

使用RestController注解，我们可以使用字符串或者JSON字符串直接返回到WEB页面或者APP,这样方便解析,RestController其实就等于Controller加ResponseBody（表示返回的事字符串或者JSON字符串）。

对于开源框架和第三方开源库，我们只需要直接引用就可以，因为SpringBoot已经帮我们做好了封装。



StringBoot不需要打包就可以直接把自己的一个应用放到Tomcat中直接运行，但是你想打成war包也是可以。



Maven配置，只需要写一个Maven jar包，就可以把相关的所有的依赖全部引入到我们得项目中，但是这样也会把我们不需要的依赖也都引入进来，所以会让我们的应用显得比较臃肿



SpringBoot最大的特点是可以通过注解可以跟xml一样实现相同的功能，



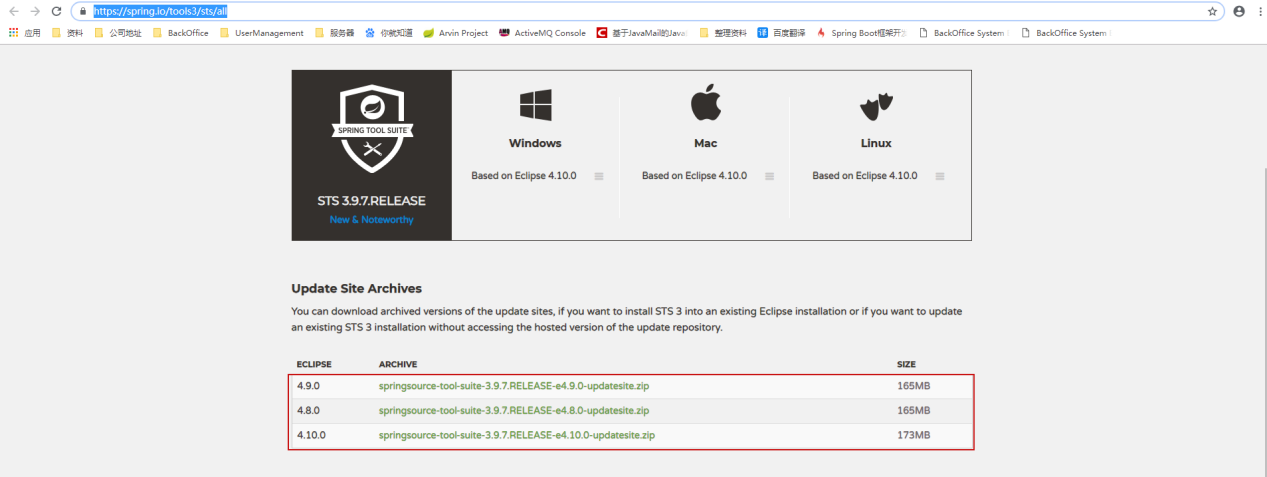
@SpringBootApplication

表示此类就是我们项目的一个主类，启动它就是启动项目

