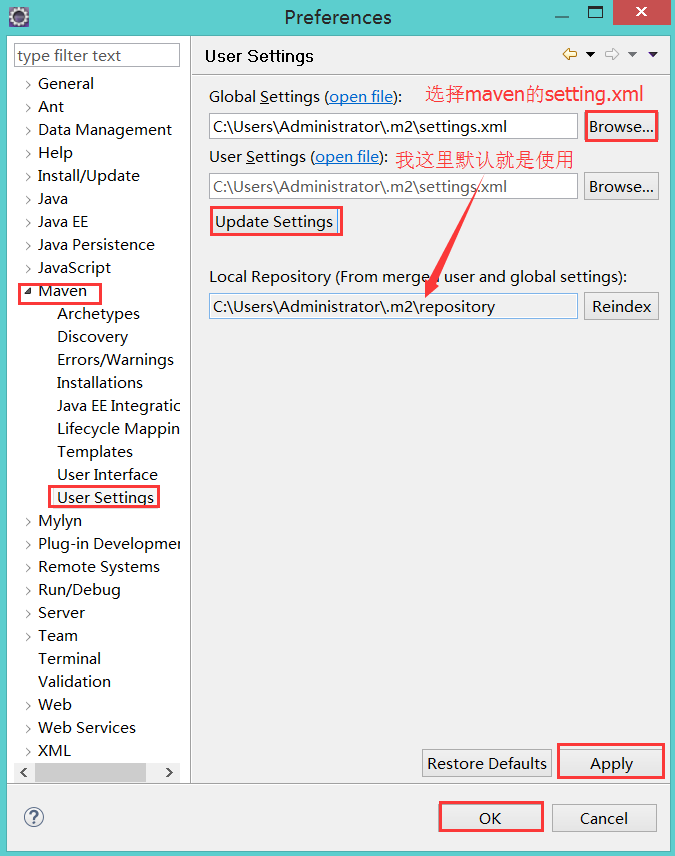


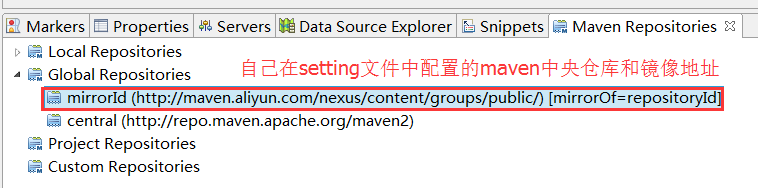
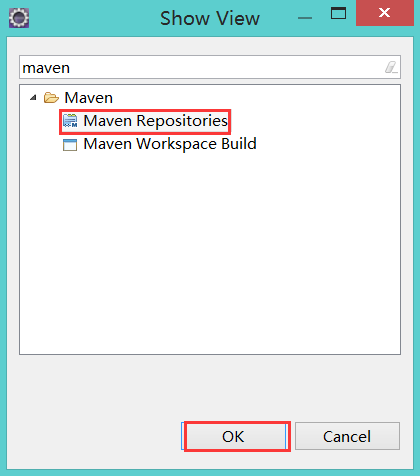
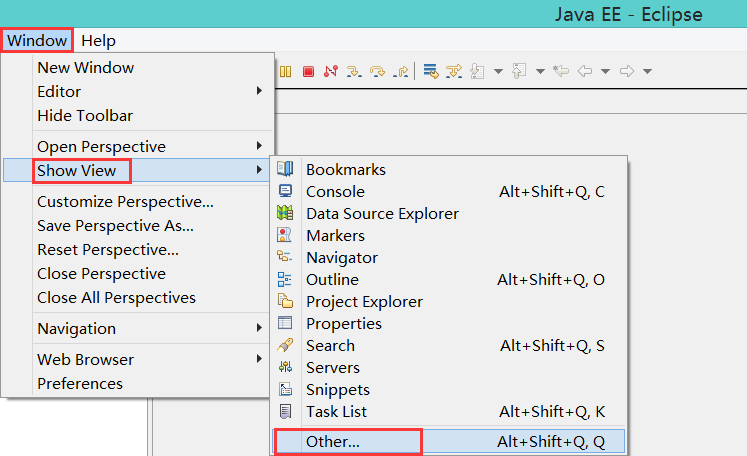
1. cmd命令运行 mvn -v 测试maven是否安装成功



### （2.）Eclipse中配置maven

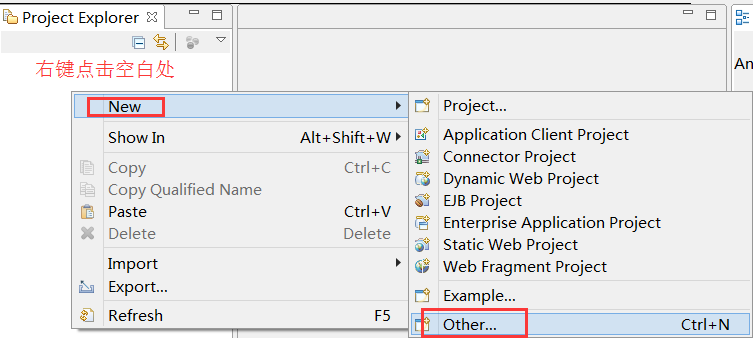


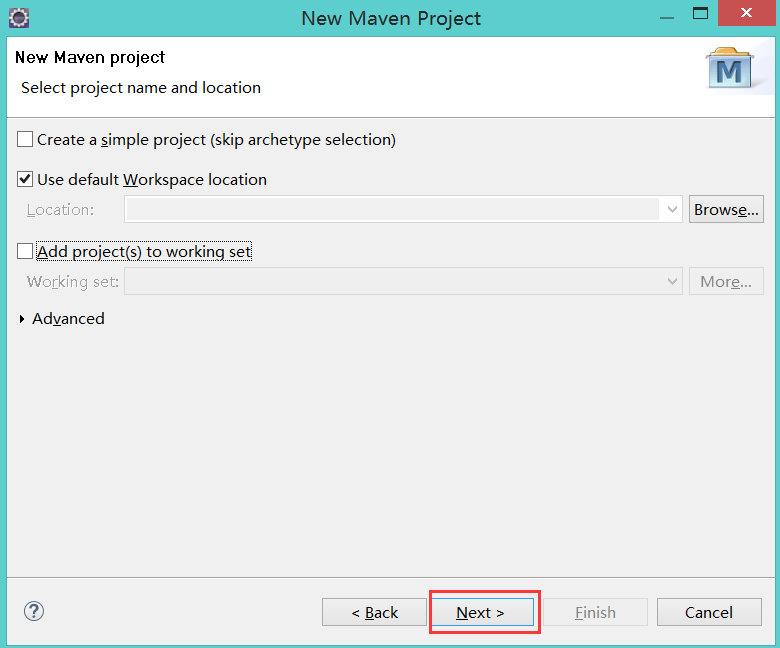
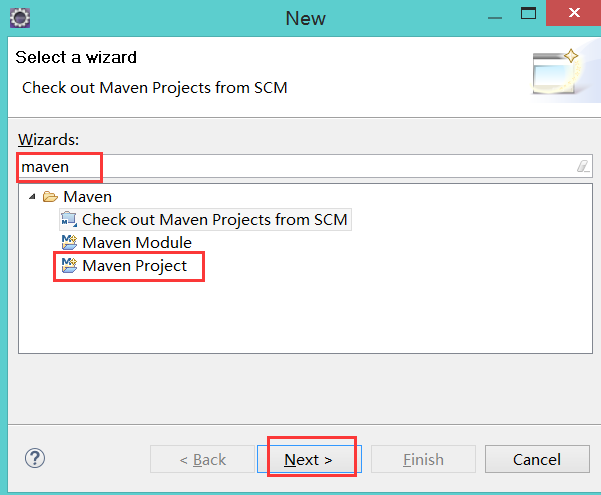


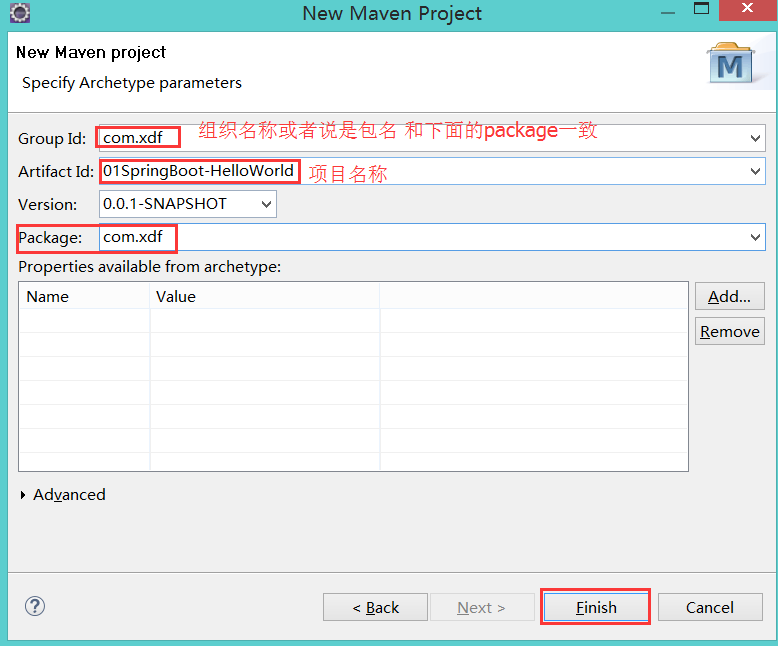
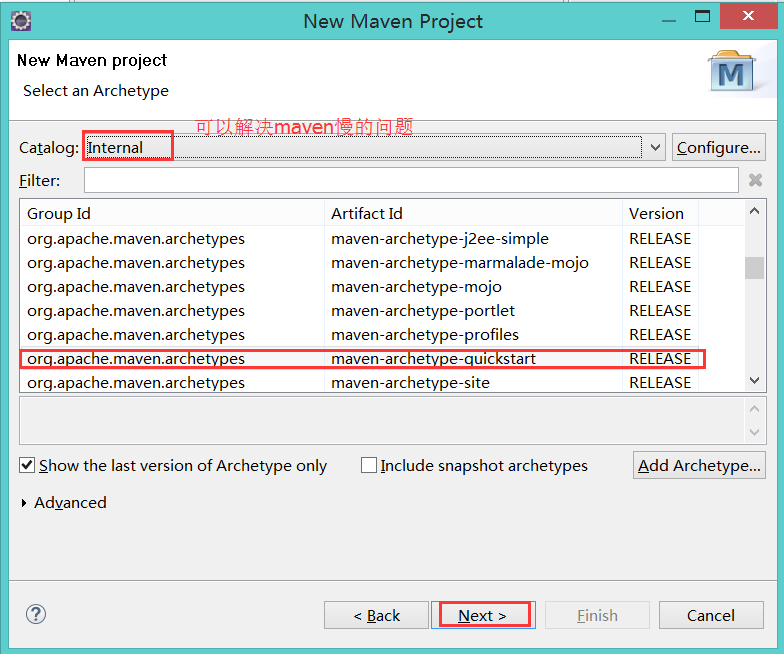


