# 引言

## Spring 传统项目分析

在传统的spring应用项目中：

* 配置相对复杂（例如大量xml）。
* 依赖冲突严重（maven依赖版本不同）。
* 扩展相对复杂 (尤其是分布式部署，微服务架构)。

思考：如何让初级程序员从复杂的配置中解脱出来呢？

## Spring 家族技术发展

Spring诞生于2002年，成型于2003，目前已经发展到了Spring5.x版本，支持JDK-11及JavaEE8。现在的spring已经不仅仅是framework，还扩展了spring boot，spring cloud等。

说明:学技术一定要了解技术发展史，并对技术的发展有一定的前瞻性。

# Spring Boot 简介

## Spring Boot 概述

现在软件市场已经形成一定的规模，系统从单体架构，分布式架构，再到微服务架构，其整个架构体系正在发生很大变化，企业现在更注重技术的开箱即用，更注重技术在生态圈中的深度融合，更注重轻量级的运维。由此spring boot诞生。

## Spring Boot 核心特性

Spring boot构建于Spring 框架基础之上，基于快速构建理念，提供了自动配置功能，可实现其开箱即用特性，可以简化整个项目的配置。其核心主要包括如下几个方面：

* 自动配置(Auto Configuration)。
* 起步依赖(Starter Dependency)。
* 健康检查(Actator)。

其中，Spring Boot官方地址为<https://spring.io/projects/spring-boot>。

# Spring Boot 快速入门

## 初始环境配置

### 准备工作

* 工具下载

1. 下载JDK1.8，并进行环境变量配置。
2. 下载maven(例如apache-maven-3.6.1)，并进行相关配置(本地库，私服)。
3. 下载sts最新版(例如sts-4.3.2.RELEASE)。

* 工作区准备

1. 定义新的工作区，例如d:/workspacests。

### Maven基本配置

打开maven中的setting.xml文件，并对其如下选项进行配置。

* 配置maven本地库，

<localRepository>${user.home}/maven/repository</localRepository>

* 配置maven私服。

<mirror>

<id>aliyun</id>

<name>aliyun Maven</name>

<mirrorOf>\*</mirrorOf>

<url>https://maven.aliyun.com/repository/central</url>

</mirror>

* 配置maven中的profile。

<profile>

<id>jdk-1.8</id>

<activation>

<activeByDefault>true</activeByDefault>

<jdk>1.8</jdk>

</activation>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<maven.compiler.compilerVersion>1.8</maven.compiler.compilerVersion>