1. **使用eclipse中的STS插件生成StringBoot项目步骤：（建议使用第二种，不会存在版本不兼容问题，比较方便）**

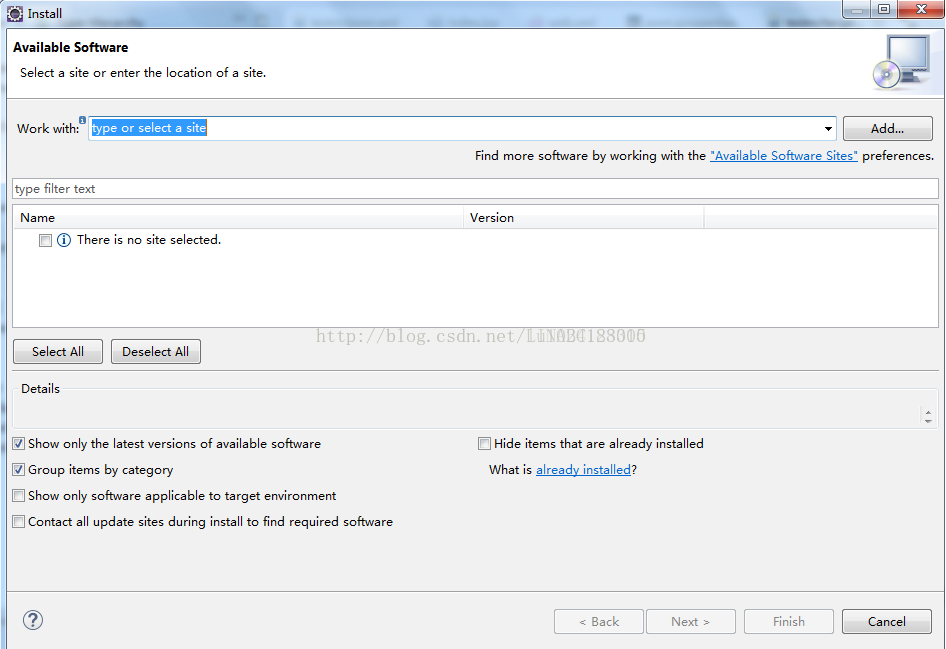
**安装STS插件方式一**

1.https://spring.io/tools3/sts/all，使用此网址进去网页，如下图：

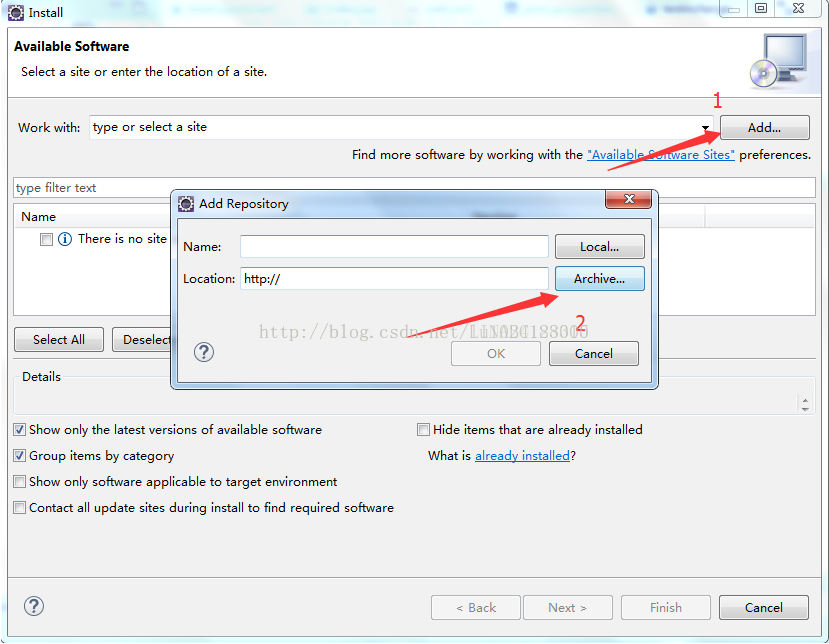


这三种可以随便使用一种，我使用的是第二种，点击需要的会自动下载一个压缩包。

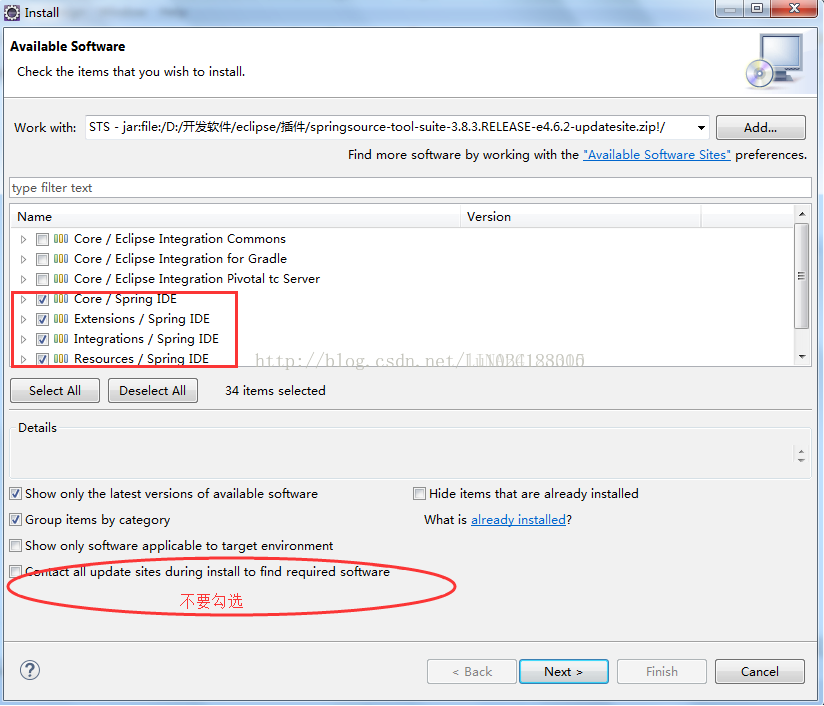
2.Eclipse中的Help->Install New Software



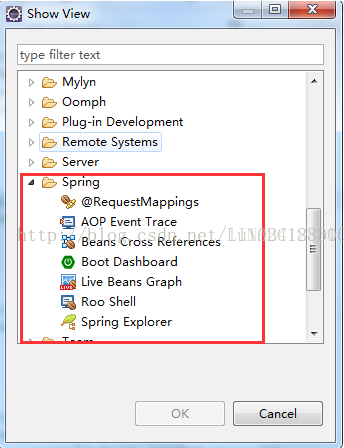
1. 点击"Add->Archive.."，然后选择刚才下载的STS插件，并给插件起个名字,点击"OK"就行了。