## 1.4:SpringBoot之HelloWorld

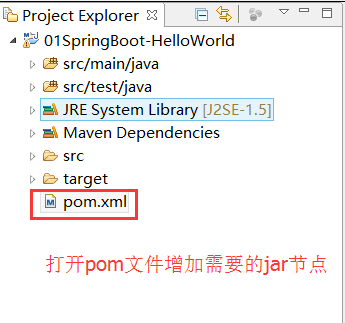
### )Eclipse中创建maven项目







### )配置pom文件节点



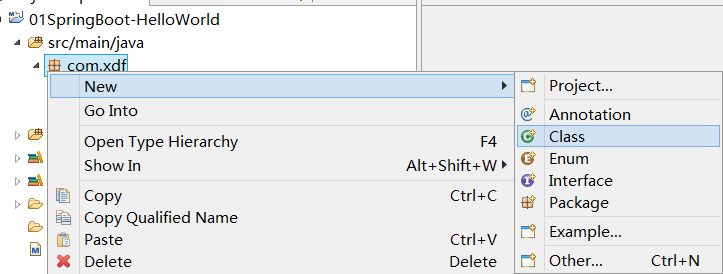


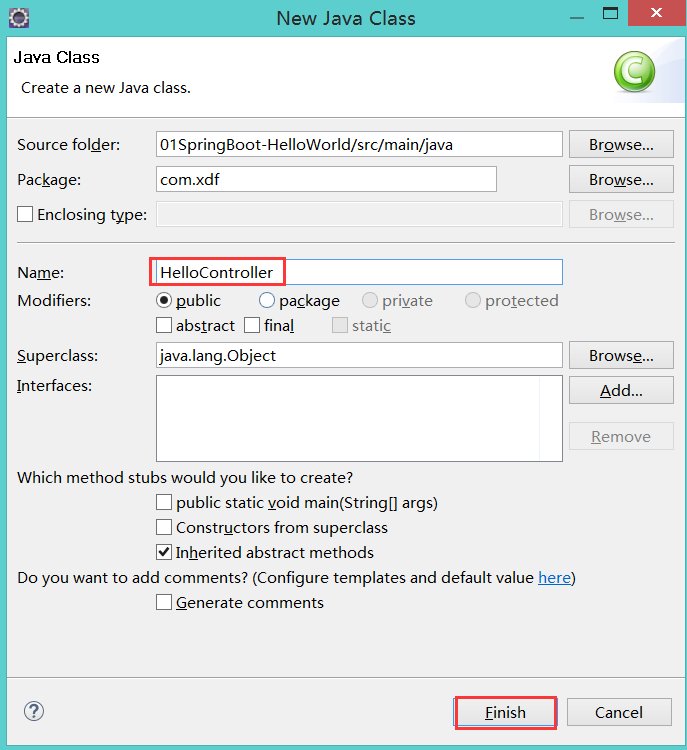
需要的pom文件！

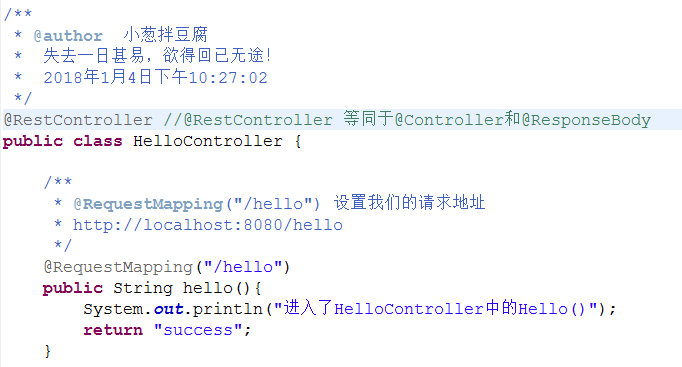


### )创建Controller

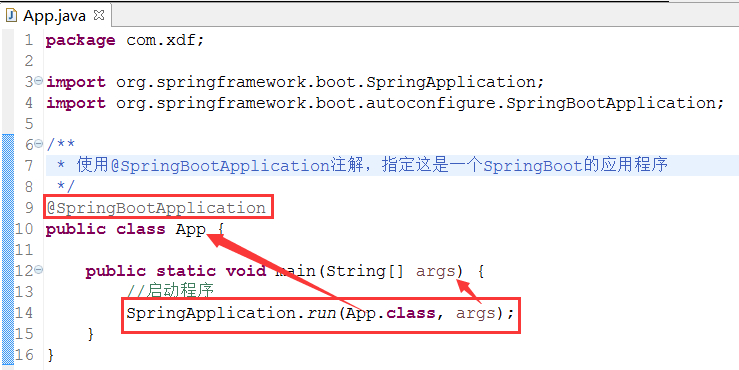
在src/main/java下的com.xdf包中创建



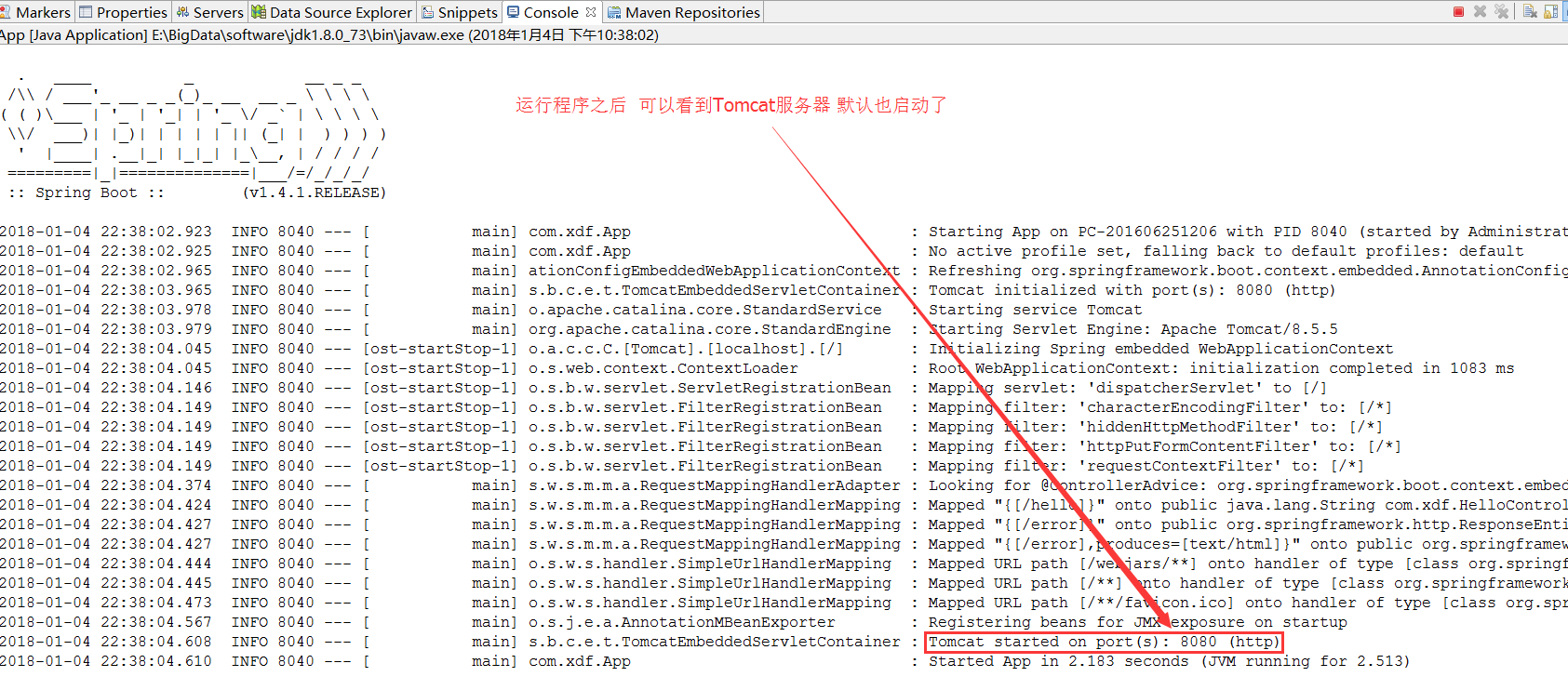




### )在App.java文件中书写测试代码并运行



启动方式：1.eclipse中直接运行

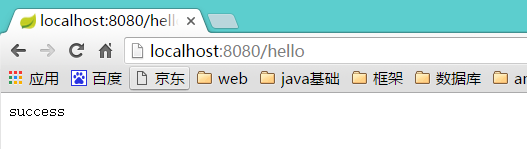


1. cmd命令进入文件在电脑中的所在位置

01.pushd 项目位置

02.mvn spring-boot:run 即可

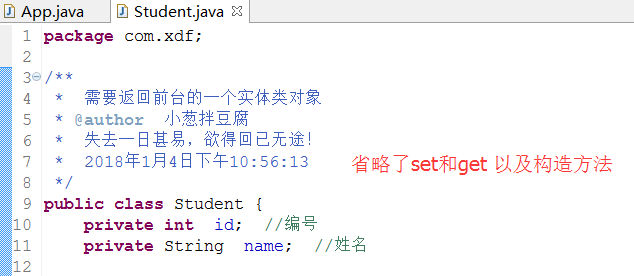
### )在浏览器中输入访问路径查看效果



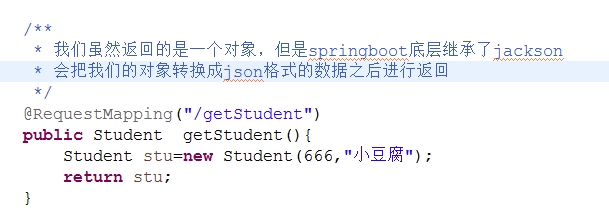
## 1.5:SpringBoot返回json数据

在src/main/java下的com.xdf包中创建

### )创建一个实体类对象Student

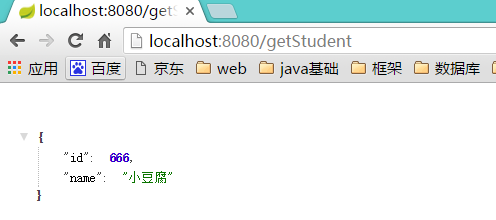


### )在HelloController文件中增加代码

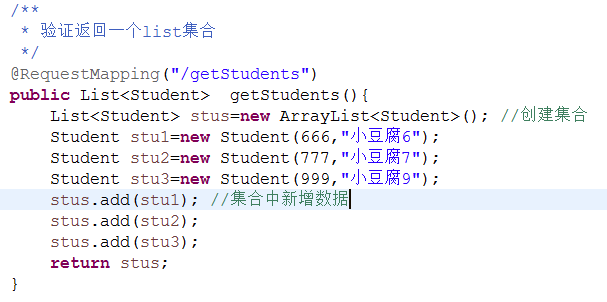


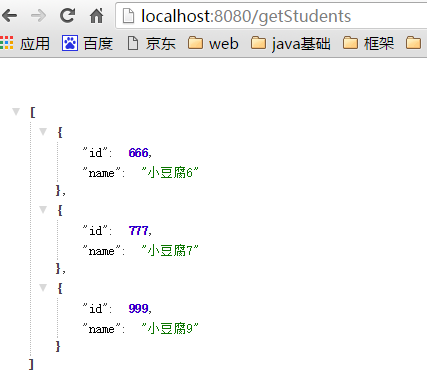