</properties>

</profile>

### STS整合maven配置

* Maven Installations配置， 如图-1所示：

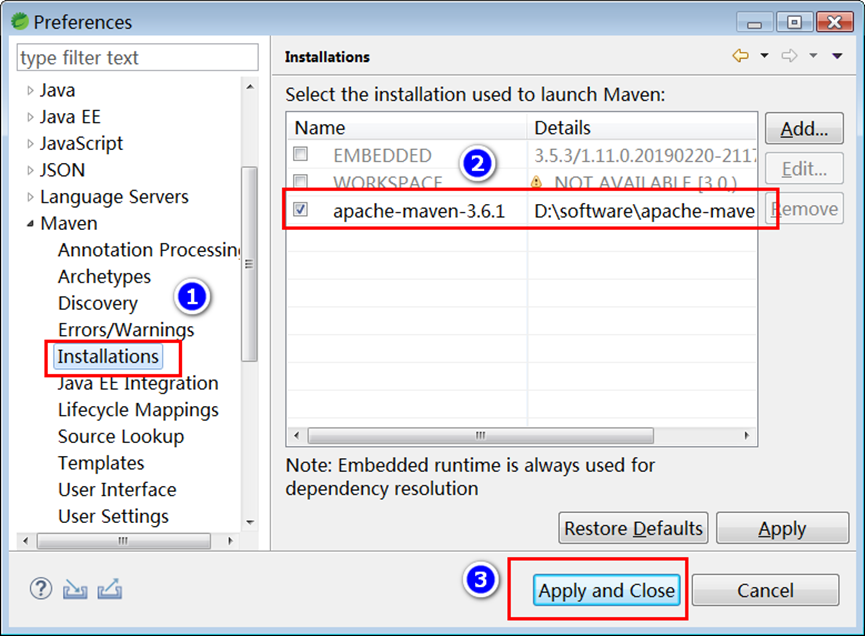


图-1

* Maven User Settings配置，如图-2所示：

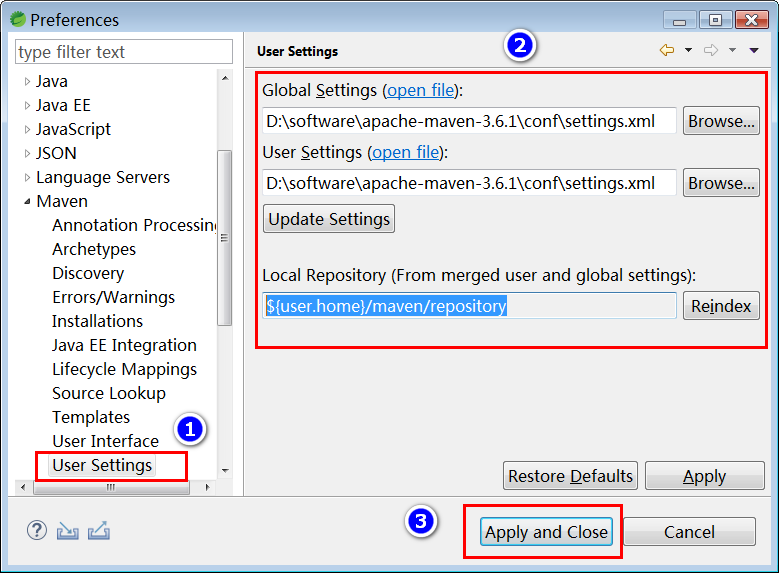


图-2

* 项目工作区编码设置，如图-3所示

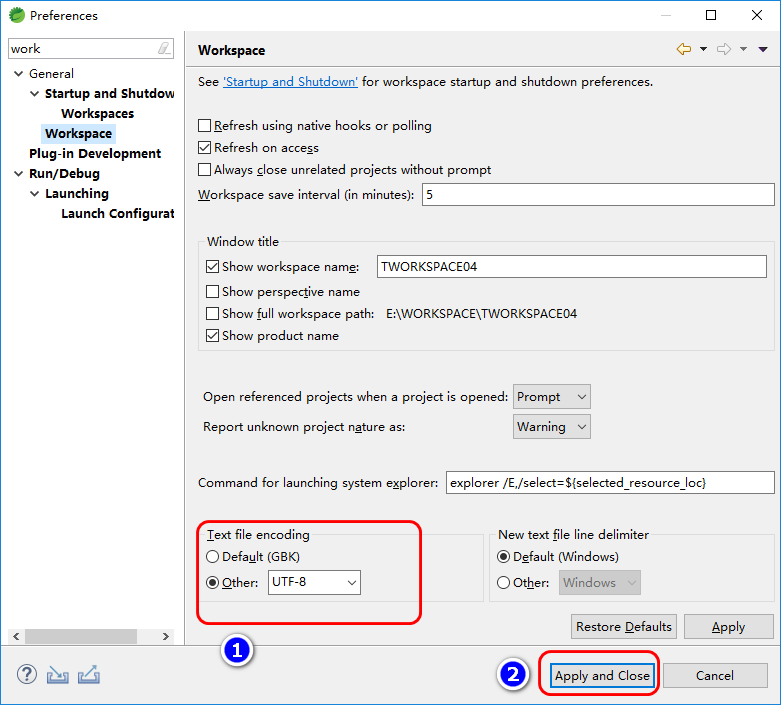


图-3

### STS工具使用基本优化

* 禁用拼写检查，取消图-4所示的拼写检查选项。

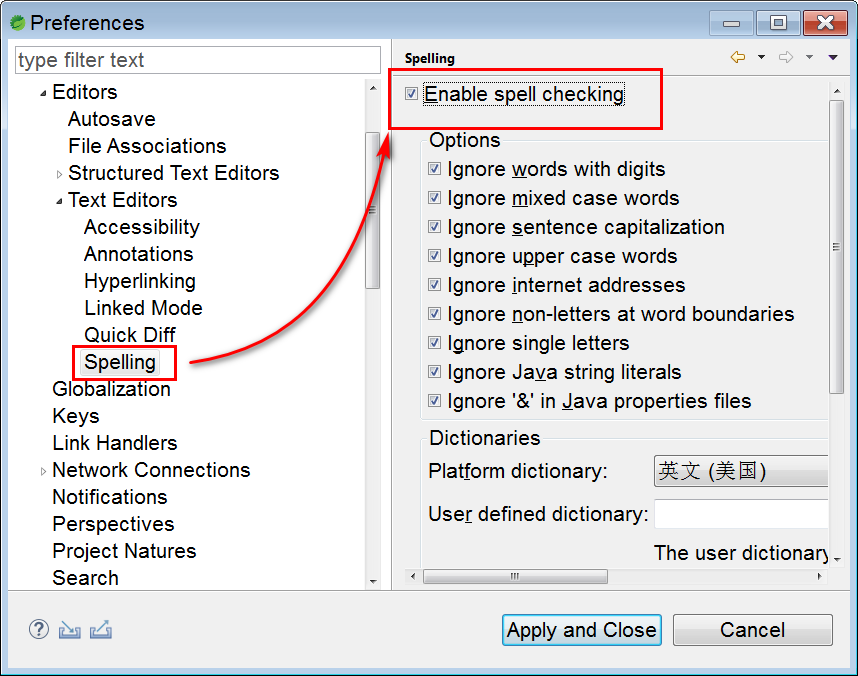


图-4

* 取消图-5所示的连接选项配置。

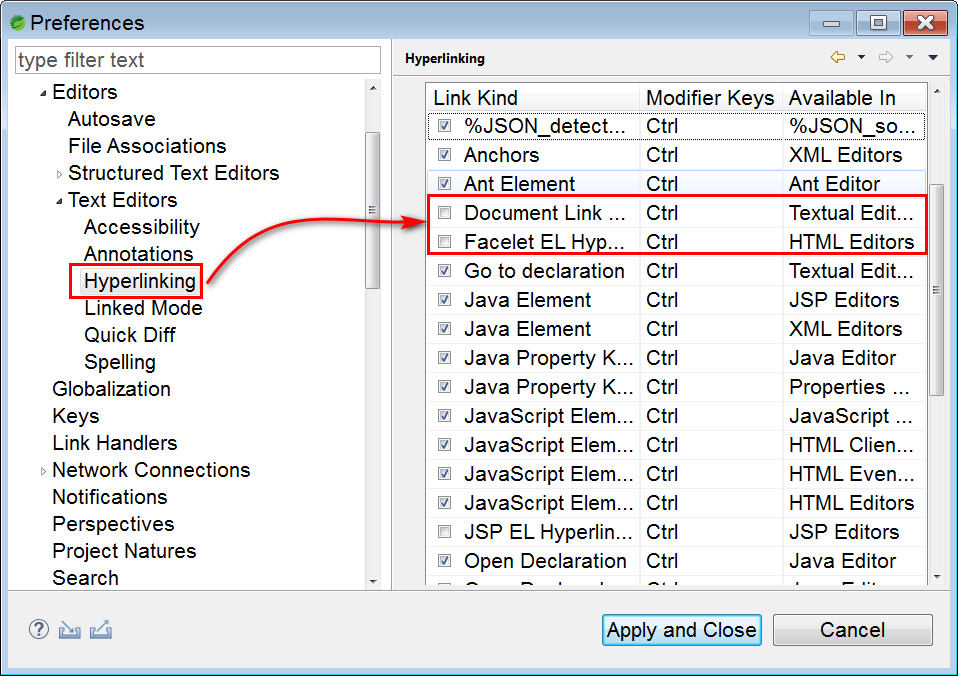


图-5

* 关掉图-6的校验功能。

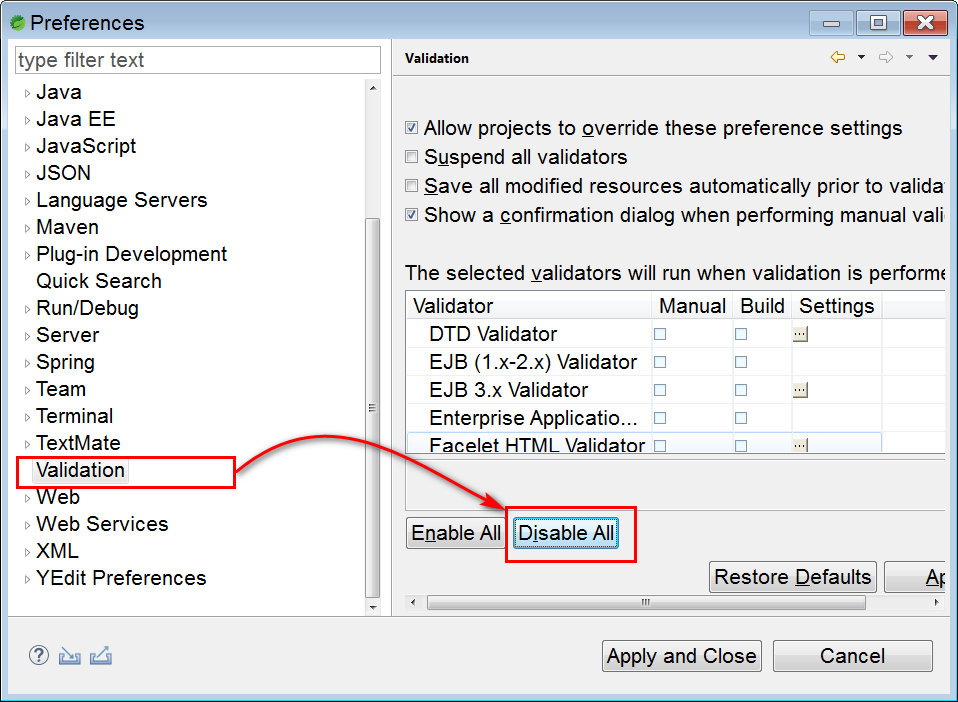


图-6

* 修改STS工具内存配置，打开SpringToolSuite4.ini文件，修改堆大小，如图-7所示：

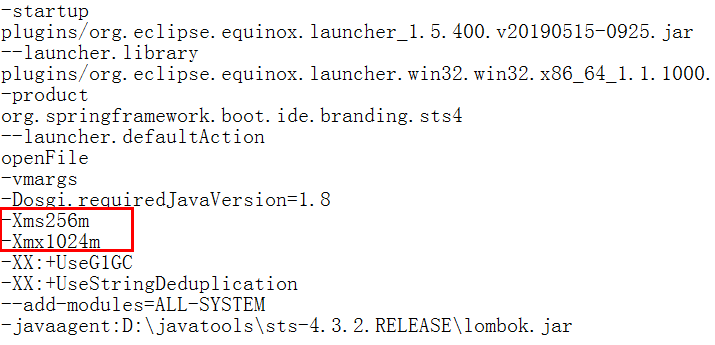