4.选择几个以IDE结尾的组件安装就可以了。

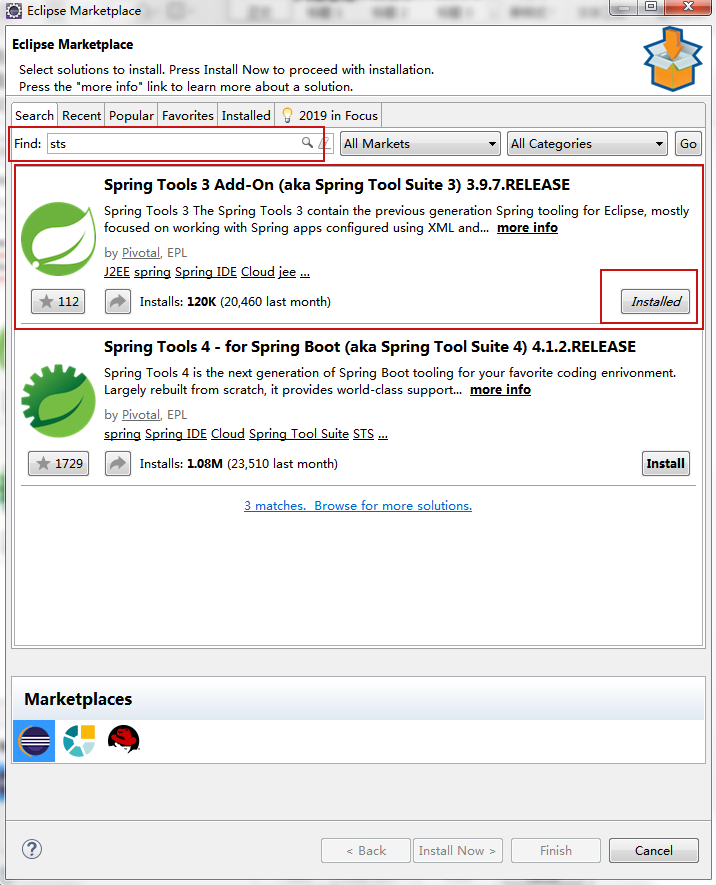


5.查看是否安装成功，使用



**安装STS插件方式二：**

**点击Eclipse上方的Help按钮，点击Eclipse Marketplace，弹出对话框并在搜索框中填写STS然后查询，如下图：**



然后点击Install(我这里已经安装过，所以是Installed)；根据指示点击按钮即可安装成功。

1. **StringBoot注解介绍，即jackson的基本演绎法：**

使用SpringBoot需要在pom.xml中加入以下依赖：

<dependency>

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

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

</dependency>

**1.@RestController**

使用SpringBoot中@RestController = SpringMVC中的@Controller + @ResponseBody（表示返回的是字符串或者json字符串）；

**2.@JsonFormat(pattern="yyyy-MM-dd hh:mm:ss",locale="zh",timezone="GMT+8")**

通过这个注解格式化日期，注解中的参数，第一个表示格式化的格式，第二个表示中国，第三个表示东八区；

**3.@JsonIgnore**

如果这个字段不需要返回或者需要忽略掉，可以使用该注解。

**4.@JsonInclude(Include.*NON\_NULL*)**

这个注解表示如果字段为空，则不返回，不为空则返回。

**三、SpringBoot热部署：**

1. **需要在pom.xml中加入一下依赖：**

<dependency>

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

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

<optional>true</optional>

</dependency>

devtools会监控我们项目下classPath下的class类，当我们的类发生变化，就会直接会重新编译那些类，但是不会加载第三方的jar包，还可以针对我们得页面进行热部署，而且不需要重启服务器；

optional=true 表示依赖不会传递本项目已经依赖好的devtools，如果以后依赖该boot项目并且想要使用devtools，就需要重新引入；

1. **需要在application.properties文件中加入一些配置，配置如下：**

spring.freemarker.cache = false 关闭缓存，实现页面即时刷新，针对于与页面的配置；

**3.以下是针对于服务端的配置；**

spring.devtools.restart.enabled = true 设置热部署生效

spring.devtools.restart.additional-paths=src/main/java 设置重启的目录，添加那哪个目录的文件需要重启；

spring.devtools.restart.exclude=static/\*\*,public/\*\* 排除哪个目录的文件不需要要重启；

spring.devtools.restart.exclude=WEB-INF/\*\* classpath目录下的WEB-INF文件夹内容修改不重启

**四、SpringBoot资源文件属性配置：**

1. **需要在pom.xml中加入以下依赖：**

<dependency>

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

<artifactId>spring-boot-configuration-processor</artifactId>

<optional>true</optional>

</dependency>

加入这些依赖可以让我们的代码读取资源文件中属性的值；

SpringBootDemoResource类中的属性必须对应resources.properties资源文件中的属性；

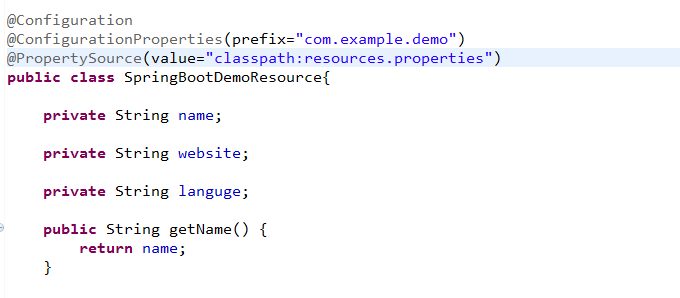
使用@Configuration注解代表@ConfigurationProperties注解和