### )运行APP.java文件

### )在浏览器中输入访问路径查看效果

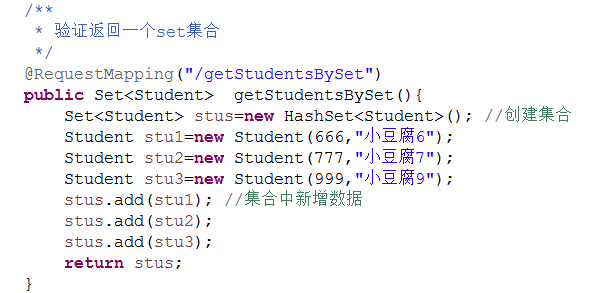


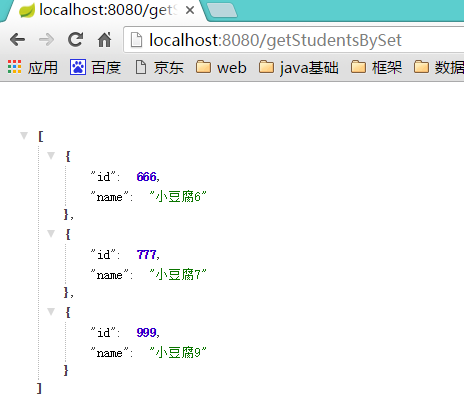
### (5.)返回list集合



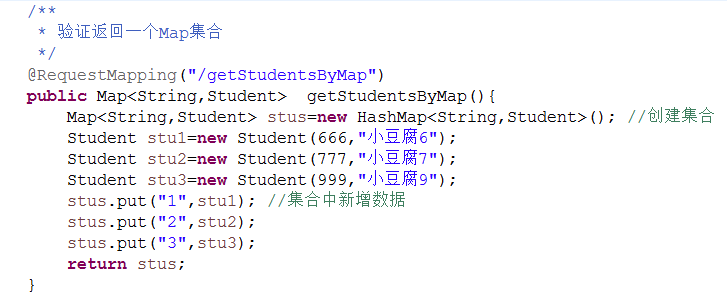


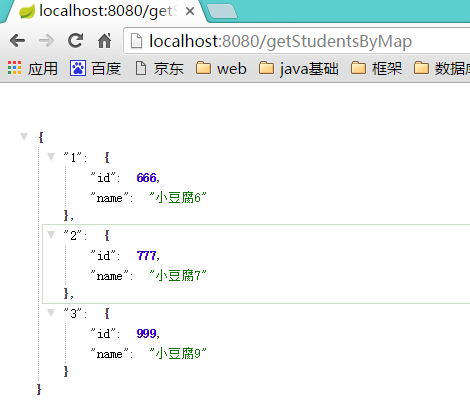
### (6.)返回set集合





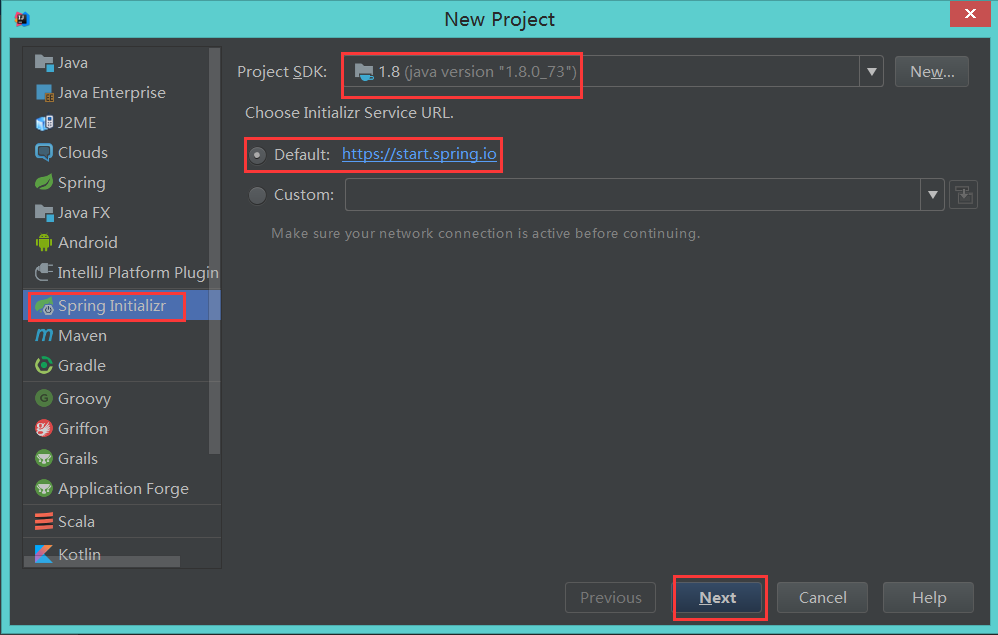
### (7.)返回Map集合

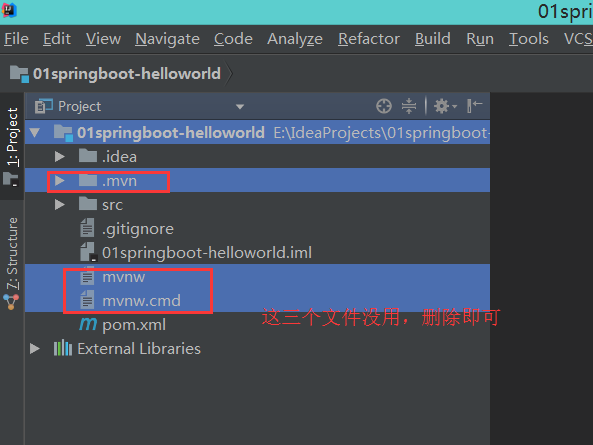
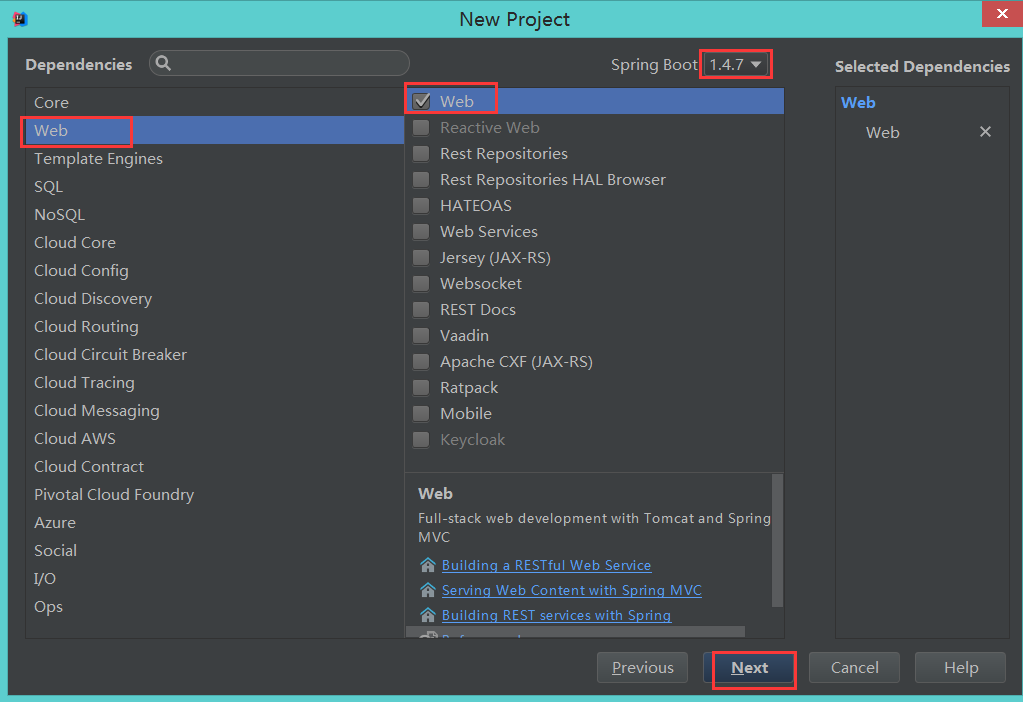
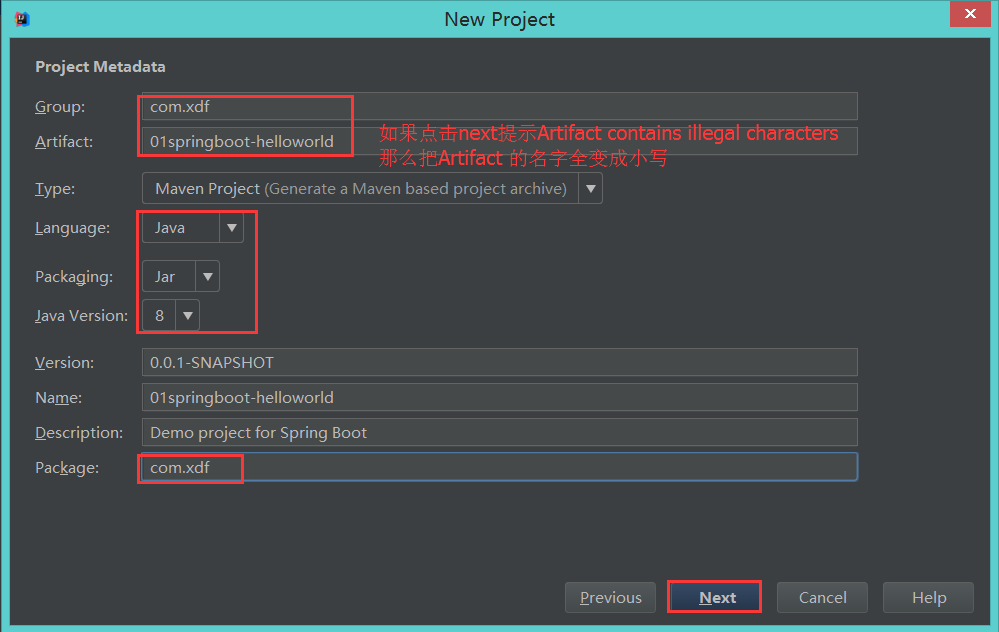




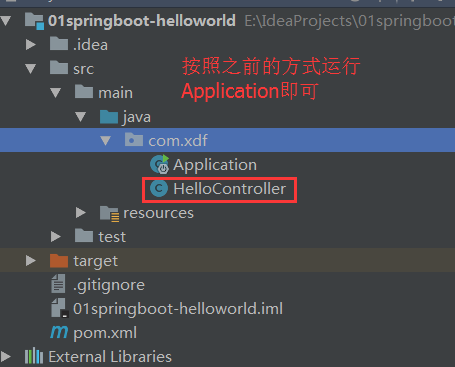
## 1.6:Idea中创建SpringBoot

如果没有Spring Initializr 请看下个内容

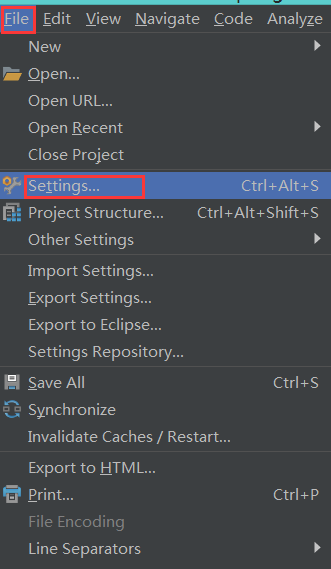


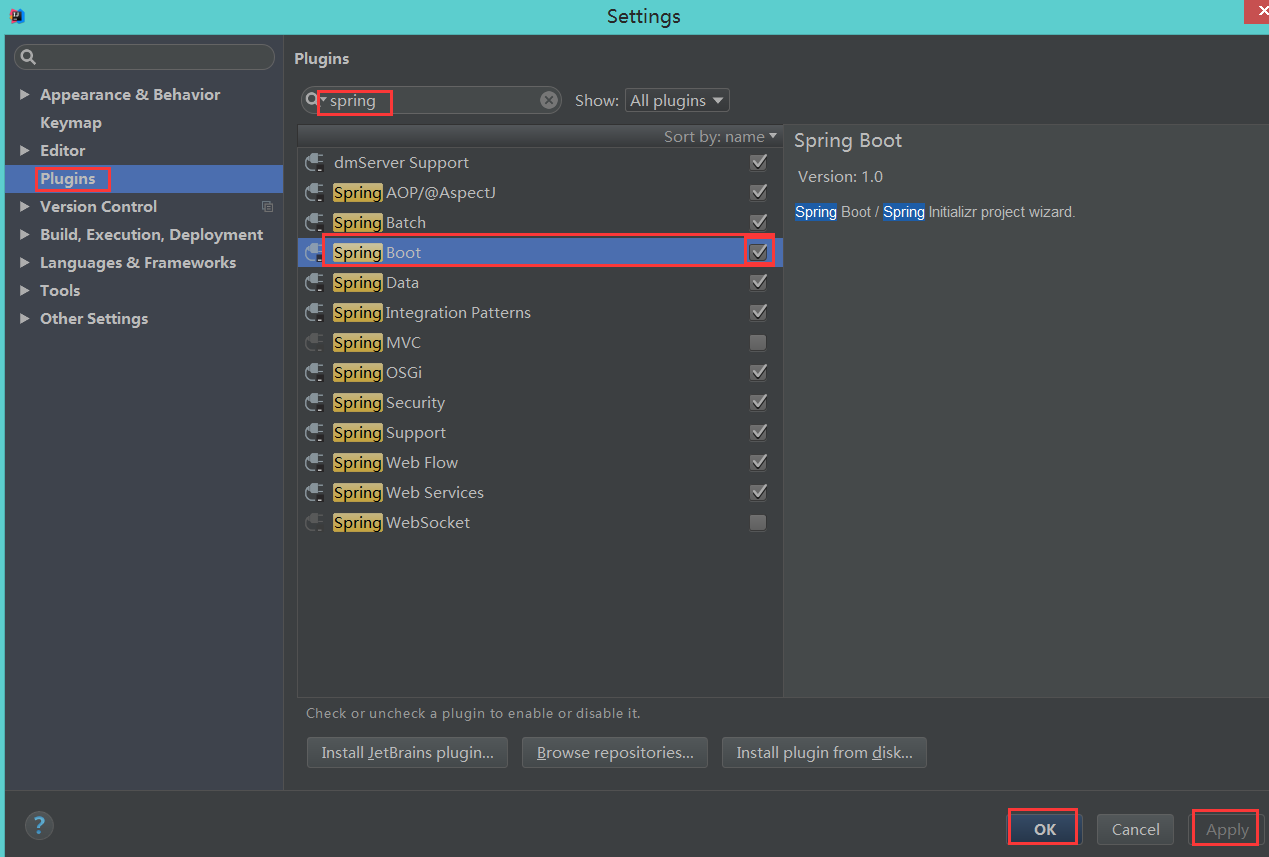


不需要配置pom文件，默认pom文件中的节点已经配置和下载了！



### 配置Spring Initializr





重启IDEA即可

## 1.7:SpringBoot使用fastjson解析数据

在pom文件中引入fastjson的依赖包

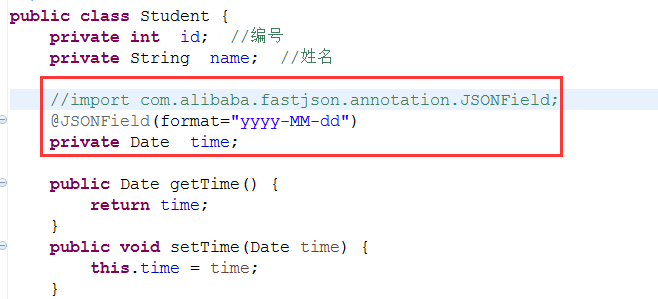


### )第一种方式 继承 WebMvcConfigurerAdapter

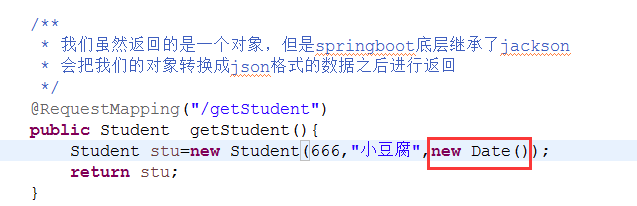
1. 让App这个启动类 继承 WebMvcConfigurerAdapter
2. 重写configureMessageConverters()