图-7

## 项目创建及结构分析

基于STS 集成开发工具，创建spring boot项目，其具体步骤如下：

第一步：打开项目新建窗口（快捷键 ctrl+n）, 搜索spring，选择spring starter 项目。如图-8所示：

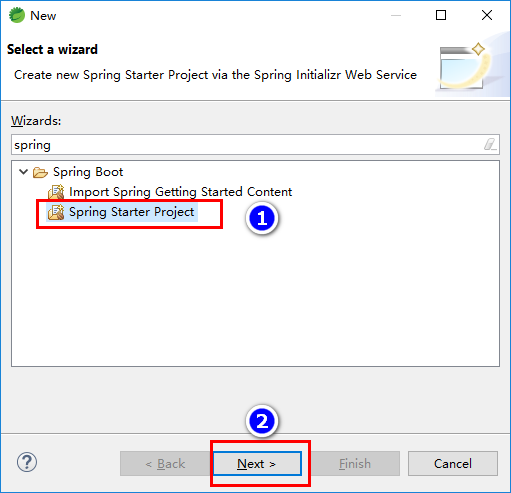


图-8

第二步：填写项目基本信息。如图-9所示：

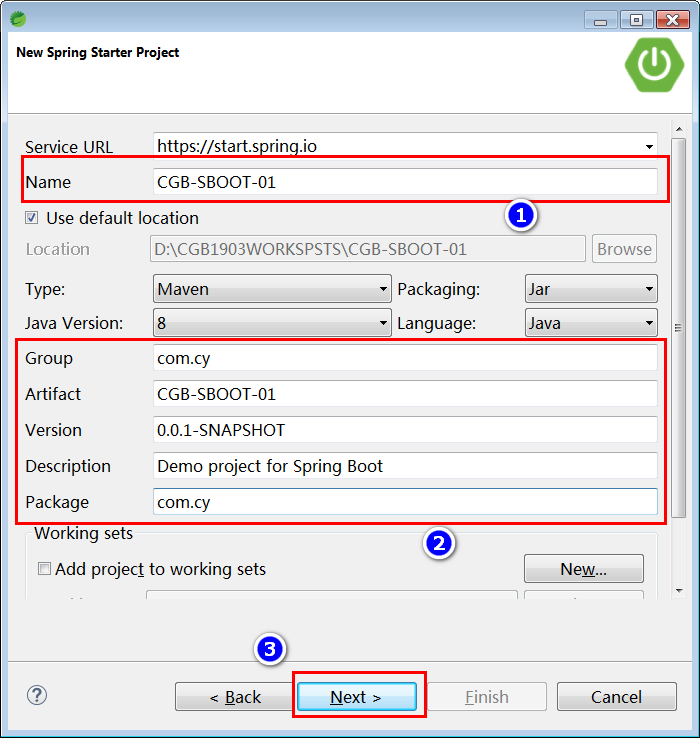


图-9

第三步：选择Spring Boot版本，如图-10所示：

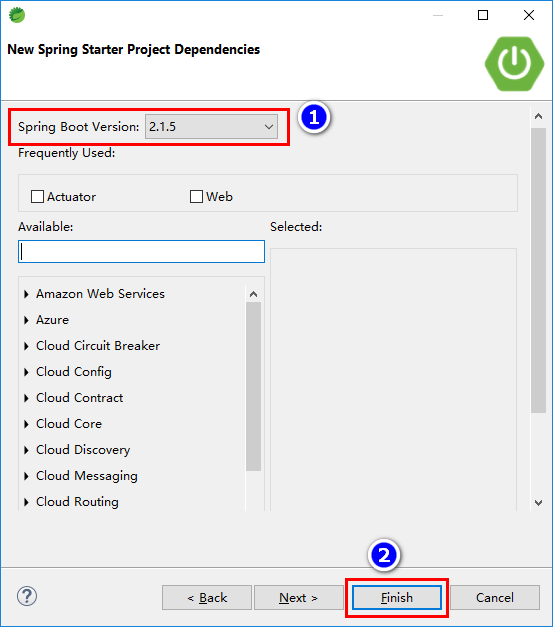


图-10

## 项目业务实现及测试

业务描述：创建一个DefaultCache类然后交给spring 管理。

**package** com.cy.pj.common.cache;

@Component

@Scope("singleton")

**public** **class** DefaultCache {

**public** Cache() {

System.***out***.println("cache()");

}

@PostConstruct

**public** **void** init() {

System.***out***.println("init()");

}

@PreDestroy

**public** **void** destory() {

System.***out***.println("destory");

}

}

添加sringboot 测试类，进行bean的获取及测试。

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** DefaultCacheTests {

@Autowired

**private** DefaultCache defaultCache;

@Test

**public** **void** testCache() {

System.out.println(defaultCache);

}

}

其中:

* @RunWith注解用于指定Junit单元测试框架的启动运行类。
* SpringBootTest 注解用于告诉spring框架,此测试类交给spring管理。

# Spring Boot 整合数据源

## 单数据源整合实现

### 使用默认的HikariCP

第一步：添加依赖。

编辑项目中pom.xml,右键项目的pom.xml文件,选择spring,如图-11所示:

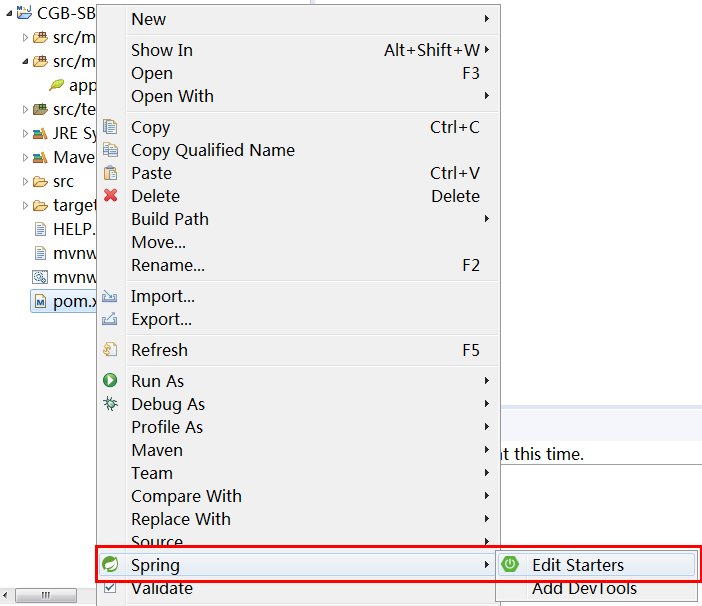


图-11

查找mysql 驱动依赖，JDBC API依赖，如图-12所示：

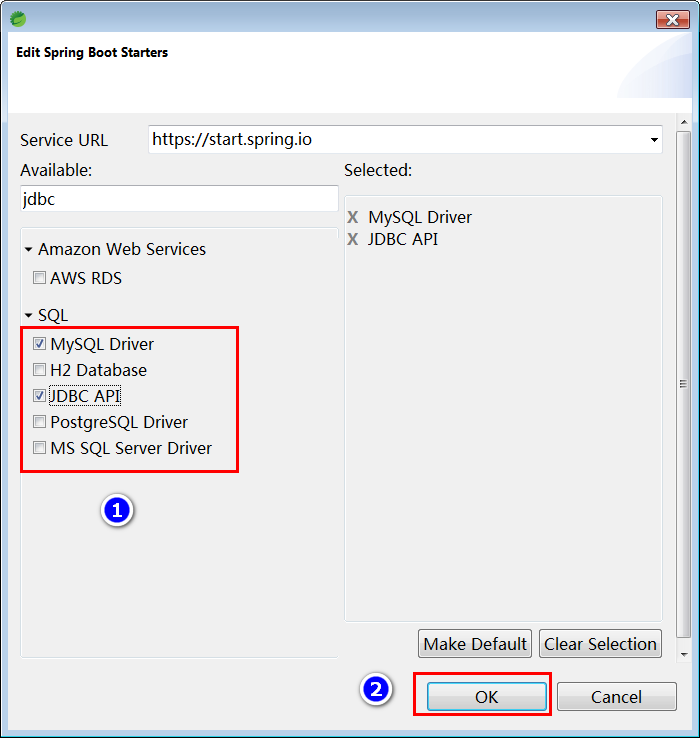


图-12

依赖添加以后,在pom.xml文件中会自动添加如下两个依赖配置：

1. mysql数据库驱动依赖。

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

1. spring对象jdbc支持。

<dependency>

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

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

</dependency>

第二步：配置连接池

打开application.properties配置文件，添加如下内容。

spring.datasource.url=jdbc:mysql:///dbgoods?serverTimezone=GMT%2B8

spring.datasource.username=root

spring.datasource.password=root

第三步：单元测试

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** DataSourceTests {

@Autowired

**private** DataSource dataSource;

@Test

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

System.out.println(dataSource.getConnection());

}

}

### 整合Druid对象

第一步：添加druid启动依赖

<dependency>

<groupId>com.alibaba</groupId>

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

<version>1.1.13</version>

</dependency>

第二步:配置数据源

打开application.properties配置文件，修改连接池内容配置。

spring.datasource.type=com.alibaba.druid.pool.DruidDataSource

spring.datasource.url=jdbc:mysql:///dbgoods?characterEncoding=utf-8&serverTimezone=GMT

spring.datasource.username=root

spring.datasource.password=root

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

第三步：简化写法，可以省略驱动

spring.datasource.url=jdbc:mysql:///dbgoods? characterEncoding=utf-8&serverTimezone=GMT

spring.datasource.username=root

spring.datasource.password=root

第四步:单元测试

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** DataSourceTests {

@Autowired

**private** DataSource dataSource;

@Test

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

System.out.println(dataSource.getConnection());

}

}

## 多数据源整合实现（扩展）

本小节作为课后作业。

# Spring Boot 整合MyBatis

## 初始配置

### 添加mybatis启动依赖

参考官网 mybatis.org/spring

<dependency>

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

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

<version>2.0.1</version>

</dependency>

### Mybatis简易配置

在spring boot的配置文件application.properties文件中添加如下内容

mybatis.configuration.default-statement-timeout=30

mybatis.configuration.map-underscore-to-camel-case=true

logging.level.com.cy=DEBUG

其中，logging.level.com.cy=DEBUG 表示要对com.cy包中的DEBUG操作日志做输出。

## 业务分析及实现

### 基本业务实现及单元测试

定义商品业务数据层接口及业务方法。

**package** com.cy.pj.goods.dao;

**import** org.apache.ibatis.annotations.Delete;

**import** org.apache.ibatis.annotations.Mapper;

@Mapper //3.4

**public** **interface** GoodsDao {

@Delete("delete from tb\_goods where id=#{id}")

**int** deleteObject(Integer id);

}

其中@Mapper由MyBatis框架中定义的一个描述Mapper接口的的注解,用于告诉Spring框架此接口由底层创建其实现类对象,并存储到spring容器.

定义测试类

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** GoodsDaoTests {

@Autowired

**private** GoodsDao goodsDao;

@Test

**public** **void** testGoodsDao() {

**int** rows=goodsDao.deleteObject(10);

System.out.println("rows="+rows);

}

}

### 拓展业务分析及实现

Step01:在GoodsDao中定义添加如下方法：

**int** deleteObjects(

@Param("ids")Integer... ids);

说明:当接口方法对应的映射语句相对比较复杂时,建议将映射语句写到对应映射文件。

Step02:在src/main/resources目录下创建mapper目录,然后在其目录添加。

GoodsMapper.xml,并添加如下内容:

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

<!DOCTYPE mapper

PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"

"http://mybatis.org/dtd/mybatis-3-mapper.dtd">

<mapper namespace=*"com.cy.pj.goods.dao.GoodsDao"*>

<delete id=*"deleteObjects"*>

delete from tb\_goods

where id in <!-- (1,2,3,4,5) -->

<foreach collection=*"ids"*

open=*"("*

close=*")"*

separator=*","*

item=*"id"*>

#{id}

</foreach>

</delete>

</mapper>

Step03:在application.properties文件中添加如下配置：

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

Step04:在GoodsDaoTests类中添加如下单元测试方法进行单元测试:

@Test

**public** **void** testDeleteObjects() {

**int** rows=goodsDao.deleteObjects(17,18);

System.***out***.println(rows);

}

# Spring Boot 整合WEB应用

## 初始配置

### 添加项目依赖

编辑pom.xml文件,添加web依赖，Thymeleaf依赖，代码如下：

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

其中： Spring Web Starter 提供Spring MVC 依赖支持，thymeleaf一个html模板引擎，提供了与Spring MVC进行整合的API，可作为MVC架构中Web应用的View层。