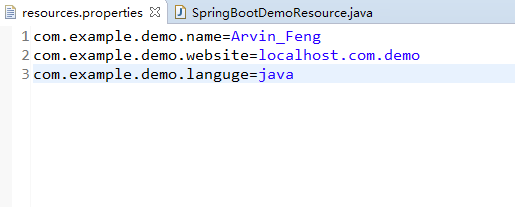
@PropertySource注解都是会引用资源文件的；

@ConfigurationProperties注解：@ConfigurationProperties(prefix="com.example.demo")其中refix的 值表示对应资源文件中属性的前缀；

@PropertySource注解：

@PropertySource(value="classpath:resources.properties")其中value的值表示我们的resources.properties资源文件的地址，我们得项目打包之后resource下的资源文件都会在classpath下面；





当我们这个SpringBootDemoResource类注入到Controller中的时候，SpringBootDemoResource类中的属性和resources.properties资源文件中com.example.demo前缀后面的属性进行一一对应的一个映射；

1. **SpringBoot整合模板引擎：**



**1.jsp和模板引擎的区别：**

jsp它的本质是一个serverlet，它的一些数据需要在我们后端进行一系列的渲染和处理，最终响应到我们得客户端进行展示，相对来说比较耗时，而模板引擎很多的渲染都是通过客户端来完成的，两者之间模板引擎的效率比jsp要好 ；

**2.SpringBoot整合freemarker：**

使用freemarker需要在pom.xml中加入以下依赖：

<dependency>

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

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

<optional>true</optional>

</dependency>

并且需要在资源文件中加入以下配置：

##############################################################

#

#freemarker 静态资源配置

#

##############################################################

#设置ftl文件路径

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

#关闭缓存，即时刷新，上线需要改为true

spring.freemarker.cache=false

#设置编码格式

spring.freemarker.charset=UTF-8

spring.freemarker.check-template-location=true

spring.freemarker.content-type=text/html

spring.freemarker.expose-request-attributes=true

spring.freemarker.expose-session-attributes=true

spring.freemarker.request-context-attribute=request

#设置文件后缀

spring.freemarker.suffix=.ftl

通过以下方式就可以访问到指定的页面，如下：

@RequestMapping("/Method\_6")

**public** String center() {

**return** "freemarker/center/center";#表示文件路径

}

还可以通过以下方式访问到指定页面，并且获取从后台返回的对象即属性，后台代码，ModelMap 类似于SpirngMVC中的ModelAndVeiow

@RequestMapping("/Method\_7")

**public** String index(ModelMap map) {

map.addAttribute("resource", resource);

**return** "freemarker/index";

}

页面可通过EL表达式获取对象中的属性值，前端代码如下：

<!DOCTYPE html>

<html>

<head lang="en">

<meta charset="UTF-8" />

<title></title>

</head>

<body>

FreeMarker模板引擎

<h1>${resource.name}</h1>

<h1>${resource.website}</h1>

<h1>${resource.languge}</h1>

</body>

</html>

**3.SpringBoot整合Thymeleaf：**

使用Thymeleaf需要在pom.xml中加入以下依赖：

<dependency>

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

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

</dependency>

并且需要在资源文件中加入以下配置：

###############################################################

#thymeleaf 静态资源配置

#

##############################################################

spring.thymeleaf.prefix=classpath:/templates/

#关闭缓存，即时刷新，上线改为true

spring.thymeleaf.cache=false

spring.thymeleaf.suffix=.html

spring.thymeleaf.mode=HTML5

spring.thymeleaf.encoding=UTF-8

spring.thymeleaf.content-type=text/html

通过以下方式就可以访问到指定的页面，如下：

@RequestMapping("/Method\_10")

**public** String center() {

**return** "thymeleaf/center/center";

}

还可以通过以下方式访问到指定页面，并且获取从后台返回的对象即属性，后台代码，ModelMap 类似于SpirngMVC中的ModelAndVeiow：

@RequestMapping("/Method\_8")

**public** String index(ModelMap map) {

map.addAttribute("projectName", "StringBoot-thymeleaf");