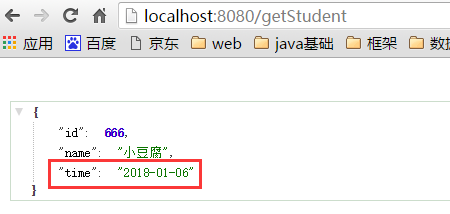
3.在Student类中增加一个Date类型的字段并且创建对应的set和get



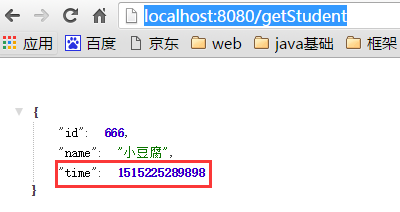
1. 修改HelloController中getStudent()



1. 运行程序，通过浏览器看效果



1. 如果不使用fastjson的效果图

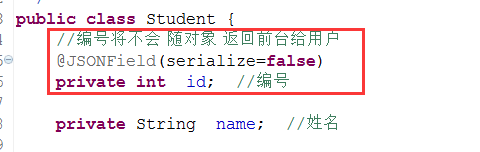


### )第二种方式@Bean

其他代码不动，只需要修改App中代码



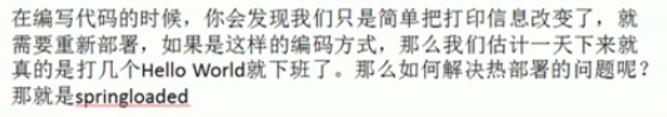
## 1.8:fastjson解析数据忽略部分属性



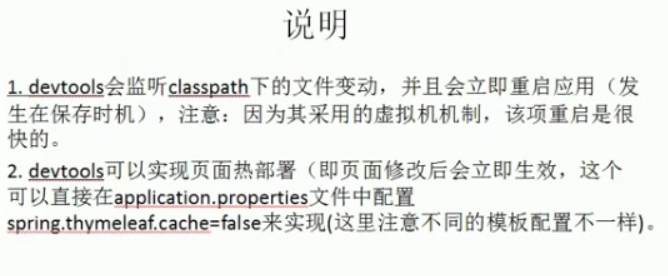
浏览器显示效果



## 1.9:SpringBoot热部署（springloader）







### )pom文件中新增热部署插件

<!-- Spring-boot-devtools 其实配置了这个就不需要下面的所有内容了-->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- 配置springloader 热部署插件 -->

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<fork>true</fork>

</configuration>

</plugin>

</plugins>

</build>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>springloaded</artifactId>

<version>1.2.4.RELEASE</version>

</dependency>

</dependencies>

<executions>

<execution>

<goals>

<goal>repackage</goal>

</goals>

<configuration>

<classifier>exec</classifier>

</configuration>

</execution>

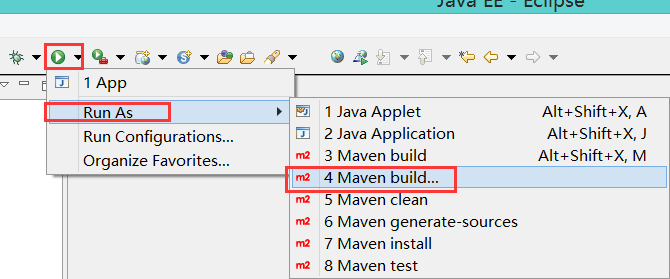
</executions>

</plugin>

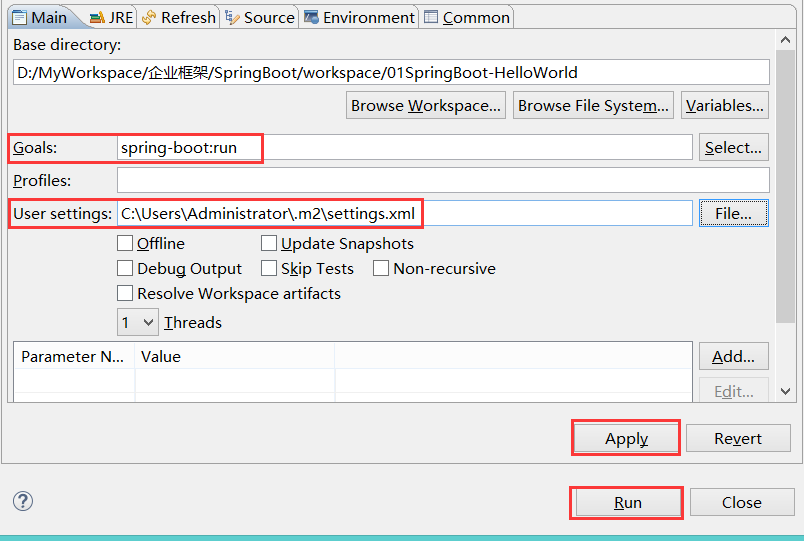
</plugins>

</build>

### (2.)第一种方式启动



spring-boot:run



如果启动错误如下

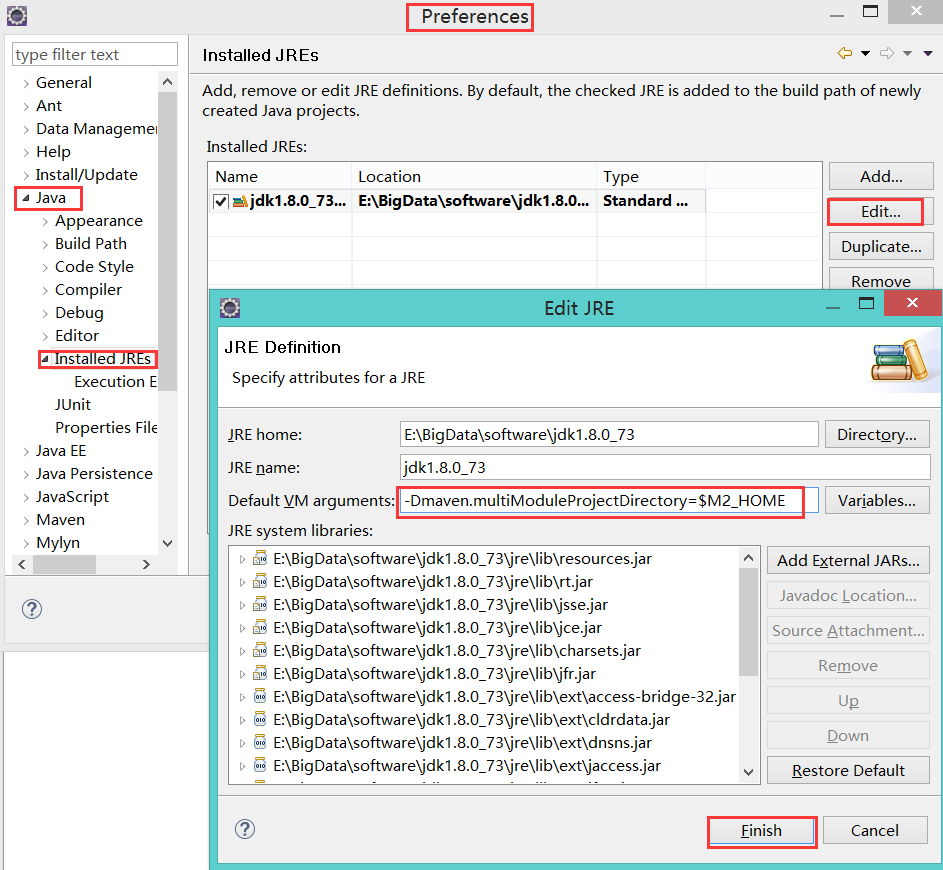


解决办法

Window->Preference->Java->Installed JREs->Edit

在Default VM arguments中设置

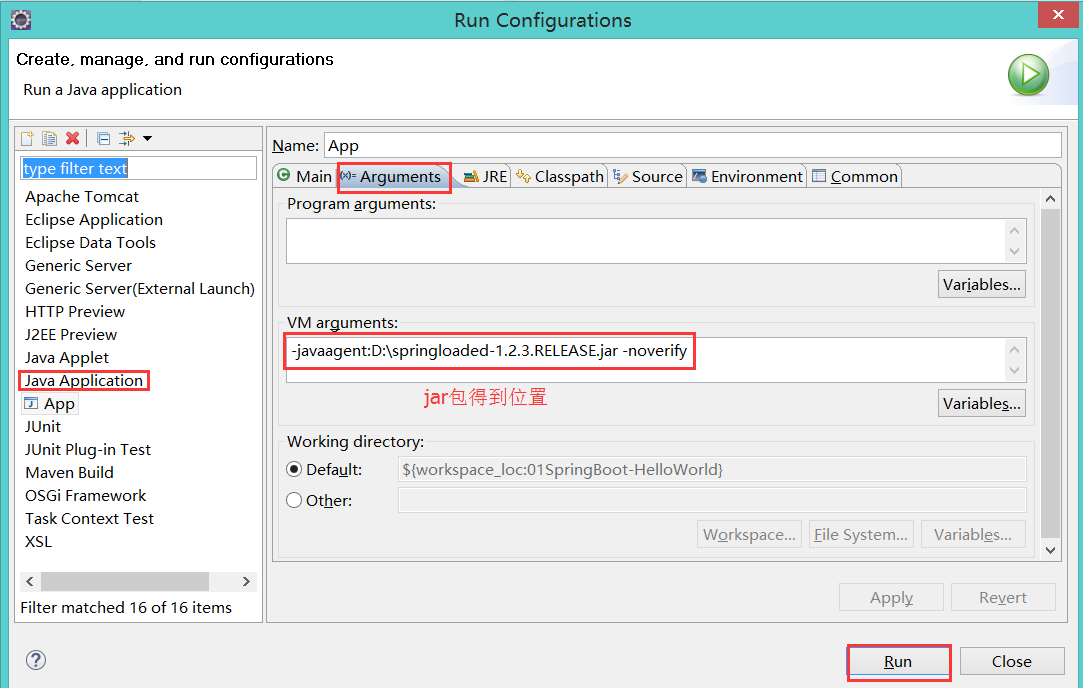
-Dmaven.multiModuleProjectDirectory=$M2\_HOME