### Spring Web简易配置

在application.properties文件中添加视图解析器。

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

spring.thymeleaf.suffix=.html

## 业务分析及实现

编写GoodsController类并将其交给spring管理。

**package** com.cy.pj.sys.controller;

@Controller

@RequestMapping("/goods/")

**public** **class** GoodsController {

@RequestMapping("doDeleteGoods")

@ResponBody

**public** String doDeleteGoods() {

**return** "delete ok";

}

@RequestMapping("doGoodsUI")

**public** String doGoodsUI() {

**return** "googds";

}

}

说明：需要在/templates/pages目录下创建goods.html,然后启动项目进行访问测试实现。

## 服务器简易配置实现

### 内嵌服务器基本配置

打开application.properties文件,修改server端口等。

server.port=80

server.tomcat.uri-encoding=UTF-8

server.tomcat.max-threads=1000

server.tomcat.min-spare-threads=30

server.servlet.context-path=/

其中:

* server.port 表示服务器启动端口。
* server.tomcat.uri-encoding=UTF-8表示GET请求编码
* server.tomcat.max-threads=1000 表示最大线程数。
* server.tomcat.min-spare-threads=30 表示最小备用线程数。
* server.servlet.context-path=/ 表示context路径配置。

### 使用Jetty替换tomcat

SpringBoot 默认使用Tomcat作为内嵌Servlet容器，可查看Spring-boot-starter-web依赖。假如需要使用其它的Web服务器，只需修改spring-boot-starter-web依赖即可。具体过程实现如下：

取消内嵌的tomcat依赖：

<dependency>

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

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

<exclusions>

<exclusion>

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

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

</exclusion>

</exclusions>

</dependency>

添加jetty依赖，Jetty也是一个类似tomcat的web服务器,假如spring boot项目中使用Jetty服务器可以做如下依赖配置。

<dependency>

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

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

</dependency>

启动服务器进行测试，检测日志输出。

# Spring Boot 项目应用扩展

## 健康检查配置及测试

在项目中添加健康检查依赖:

<dependency>

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

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

</dependency>

健康检查分析：在浏览器中输入如下地址：

<http://localhost/actuator/health>

假如希望查看更多actuator选项，可以在spring boot中配置文件application.properties中添加如下语句:

management.endpoints.web.exposure.include=\*

此时在浏览器地址栏可以输入.<http://localhost/actuator/beans> 查看所有的

spring 容器中的bean信息。

## 热部署配置及实现

Springboot web项目修改了某个类以后,默认不会自动重新部署加载,需要手动重启服务器,假如需要自动部署,可以添加如下依赖:

<dependency>

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

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

<scope>runtime</scope>

</dependency>

说明:此依赖可通过STS 工具中的Edit starters方式进行添加。

## Lombok插件应用

Lombok是一个第三库，可以lombok中提供的API可以在程序编译时自动织入一些属性或相关方法，例如省略pojo类中set/get/toString等方法的编写,我们写的java程序可以在编译时，自动将这些方法添加到.class文件中，其应用步骤如下：

第一步：添加依赖。

<dependency>

<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

第二步:根据pom.xml文件中的配置，找到lombok文件夹下的lombok-xxx.jar文件的具体位置。

第三步:启动操作系统命令行客户端，并切换到lombok-xxx.jar文件所在的目录，然后，利用java –jar 指令启动lombok插件，如图-13所示：

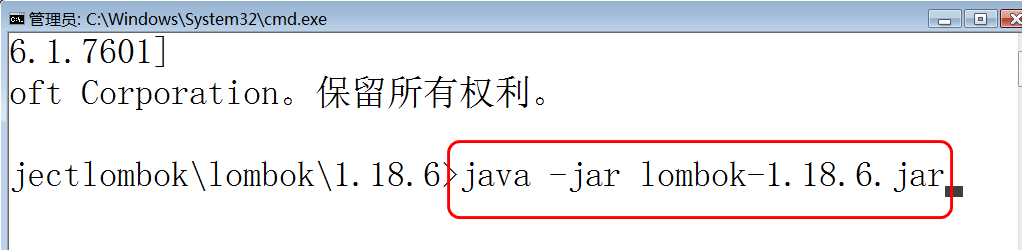


图-13

回车执行如上命令，会弹出选择安装对话框，如图-14所示：



图-14

然后在弹出的框中选择sts启动执行文件(SpringToolSuite4.exe)进行安装,

安装成功以后会出现如下弹出框，如图-15所示：

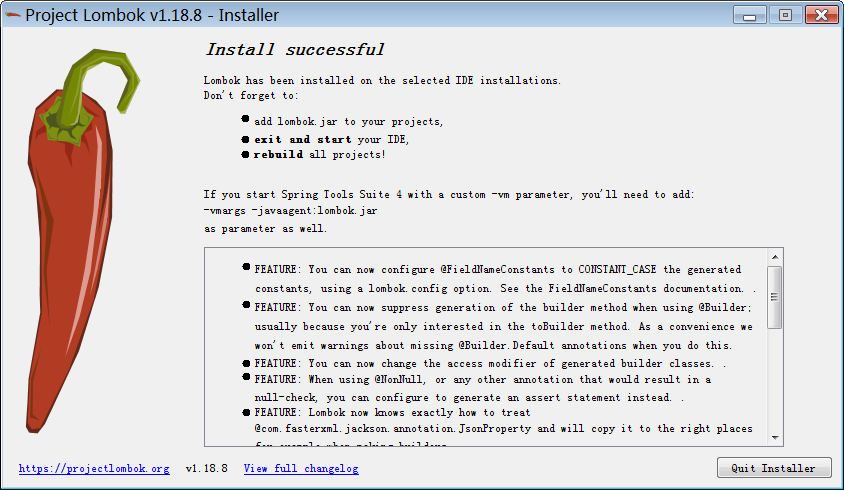


图-15

安装成功以后可在STS工具的根目录创建一个lombok.jar文件,同时会在SpringToolSuite4.ini文件的尾部添加lombok.jar文件所在的具体路径，例如