**return** "thymeleaf/index";

}

页面可通过Thymeleaf自带的标签获取后台返回的属性值，前端代码如下：

<!DOCTYPE html>

<html>

<head lang=*"en"*>

<meta charset=*"UTF-8"* />

<title></title>

</head>

<body>

Thymeleaf模板引擎

<h1 th:text=*"${projectName}"*>hello world~~~~~~~</h1>

</body>

</html>

1. **Thymeleaf中常用标签的使用方法：**
2. 基本使用方式：

后台代码：

SpringBootDemoPO po = **new** SpringBootDemoPO();

po.setProjectName("SpringBoot\_\_\_");

po.setProjectAuthor("Arvin\_Feng");

po.setProjectDate(**new** Date());

po.setProjectRemark("第一个SpringBoot项目例子，第四个方法");

po.setProjectRemark1("测试注解@JsonIgnore");

po.setProjectRemark2("<font color='green'><b>hello imooc</b></font>");

po.setAge(18);

map.addAttribute("user", po);

前台代码获取对象属性值：

<div>

项目名称：<input th:id=*"${user.projectName}"* th:name=*"${user.projectName}"* th:value=*"${user.projectName}"*/>

<br/>

项目作者：<input th:value=*"${user.projectAuthor}"*/>

<br/>

项目日期：<input th:value=*"${#dates.format(user.projectDate, 'yyyy-MM-dd')}"*/>

<br/>

项目备注：<input th:value=*"${user.projectRemark}"*/>

<br/>

项目备注1：<input th:value=*"${user.projectRemark1}"*/>

<br/>

项目备注2：<input th:value=*"${user.projectRemark2}"*/>

<br/>

</div>

1. 对象引用方式：

后台代码：

SpringBootDemoPO po = **new** SpringBootDemoPO();

po.setProjectName("SpringBoot\_\_\_");

po.setProjectAuthor("Arvin\_Feng");

po.setProjectDate(**new** Date());

po.setProjectRemark("第一个SpringBoot项目例子，第四个方法");

po.setProjectRemark1("测试注解@JsonIgnore");

po.setProjectRemark2("<font color='green'><b>hello imooc</b></font>");

po.setAge(18);

map.addAttribute("user", po);

前台代码通过对象获取属性值：

<div th:object=*"${user}"*>

项目名称：<input th:id=*"\*{projectName}"* th:name=*"\*{projectName}"* th:value=*"\*{projectName}"*/>

<br/>

项目作者：<input th:value=*"\*{projectAuthor}"*/>

<br/>

项目日期：<input th:value=*"\*{#dates.format(projectDate, 'yyyy-MM-dd')}"*/>

<br/>

项目备注：<input th:value=*"\*{projectRemark}"*/>

<br/>

项目备注1：<input th:value=*"${projectRemark1}"*/>

<br/>

项目备注2：<input th:value=*"${projectRemark2}"*/>

<br/>

</div>

1. 时间类型转换：

Forma（str,str1）,str表示某个具体的值，str1表示需要转换成的某种格式；

项目日期：<input th:value=*"${#dates.format(user.projectDate, 'yyyy-MM-dd')}"*/>

1. Text与UText:

后台代码：

po1.setProjectRemark2("<font color='green'><b>hello imooc</b></font>");

前端代码：

text 与 utext ：<br/>

使用text：<span th:text=*"${user.projectRemark2}"*>abc</span>

<br/>

使用utext：<span th:utext=*"${user.projectRemark2}"*>abc</span>

<br/>

使用text不会把html代码解析出来，而是把html代码当做一个字符串呈现出来，但是utext会把html代码解析并且呈现出来；

1. URL:

使用方式：<a href=*""* th:href=*"@{http://www.baidu.com}"*>度娘</a>

1. 引入静态资源文件js/css:

需要在application.properties资源文件中加入一段配置，配置如下：

#设置静态文件路径，js，css等

spring.mvc.static-path-pattern=/static/\*\*

例子：

<script th:src="@{/static/js/thymeleafTest.js}"></script>

1. 条件判断th:if：

例子：

<div th:if=*"${user.age} == 18"*>等于18岁</div>

<div th:if=*"${user.age} gt 18"*>大于18岁</div>

<div th:if=*"${user.age} lt 18"*>小于18岁</div>

<div th:if=*"${user.age} ge 18"*>大于等于18岁</div>

<div th:if=*"${user.age} le 18"*>小于等于18岁</div>

1. Th:select选择框

<select>

<option >选择框</option>

<option th:selected=*"${user.projectAuthor eq 'lee'}"*>lee</option>

<option th:selected=*"${user.projectAuthor eq 'imooc'}"*>imooc</option>

<option th:selected=*"${user.projectAuthor eq 'LeeCX'}"*>LeeCX</option>

<option th:selected=*"${user.projectAuthor eq 'Arvin\_Feng'}"*>Arvin\_Feng</option>

</select>

1. 循环th:each:

<table>

<tr>

<th>姓名</th>

<th>年龄</th>

<th>年龄备注</th>

<th>生日</th>

</tr>

<tr th:each=*"person:${userList}"*>

<td th:text=*"${person.projectAuthor}"*></td>

<td th:text=*"${person.age}"*></td>

<td th:text=*"${person.age gt 18} ? 你老了 : 你很年轻"*>18岁</td>

<td th:text=*"${#dates.format(user.projectDate, 'yyyy-MM-dd hh:mm:ss')}"*></td>

</tr>

</table>

1. th:switch与th:case:

<div th:switch=*"${user.projectName}"*>

<p th:case=*"'lee'"*>lee</p>

<p th:case=*"#{roles.manager}"*>普通管理员</p>

<p th:case=*"#{roles.superadmin}"*>超级管理员</p>

<p th:case=*"\*"*>其他用户</p>

</div>

1. **Spring Boot与Mybatis整合：**
2. 使用generatorConfig生成Mappper以及pojo（实体类）；
3. 、在pom.xml中引入一下依赖：

<!--mybatis-->

<dependency>

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

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

<version>1.3.1</version>

</dependency>

<!--mapper-->

<dependency>

<groupId>tk.mybatis</groupId>

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

<version>1.2.4</version>

</dependency>

<dependency>

<groupId>com.alibaba</groupId>

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

<version>1.1.9</version>

</dependency>

<dependency>

<groupId>org.mybatis.generator</groupId>

<artifactId>mybatis-generator-core</artifactId>

<version>1.3.2</version>

<scope>compile</scope>

<optional>true</optional>

</dependency>

1. 、在application.properties资源文件中加入Mibatis热部署配置，如下：

#为mybatis设置热部署，生产环境可删除

restart.include.mapper=/mapper-[\\w-\\.]+jar

restart.include.pagehelper=/pagehelper-[\\w-\\.]+jar

1. 、在application.properties资源文件中加入数据源和 Mibatis配置，如下：

############################################################

# 配置数据源，

############################################################

spring.datasource.url=jdbc:mysql://10.205.138.135:8066/usermanager?useUnicode=true&amp;characterEncoding=UTF-8

spring.datasource.username=usermanager

spring.datasource.password=123456

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

spring.datasource.druid.initial-size=1

spring.datasource.druid.min-idle=1

spring.datasource.druid.max-active=20

spring.datasource.druid.test-on-borrow=true

spring.datasource.druid.stat-view-servlet.allow=true

############################################################

# mybatis 配置

############################################################

# mybatis 配置

#表示生成的PO在com.example.demo.po这个路径下面

mybatis.type-aliases-package=com.example.demo.po

#表示生成的mapper.mxl文件在classpath下的mapper文件下

mybatis.mapper-locations=classpath:mapper/\*.xml

# 通用Mapper配置

#表示指向通用Mapper的Mapper类（完整路径）

mapper.mappers=com.example.demo.util.MyMapper

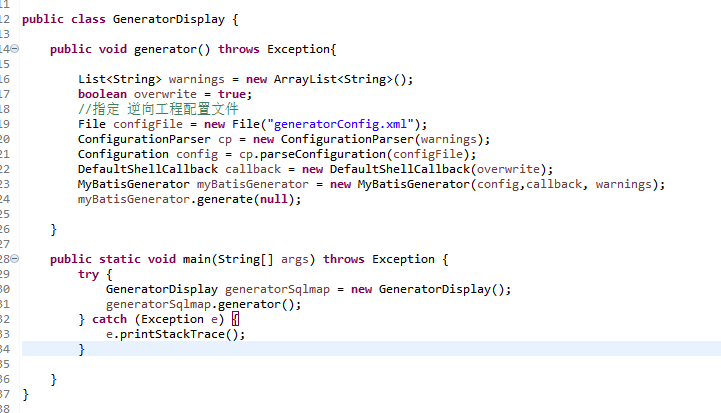
mapper.not-empty=false

mapper.identity=MYSQL

1. 、在项目根目录下面创建Mibatis配置文件“generatorConfig.xml”，内容如下：



1. 、运行GeneratorDisplay这个类或者运行这个类中的main方法，即可生成，内容如下：



1. 实现整合Mybatis的CRUD功能；
2. 、首先需要在启动类中加入一下几个注解：

//Spring Boot 自动扫描 mybatis mapper 包路径下的类

@MapperScan(basePackages = "com.example.demo.mapper")

//扫描 所有需要的包, 包含一些自用的工具类包 所在的路径

@ComponentScan(basePackages= {"com.example.demo"})