### (3.)第二种方式启动

需要下载jar包

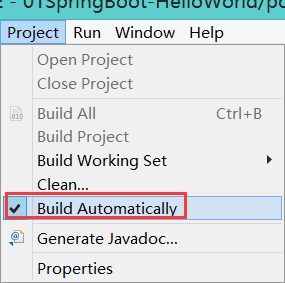




-javaagent:D:\springloaded-1.2.3.RELEASE.jar -noverify

之后我们再去修改代码或者新增类就不需要重新启动了！

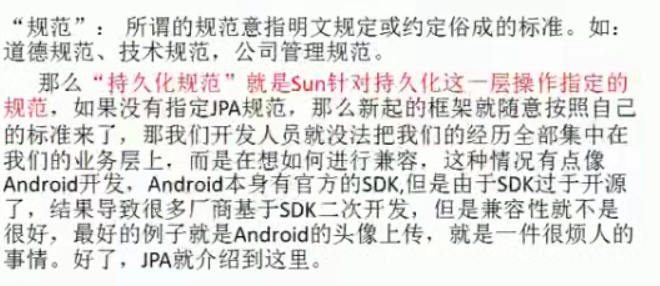
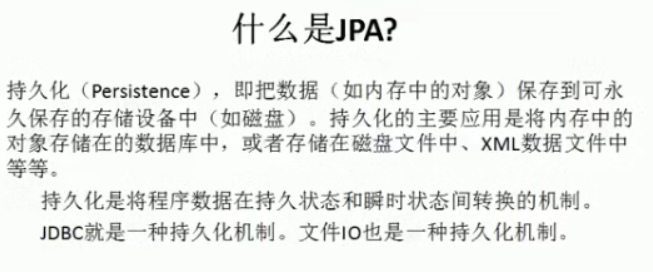
### (4.)可能出现的问题

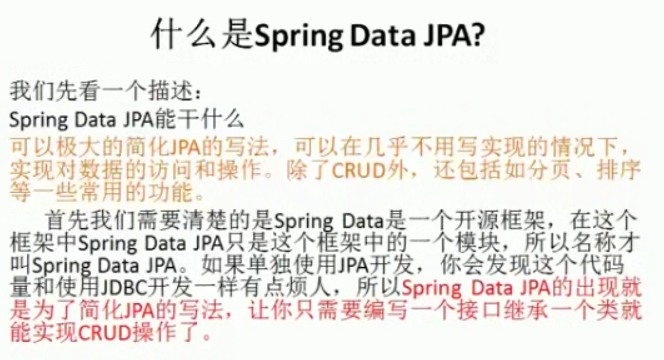
 自动编译的功能

## 1.10:SpringBoot 与 JPA

### (1.)基本理论

JPA是Java Persistence API的简称，中文名Java持久层API，是JDK 5.0注解或XML描述对象－关系表的映射关系，并将运行期的实体[对象持久化](https://baike.baidu.com/item/%E5%AF%B9%E8%B1%A1%E6%8C%81%E4%B9%85%E5%8C%96" \t "https://baike.baidu.com/item/JPA/_blank)到数据库中。





### )实现数据的增删改查

1.在pom文件中引入需要的依赖

<!-- 引入mysql需要的依赖 -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<!-- 引入spring data jpa需要的依赖 -->

<dependency>

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

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

</dependency>

1. 创建resource folder src/main/resource 文件夹
2. 创建并配置application.properties文件

spring.datasource.url=jdbc:mysql://localhost:3306/test

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driverClassName=com.mysql.jdbc.Driver

spring.datasource.max-active=20

spring.datasource.max-idle=5

spring.datasource.min-idle=5

spring.datasource.initial-size=10

spring.jpa.database=MYSQL

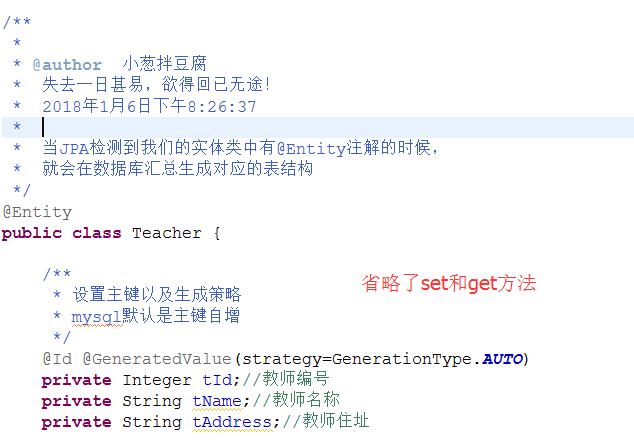
spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

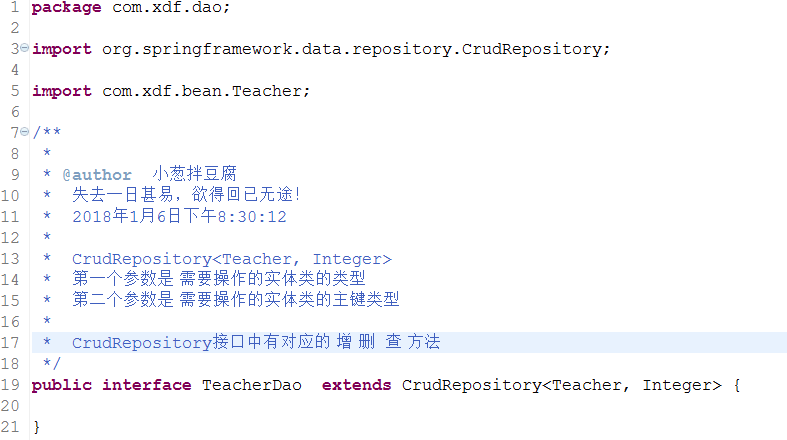
spring.jpa.hibernate.naming-strategy=org.hibernate.cfg.ImprovedNamingStrategy

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

1. 创建Teacher 实体类



1. 创建dao层



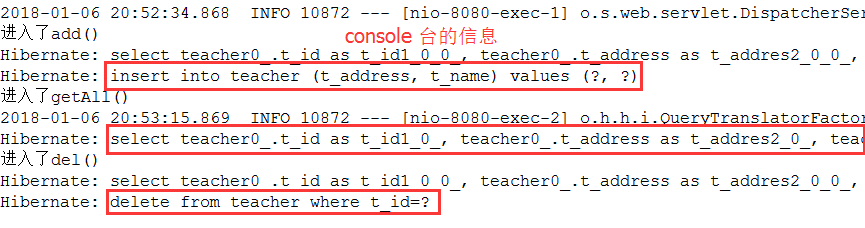
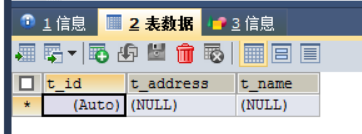
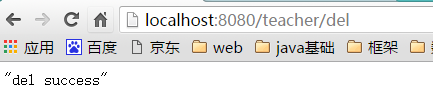
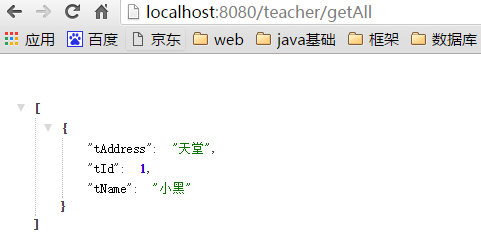
1. 创建service层



1. 创建controller层



1. 测试代码 观察效果

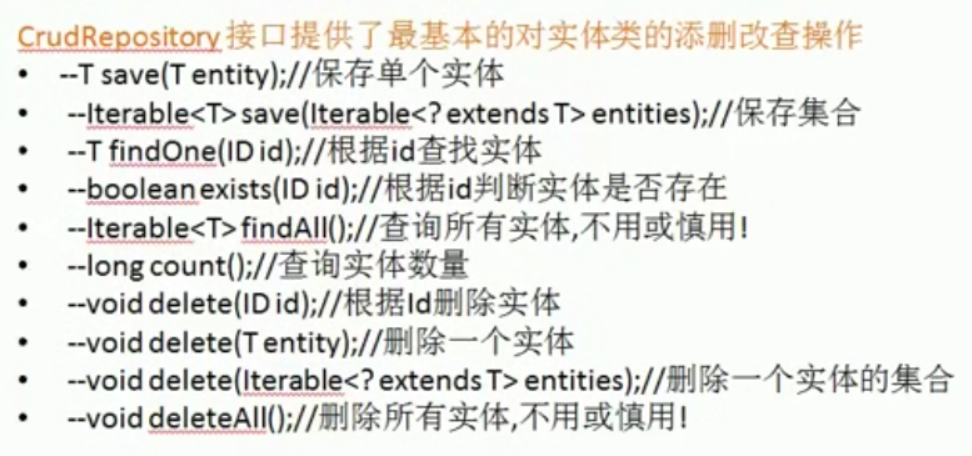


### )Repository接口



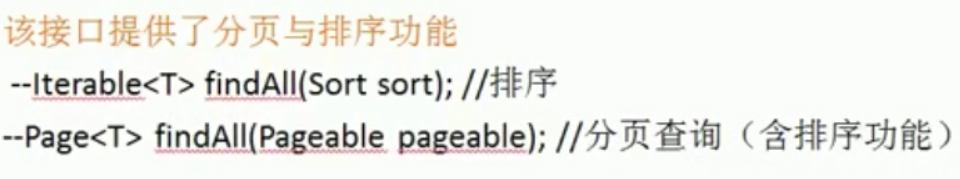
### )CrudRepository接口





### )PagingAndSortingRepository接口





## 1.11:SpringBoot JdbcTemplate

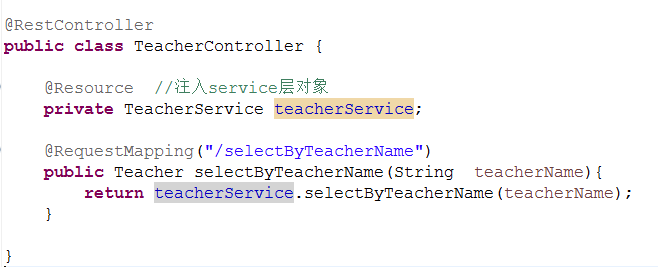
### )创建TecherDao类



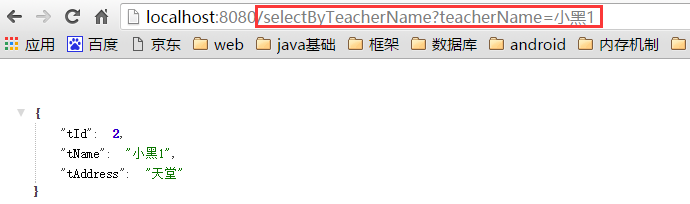
### )创建TeacherService类



### )创建TeacherController类



### )通过浏览器访问即可



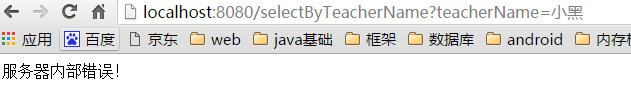
## 1.12:SpringBoot异常处理

针对于1.11的案例，如果我们再浏览器中输入了一个数据库中没有的名字，则会发生异常！而且用户会看到错误信息，显然这是不友好的！所以引入了我们的异常处理！

### (1.)创建一个异常处理类MyException



### (2.)通过浏览器访问即可



## 1.13:SpringBoot配置server

### (1.)**server配置**

server.address指定server绑定的地址

server.compression.enabled是否开启压缩，默认为false.

server.compression.excluded-user-agents指定不压缩的user-agent，多个以逗号分隔，默认值为:text/html,text/xml,text/plain,text/css

server.compression.mime-types指定要压缩的MIME type，多个以逗号分隔.

server.compression.min-response-size执行压缩的阈值，默认为2048

server.context-parameters.[param name]设置servlet context 参数

server.context-path设定应用的context-path.

server.display-name设定应用的展示名称，默认: application

server.jsp-servlet.class-name设定编译JSP用的servlet，默认: org.apache.jasper

.servlet.JspServlet)

server.jsp-servlet.init-parameters.[param name]设置JSP servlet 初始化参数.

server.jsp-servlet.registered设定JSP servlet是否注册到内嵌的servlet容器，默认true

server.port设定http监听端口

server.servlet-path设定dispatcher servlet的监听路径，默认为: /

### (2.)**cookie、session配置**

server.session.cookie.comment指定session cookie的comment

server.session.cookie.domain指定session cookie的domain

server.session.cookie.http-only是否开启HttpOnly.

server.session.cookie.max-age设定session cookie的最大age.

server.session.cookie.name设定Session cookie 的名称.

server.session.cookie.path设定session cookie的路径.

server.session.cookie.secure设定session cookie的“Secure” flag.

server.session.persistent重启时是否持久化session，默认false

server.session.timeoutsession的超时时间

server.session.tracking-modes设定Session的追踪模式(cookie, url, ssl).

### (3.)**ssl配置**

server.ssl.ciphers是否支持SSL ciphers.

server.ssl.client-auth设定client authentication是wanted 还是 needed.

server.ssl.enabled是否开启ssl，默认: true

server.ssl.key-alias设定key store中key的别名.

server.ssl.key-password访问key store中key的密码.

server.ssl.key-store设定持有SSL certificate的key store的路径，通常是一个.jks文件.

server.ssl.key-store-password设定访问key store的密码.

server.ssl.key-store-provider设定key store的提供者.

server.ssl.key-store-type设定key store的类型.

server.ssl.protocol使用的SSL协议，默认: TLS

server.ssl.trust-store持有SSL certificates的Trust store.

server.ssl.trust-store-password访问trust store的密码.

server.ssl.trust-store-provider设定trust store的提供者.

server.ssl.trust-store-type指定trust store的类型.