-javaagent:D:\software\sts-4.2.2.RELEASE\lombok.jar。

其中lombok.jar会默认创建在sts软件的根目录，如图-16所示：

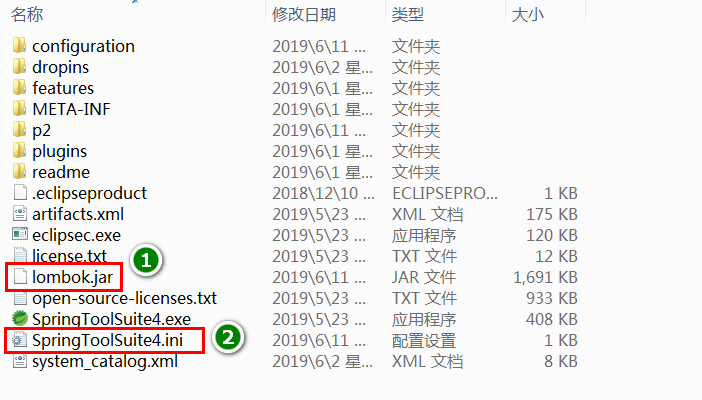


图-16

说明:安装成功以后重启sts 工具.(假如安装成功以后再次启动sts,此时无法启动,可能与STS所在目录有关系,可以将安装目录下的lombok.jar文件拷出到特定目录,然后修改SpringToolSuite4.ini文件的最后一行的lombok.jar路径即可)。

Lombok 常用注解分析：

* + **@Setter** 注解在类或字段，注解在类时为所有字段生成setter方法，注解在字段上时只为该字段生成setter方法。
  + **@Getter** 使用方法同上，区别在于生成的是getter方法。
  + **@ToString** 注解在类，添加toString方法。
  + **@EqualsAndHashCode** 注解在类，生成hashCode和equals方法。
  + **@NoArgsConstructor** 注解在类，生成无参的构造方法。
  + **@RequiredArgsConstructor** 注解在类，为类中需要特殊处理的字段生成构造方法，比如final和被@NonNull注解的字段。
  + **@AllArgsConstructor** 注解在类，生成包含类中所有字段的构造方法。
  + **@Data** 注解在类，生成setter/getter、equals、canEqual、hashCode、toString方法，如为final属性，则不会为该属性生成setter方法。
  + **@Slf4j** 注解在类，生成log变量，严格意义来说是常量。private static final Logger log = LoggerFactory.getLogger(UserController.class);

**代码案例分析：**

@Slf4j

@Data //setter,getter,toString()

@Accessors(chain = **true**)

@NoArgsConstructor

@AllArgsConstructor

**public** **class** Goods **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 6239917530570596544L;

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

**private** Long id;;

**private** String name;

**private** String remark;

**private** Date createdTime;

**public** String doPrint() {

***log***.info("goods.display()");

**return** toString();

}

}

测试类：

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** LombokTests {

@Test

**public** **void** testGoods() {

Goods g=**new** Goods();

g.setId(1000L);

g.setName("spring boot");

g.setRemark("spring boot ...");

g.setCreatedTime(**new** Date());

System.***out***.println(g);

g=**new** Goods(200L, "lombok", "lombok...", **new** Date());

System.***out***.println(g);

g=**new** Goods()

.setId(300L)

.setName("framework")

.setRemark("framework...")

.setCreatedTime(**new** Date());

g.doPrint();

}

}

## Mybatis-Plus 插件应用

[MyBatis-Plus](https://github.com/baomidou/mybatis-plus)（简称 MP）是一个 [MyBatis](http://www.mybatis.org/mybatis-3/" \t "_blank) 的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生，如图-17所示：

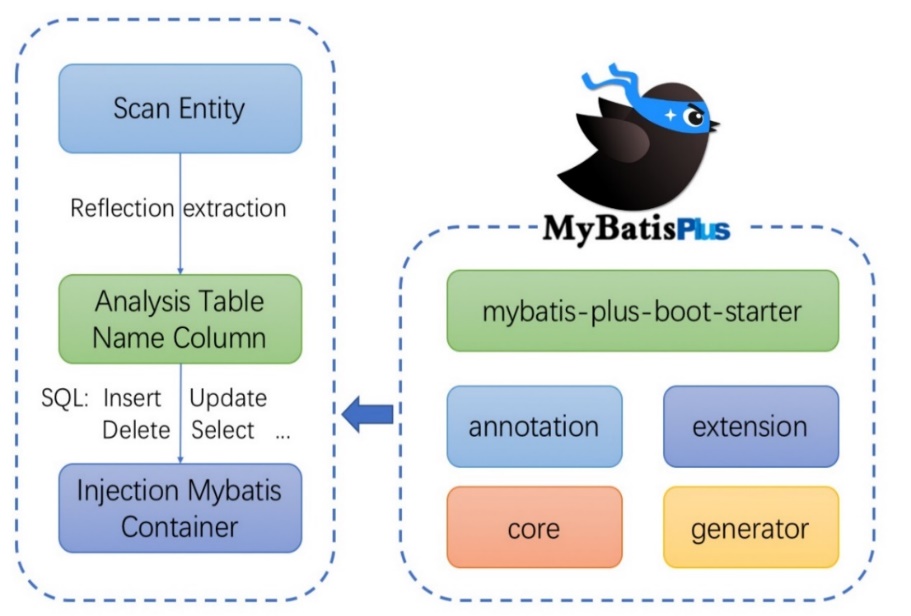


图-17

说明：Mybatis-plus插件的具体应用，自学实现。

# 总结

## 重难点分析

* Spring boot 核心优势？（开箱即用）
* Spring boot 项目的创建方式，启动依赖，自动配置。
* Spring boot 整合数据源,mybatis,spring mvc的实现。