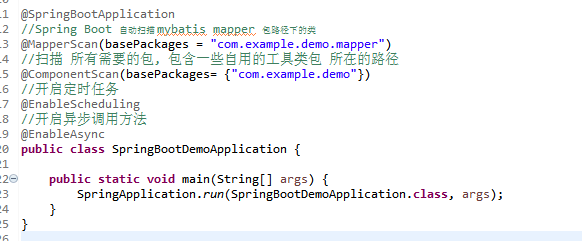
//开启定时任务

@EnableScheduling

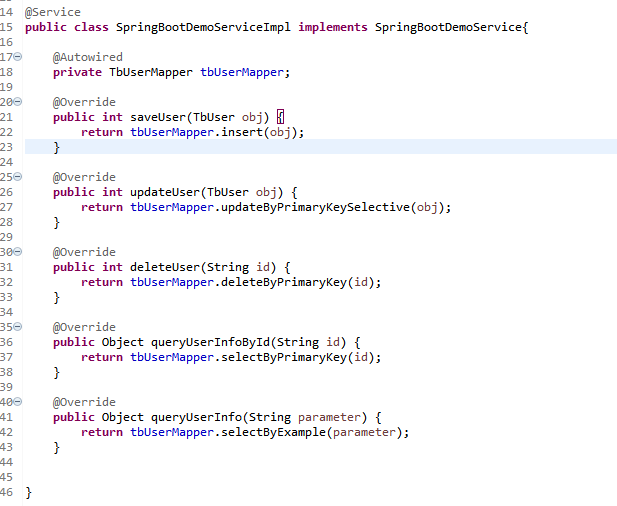
//开启异步调用方法

@EnableAsync

使用方法，如下图：



对于持久层操作是，直接使用项目生成的Mapper类即可，使用生成的Mapper类对象就可以完成CRUD四大功能，只需在Service实现类中注入新生成的Mapper类即可使用此类的对象如下图：



上图中TbUserMapper类就是通过generatorConfig生成的Mapper类；Controller和Service和Spring写法一致。

1. 实现整合Mybatis-pagehelper的分页功能；
2. 、需要在pom.xml文件中加入以下pagehelper的相关依赖，如下：

<!--pagehelper-->

<dependency>

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

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

<version>1.2.3</version>

</dependency>

1. 、需要在application.properties资源文件中加入以下pagehelper的相关配置，如下：

# 分页插件配置

pagehelper.helperDialect=mysql

pagehelper.reasonable=true

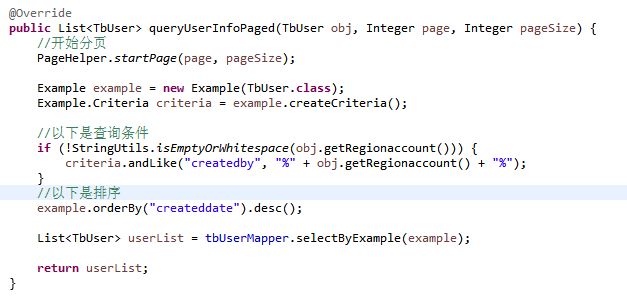
pagehelper.supportMethodsArguments=true

pagehelper.params=count=countSql

//开始分页，只需要在Service实现类中加入以下代码，page参数表示第几页，pageSize表示每页查询多少条，即可实现分页；原理是 PageHelper实质是拦截到了SQL语句，然后针对SQL进行封装，来达到分页的效果。

PageHelper.*startPage*(page, pageSize);

使用方法，如下图：

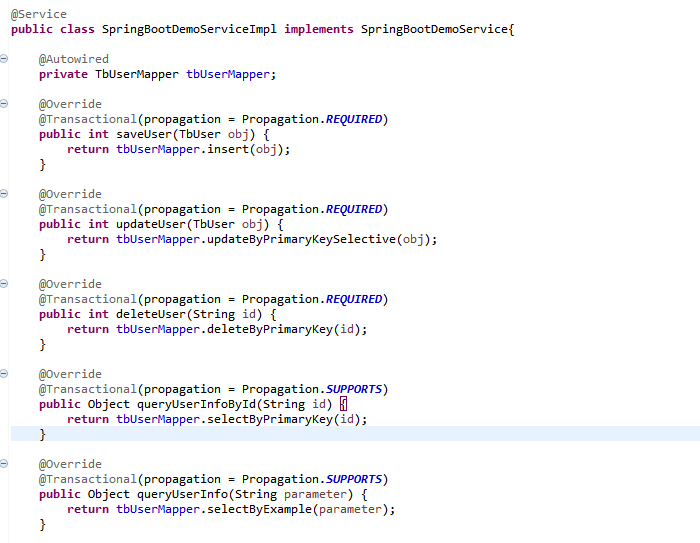


1. 实现自定义Mapper；

与Mybatis用法一致；

1. 引入事物执行回滚；

使用方法如下：



事物的隔离级别主要使用DEFAULT默认的

@Transactional(propagation = Propagation.***REQUIRED***)

事物的传播行为主要是用 REQUIRED SUPPORTS

@Transactional(propagation = Propagation.***SUPPORTS***)

REQUIRED 表示当一个操作需要执行的时候，如果有事务，那么就会把这个操作加入到事务中并且执行，如果没有事物，就会新起一个事务并且去执行；通常在新增、更新、删除的时候会使用REQUIRED ；

SUPPORTS 表示当一个操作需要执行的时候，如果有事务，那么就会把这个操作加入到事务中并且执行，如果没有事物，那么就会脱离事务去执行；通常在查询的时候会使用SUPPORTS ；