### (4.)**tomcat**

server.tomcat.access-log-enabled是否开启access log ，默认: false)

server.tomcat.access-log-pattern设定access logs的格式，默认: common

server.tomcat.accesslog.directory设定log的目录，默认: logs

server.tomcat.accesslog.enabled是否开启access log，默认: false

server.tomcat.accesslog.pattern设定access logs的格式，默认: common

server.tomcat.accesslog.prefix设定Log 文件的前缀，默认: access\_log

server.tomcat.accesslog.suffix设定Log 文件的后缀，默认: .log

server.tomcat.background-processor-delay后台线程方法的Delay大小: 30

server.tomcat.basedir设定Tomcat的base 目录，如果没有指定则使用临时目录.

server.tomcat.internal-proxies设定信任的正则表达式，默认:“10\.\d{1,3}\.\d{1,3}\.\d{1,3}| 192\.168\.\d{1,3}\.\d{1,3}| 169\.254\.\d{1,3}\.\d{1,3}| 127\.\d{1,3}\.\d{1,3}\.\d{1,3}| 172\.1[6-9]{1}\.\d{1,3}\.\d{1,3}| 172\.2[0-9]{1}\.\d{1,3}\.\d{1,3}|172\.3[0-1]{1}\.\d{1,3}\.\d{1,3}”

server.tomcat.max-http-header-size设定http header的最小值，默认: 0

server.tomcat.max-threads设定tomcat的最大工作线程数，默认为: 0

server.tomcat.port-header设定http header使用的，用来覆盖原来port的value.

server.tomcat.protocol-header设定Header包含的协议，通常是 X-Forwarded-Proto，如果remoteIpHeader有值，则将设置为RemoteIpValve.

server.tomcat.protocol-header-https-value设定使用SSL的header的值，默认https.

server.tomcat.remote-ip-header设定remote IP的header，如果remoteIpHeader有值，则设置为RemoteIpValve

server.tomcat.uri-encoding设定URI的解码字符集.

### (5.)**undertow**

server.undertow.access-log-dir设定Undertow access log 的目录，默认: logs

server.undertow.access-log-enabled是否开启access log，默认: false

server.undertow.access-log-pattern设定access logs的格式，默认: common

server.undertow.accesslog.dir设定access log 的目录.

server.undertow.buffer-size设定buffer的大小.

server.undertow.buffers-per-region设定每个region的buffer数

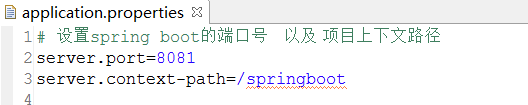
server.undertow.direct-buffers设定堆外内存

server.undertow.io-threads设定I/O线程数.

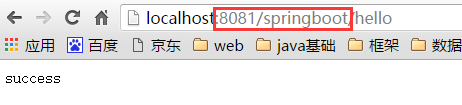
server.undertow.worker-threads设定工作线程数

### (6.)配置端口号和项目路径案例

修改application.properties文件



浏览器中测试



## 1.14:SpringBoot使用thymeleaf

### (1.)pom文件中增加thymeleaf的依赖和版本号

<!-- 添加thymeleaf模版的依赖 -->

<dependency>

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

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

</dependency>

在properties节点中增加，因为thymeleaf 模版需要html页面的所有元素都自身闭合！

3.0之后的版本不在需要！

<thymeleaf.version>3.0.0.RELEASE</thymeleaf.version>

<thymeleaf-layout-dialect.version>2.0.0</thymeleaf-layout-dialect.version>

### (2.)application.properties文件中新增thymeleaf配置

# 设置thymeleaf

#spring.thymeleaf.prefix=classpath:/templates/

#spring.thymeleaf.suffix=.html

#spring.thymeleaf.mode=HTML5

#spring.thymeleaf.encoding=UTF-8

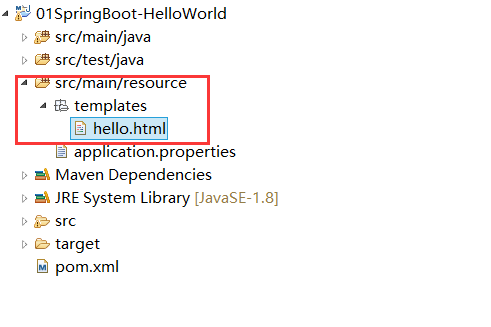
#spring.thymeleaf.content-type=text/html

#springboot 官方文档建议我们关闭thymeleaf的缓存

spring.thymeleaf.cache=fasle

### (3.)在src/main/resource文件夹下创建templates文件夹

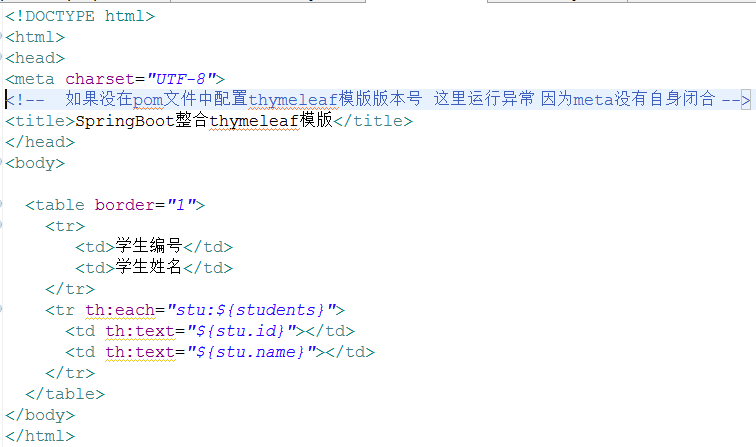
并创建一个hello.html以备后续使用



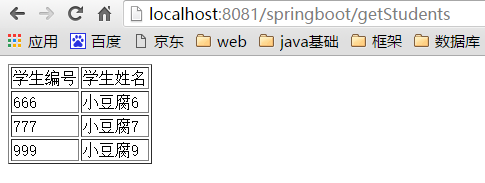
### (4.)创建HelloController书写代码



### )书写hello.html页面代码



### )浏览器测试



## 1.15:SpringBoot使用freemarker

### (1.)pom文件中增加freemarker的依赖

<!-- 添加freemarker模版的依赖 -->

<dependency>

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

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

</dependency>

### (2.)application.properties文件中新增freemarker配置

## Freemarker 配置

spring.freemarker.template-loader-path=classpath:/templates/

spring.freemarker.cache=false

spring.freemarker.charset=UTF-8

spring.freemarker.check-template-location=true

spring.freemarker.content-type=text/html

spring.freemarker.expose-request-attributes=false

spring.freemarker.expose-session-attributes=false

spring.freemarker.request-context-attribute=request

下面的配置可供参考

spring.freemarker.allow-request-override=false # Set whether HttpServletRequest attributes are allowed to override (hide) controller generated model attributes of the same name.

spring.freemarker.allow-session-override=false # Set whether HttpSession attributes are allowed to override (hide) controller generated model attributes of the same name.

spring.freemarker.cache=false # Enable template caching.

spring.freemarker.charset=UTF-8 # Template encoding.

spring.freemarker.check-template-location=true # Check that the templates location exists.

spring.freemarker.content-type=text/html # Content-Type value.

spring.freemarker.enabled=true # Enable MVC view resolution for this technology.

spring.freemarker.expose-request-attributes=false # Set whether all request attributes should be added to the model prior to merging with the template.

spring.freemarker.expose-session-attributes=false # Set whether all HttpSession attributes should be added to the model prior to merging with the template.

spring.freemarker.expose-spring-macro-helpers=true # Set whether to expose a RequestContext for use by Spring's macro library, under the name "springMacroRequestContext".

spring.freemarker.prefer-file-system-access=true # Prefer file system access for template loading. File system access enables hot detection of template changes.

spring.freemarker.prefix= # Prefix that gets prepended to view names when building a URL.

spring.freemarker.request-context-attribute= # Name of the RequestContext attribute for all views.

spring.freemarker.settings.\*= # Well-known FreeMarker keys which will be passed to FreeMarker's Configuration.

spring.freemarker.suffix= # Suffix that gets appended to view names when building a URL.

spring.freemarker.template-loader-path=classpath:/templates/ # Comma-separated list of template paths.

spring.freemarker.view-names= # White list of view names that can be resolved.

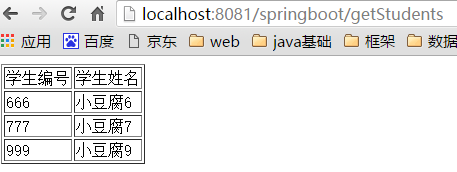
### (3.)在src/main/resource/templates文件夹中创建helloFtl.ftl文件



### (4.)创建HelloController书写代码

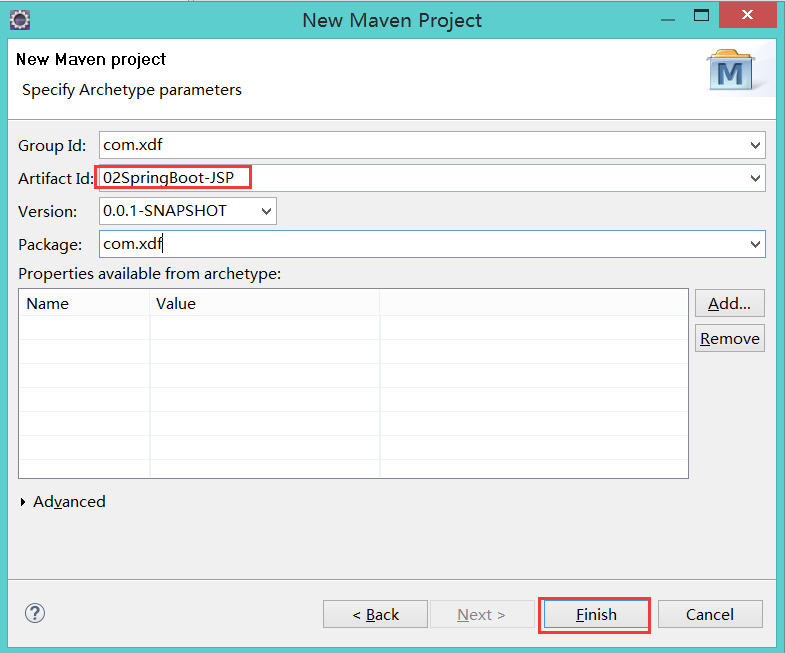
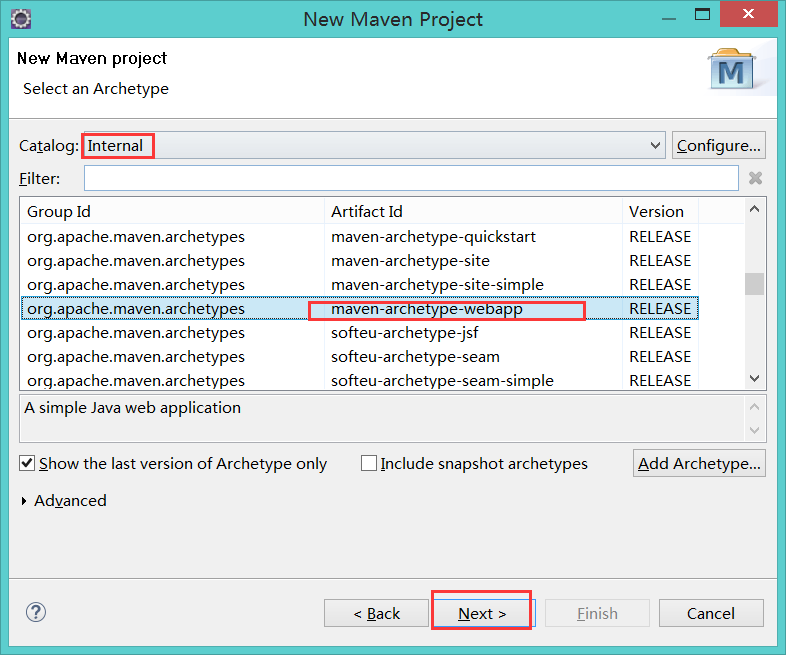


### (5.)浏览器测试



## 1.16:SpringBoot使用JSP

### )创建一个maven web Project



### )pom文件中增加依赖包



<!--

配置springboot的父节点依赖，之后引入就不需要添加version配置了！

springboot会自动选择最合适的版本进行添加依赖

-->

<parent>

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

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

<version>1.4.1.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<!--指定JDK版本 我们使用的是1.8，不配置默认环境是1.6 -->

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<!-- spring-boot-starter-web是为我们提供了包括mvc,aop等需要的一些jar -->

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

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

<!-- 因为我们已经配置了 parent 中的version 所以这里不需要指定version了 -->

</dependency>

<!-- Spring-boot-devtools -->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- 配置servlet需要的依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

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

</dependency>

<!-- 配置jstl标签库需要的依赖 -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<!-- 添加tomcat的支持 -->

<dependency>

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

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

</dependency>

<dependency>

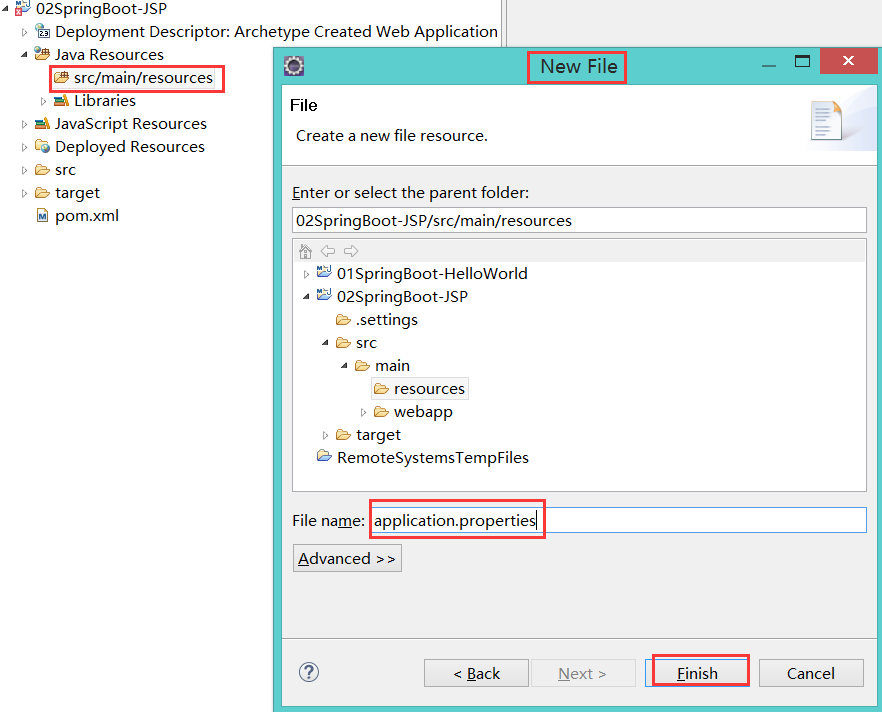
<groupId>org.apache.tomcat.embed</groupId>

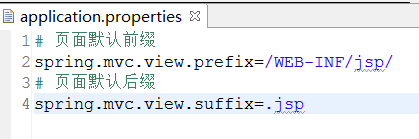
<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

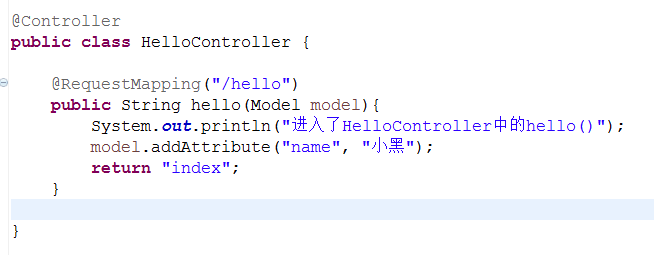
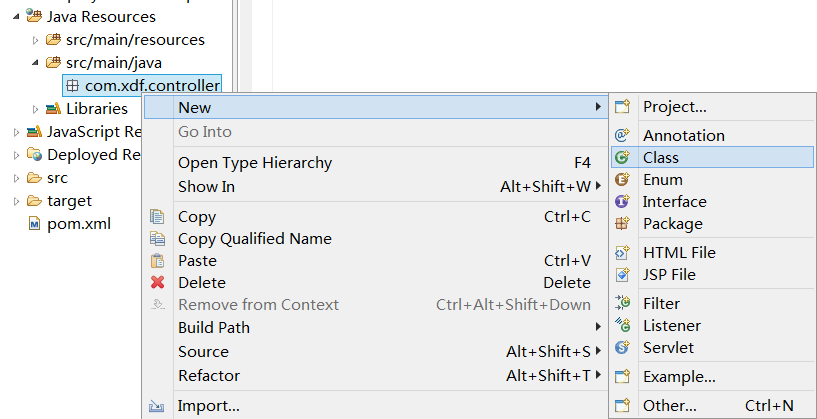
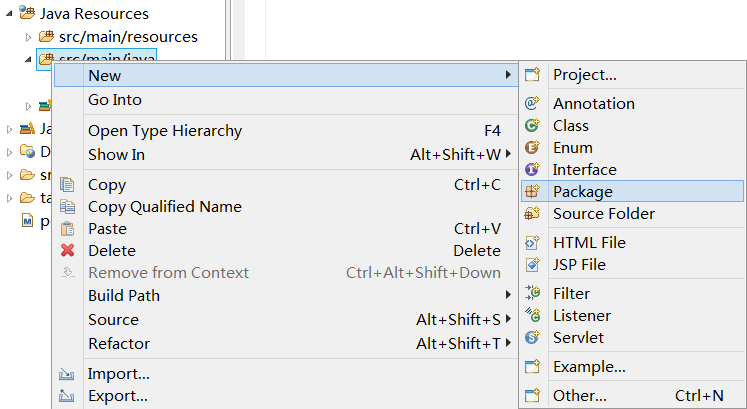
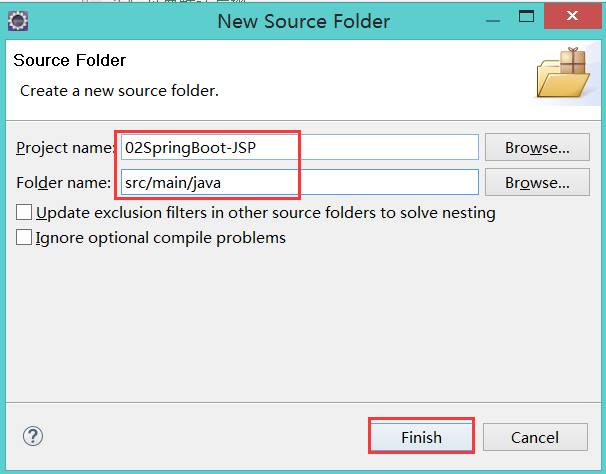
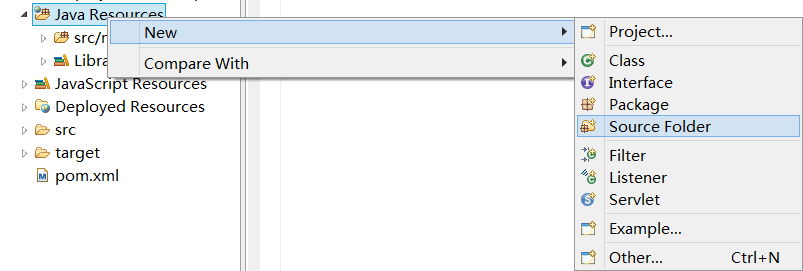
</dependencies>

### )创建一个配置文件 并 配置

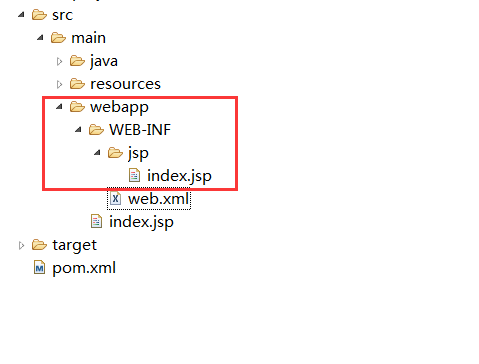




### )创建一个controller

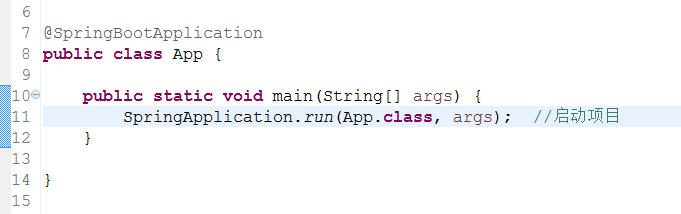
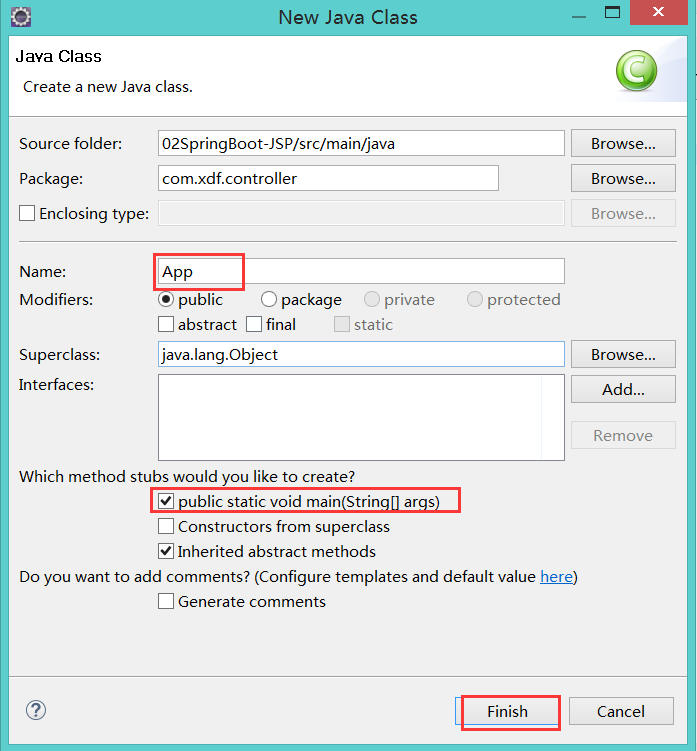
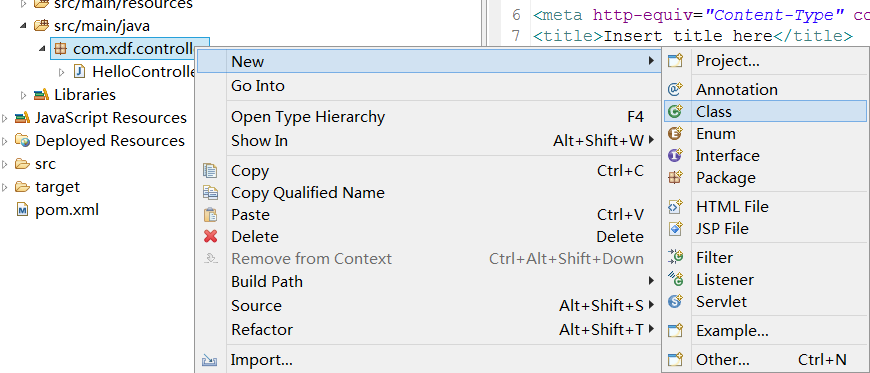