1. **Spring Boot整合 Redis：**
2. 在pom.xml中引入Redis的相关依赖；

<!-- 引入 redis 依赖 -->

<dependency>

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

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

</dependency>

1. 在application.properties资源文件中加入Redis相关配置；

############################################################

# Redis相关配置

############################################################

# Redis数据库索引（默认为0）

spring.redis.database=0

# Redis服务器地址

spring.redis.host=127.0.0.1

# Redis服务器连接端口

spring.redis.port=6379

# Redis服务器连接密码（默认为空）

spring.redis.password=

# 连接池最大连接数（使用负数表示没有限制）

spring.redis.pool.max-active=1000

# 连接池最大阻塞等待时间（使用负数表示没有限制）

spring.redis.pool.max-wait=-1

# 连接池中最大空间键连接数

spring.redis.pool.max-idle=10

# 连接池中最小空间键连接数

spring.redis.pool.min-idle=2

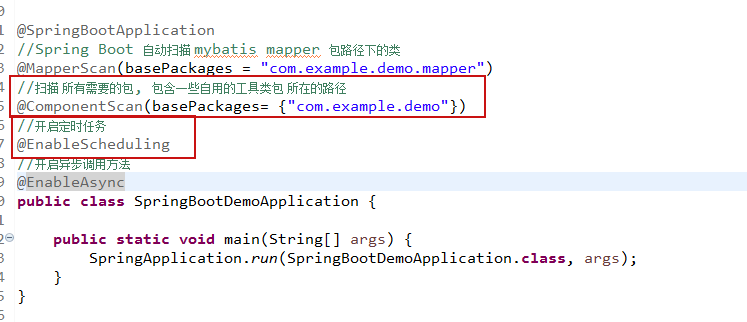
# 连接超时时间（毫秒）

spring.redis.timeout=10000

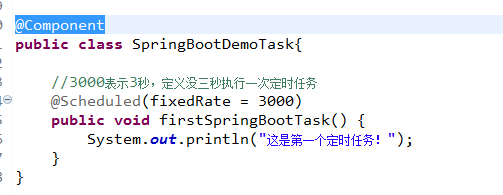
1. 使用方法如下；



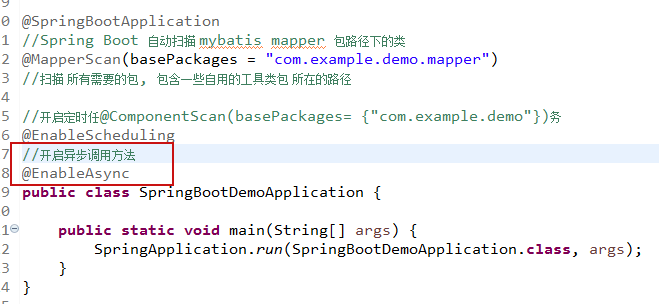
1. **Spring Boot整合定时任务：**
2. 使用注解在StringBoot中的启动类中加入@EnableScheduling 注解，并且加上@ComponentScan(basePackages= {"com.example.demo"})注解（表示需要扫描的包路径），便可以实现全自动扫描。



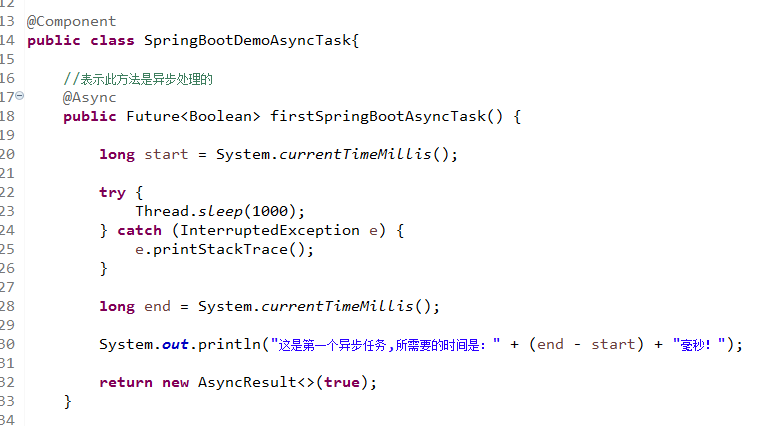
1. 在定时任务的类上面加一个@Component注解表示此类是一个组件，以至于可以被扫描到。



1. 定时任务表达式生成地址：<http://cron.qqe2.com/>
2. **Spring Boot整合异步任务以及使用场景：**
3. 使用注解在StringBoot中的启动类中加入@EnableAsync注解，便可以实现自动扫描。



1. 在定时任务的类上面加一个@Component注解表示此类是一个组件，以至于可以被扫描到，然后再方法上加入@Async注解（表示此方法是异步的），去掉@Async注解则表示同步。



1. 异步任务使用场景；
2. 、发送短信
3. 发送邮件
4. App消息推送
5. 节省运维凌晨发布任务时间提供效率
6. **Spring Boot如何使用拦截器：**

1.在类上加上@Configuration注解并且需要继承WebMvcConfigurerAdapter类，所以此类表明当前类是一个配置类，可以配置拦截器，等其他组件；

2.重写addInterceptors添加需要的拦截器地址，然后通过对这些拦截器进行注册就可以使用。

3.配置类写法：