### )在WEB-INF创建jsp文件夹和index.jsp文件





### )创建启动类App.java

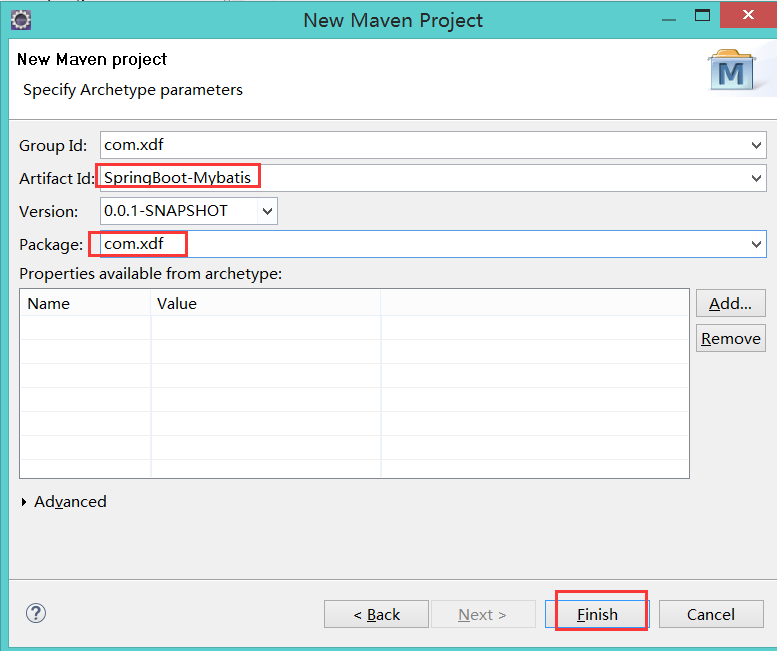
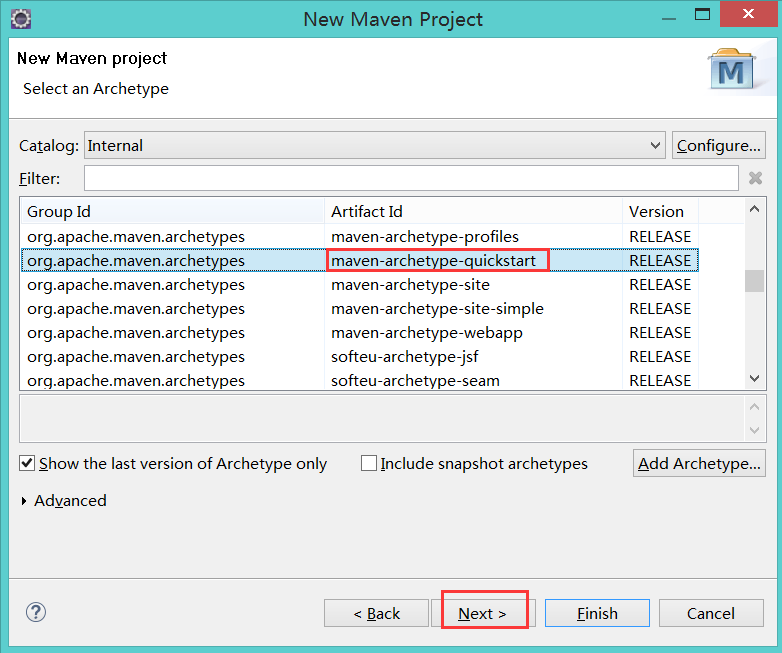
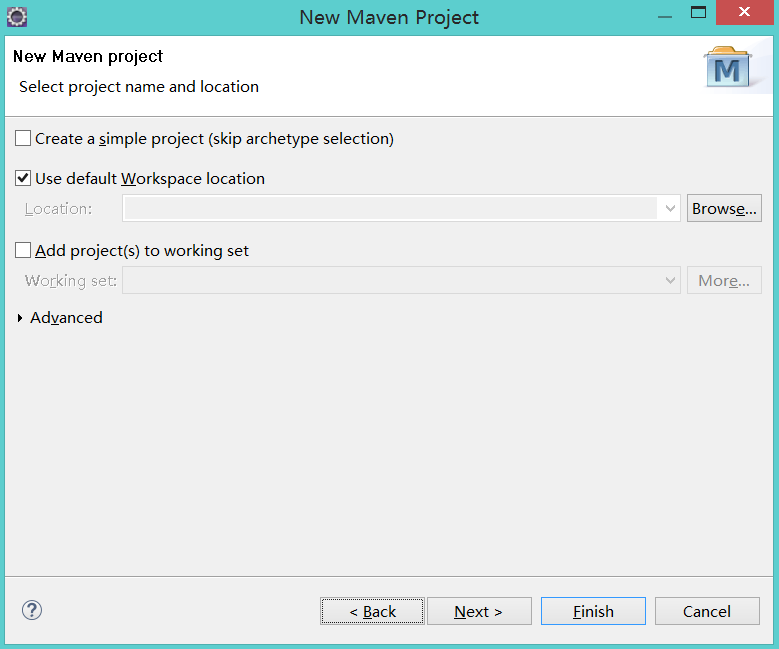


### )浏览器测试



## 1.17:SpringBoot整合Mybatis

### (1.)创建一个maven Project



### (2.)引入需要的pom文件节点

<!--

配置springboot的父节点依赖，之后引入就不需要添加version配置了！

springboot会自动选择最合适的版本进行添加依赖

-->

<parent>

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

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

<version>1.4.1.RELEASE</version>

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<!--指定JDK版本 我们使用的是1.8，不配置默认环境是1.6 -->

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<!-- spring-boot-starter-web是为我们提供了包括mvc,aop等需要的一些jar -->

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

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

<!-- 因为我们已经配置了 parent 中的version 所以这里不需要指定version了 -->

</dependency>

<!-- Spring-boot-devtools -->

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<optional>true</optional>

</dependency>

<!-- 引入mysql需要的依赖 -->

<dependency>

<groupId>mysql</groupId>

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

</dependency>

<!-- 引入springboot-mybatis的依赖 -->

<dependency>

<groupId>org.mybatis.spring.boot</groupId>

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

<version>1.1.1</version>

</dependency>

<!-- 配置Mybatis分页插件依赖 -->

<dependency>

<groupId>com.github.pagehelper</groupId>

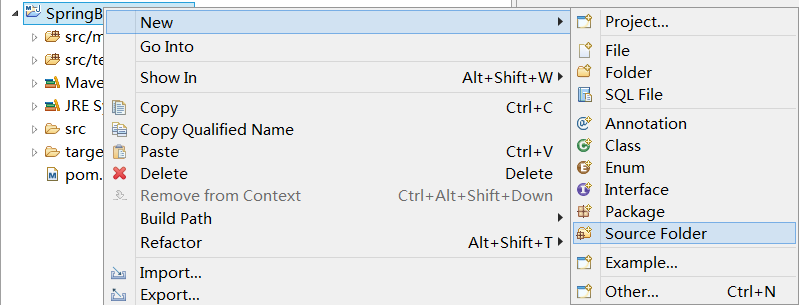
<artifactId>pagehelper</artifactId>

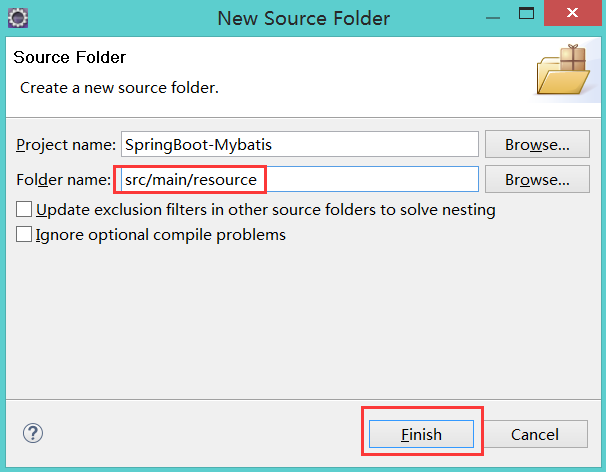
<version>4.1.0</version>

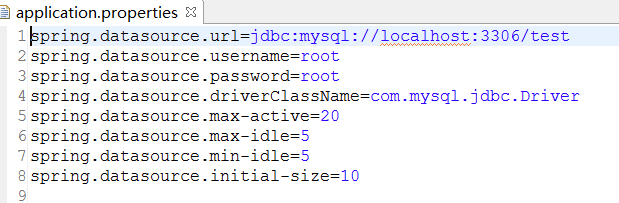
</dependency>

</dependencies>

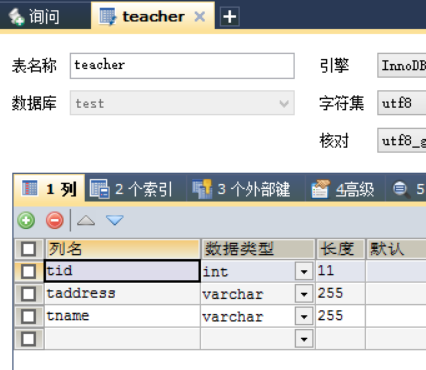
### )创建src/main/resource源文件夹和application.properties

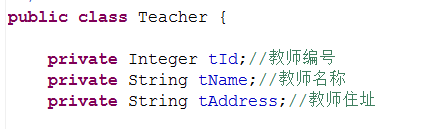




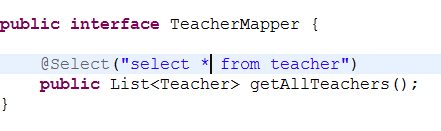


### )创建Teacher实体类和数据库表

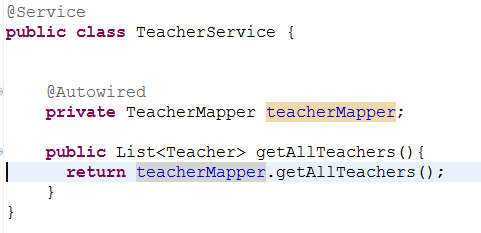




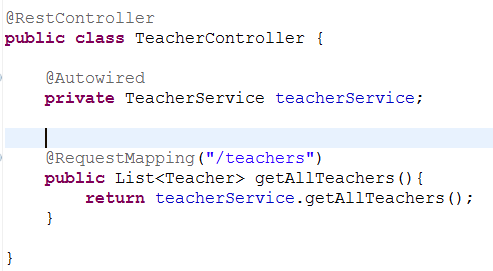
### )创建TeacherMapper接口



### )创建TeacherService实现类



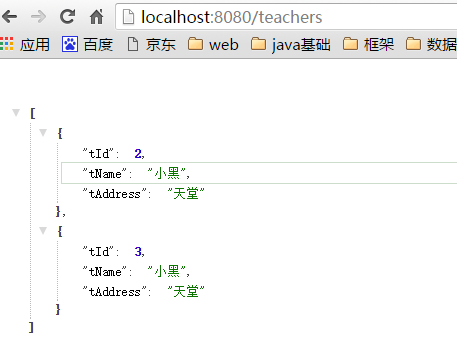
### )创建TeacherController



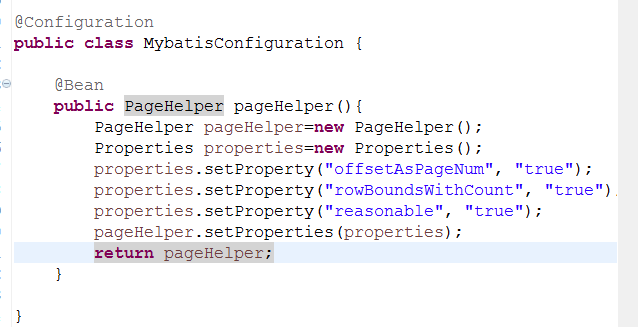
### )创建App启动类



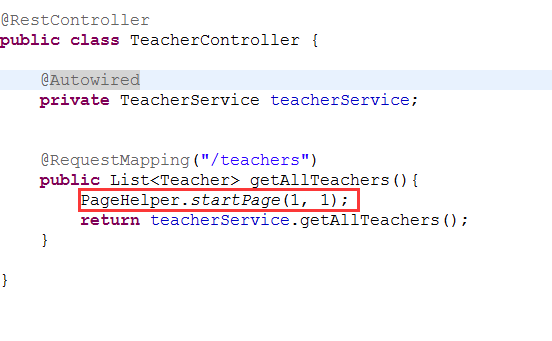
### )浏览器运行测试



### )分页显示数据创建一个配置文件



比如说现在想页面中只显示一条数据，只需要在controller中增加如下代码！

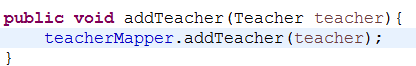


### )新增数据之后想获取id

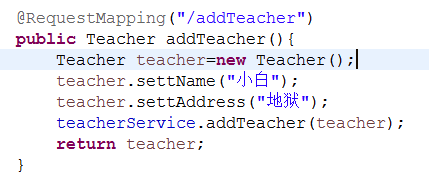
TeacherMapper中新增



TeacherService中新增



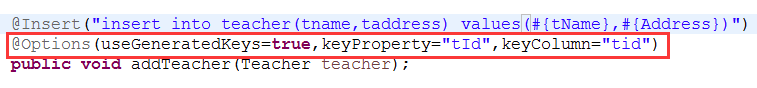
TeacherController中新增



浏览器中测试发现没有id



在TeacherMapper中增加代码



浏览器中测试