## FAQ分析

* 创建spring starter 项目时pom.xml文件错误？(工具版本，maven私服，本地库)
* 项目中application.properties文件没有提示？（关闭重新打开）
* SpringBoot 启动动类的包扫描位置？(启动类所在包以及子包)
* SpringBoot 内置的连接池是什么？(HiKariCP)
* SpringBoot 连接数据库的配置没有问题,数据库连不上？(服务是否启动，端口是否正确)
* SpringBoot 整合mybatis时候是否可以同时存在注解和xml的映射？
* SpringBoot 整合mybatis时如何实现SQL日志的输出?
* SpringBoot默认支持热部署吗?(需要配置)

## BUG分析

* **数据库版本及url配置问题，如图-18所示：**

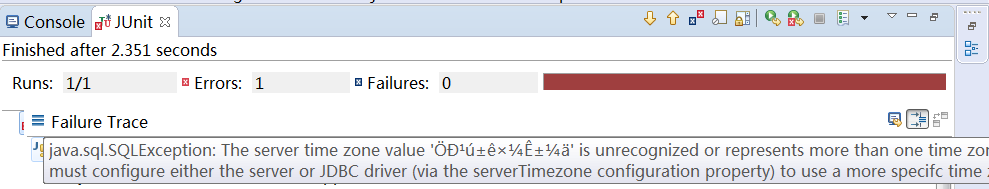


图-18

其解决方案为：

1. 降低数据库驱动版本 (传统的maven项目通常会这么做)
2. 在url中追加时区配置(serverTimezone=GMT%2B8)

* **JVM端口号被占用。**

Caused by: java.net.BindException: Address already in use: bind

at sun.nio.ch.Net.bind0(Native Method) ~[na:1.8.0\_45]

at sun.nio.ch.Net.bind(Unknown Source) ~[na:1.8.0\_45]

at sun.nio.ch.Net.bind(Unknown Source) ~[na:1.8.0\_45]

其解决方案为：

1. 关闭已启动的tomcat,重新启动。
2. 杀进程
3. 查找进程id:netstat -ano|findstr "8080"
4. 基于进程id杀进程:taskkill /f /pid 进程id

* **指定命名空间下的元素ID已经存在。**

Caused by: java.lang.IllegalArgumentException:

Mapped Statements collection already contains value for

com.cy.pj.goods.dao.GoodsDao.deleteObject. please check file

[D:\WORKSPSTS\CGB-SBOOT-01\target\classes\mapper\ GoodsMapper.xml]

and com/cy/pj/goods/dao/GoodsDao.java (best guess)

其解决方案为：

1. 可能是GoodsDao接口或映射文件对应的映射元素有重复。

* **单元测试方法没有找到，如图-19所示：**

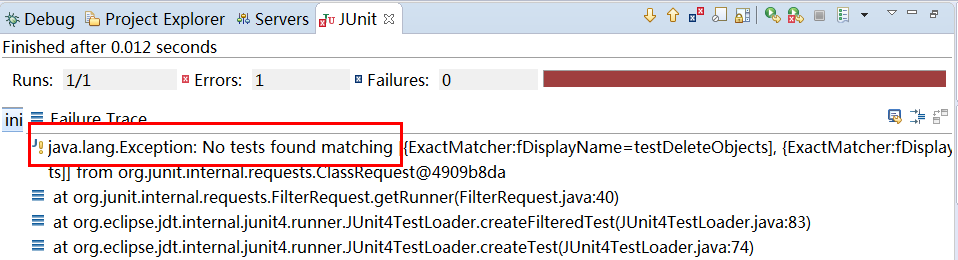


图-19

其解决方案为:

1. 单元测试方法可能使用了private修饰,有返回值或有参数。
2. 单元测试方法没写@Test注解。
3. Spring Boot项目有多个启动类。
4. Spring Boot 单元测试类没有写在单元测试包中。

* **MyBatis 绑定异常，如图-20所示**

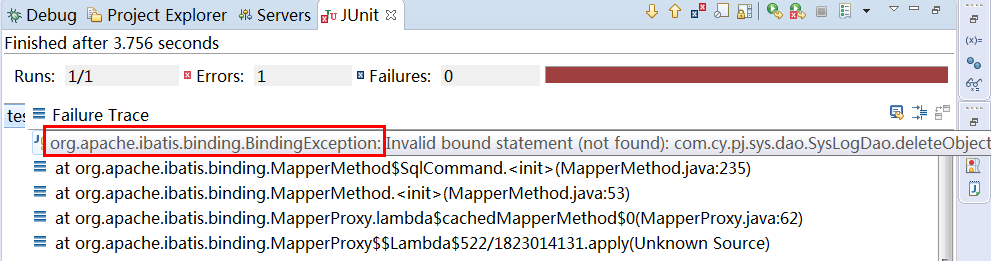


图-20

其解决方案为：

1. 检查接口类全名与映射文件中的命名空间是否不一致。
2. 检查接口方法名与映射文件中的方法名是否不一致。
3. 检查接口方法参数与映射文件可接收的参数不匹配。
4. 检查接口对应的映射文件路径是否配置不正确(mapper-locations)。

* **URL请求资源不存在，如图-21所示：**

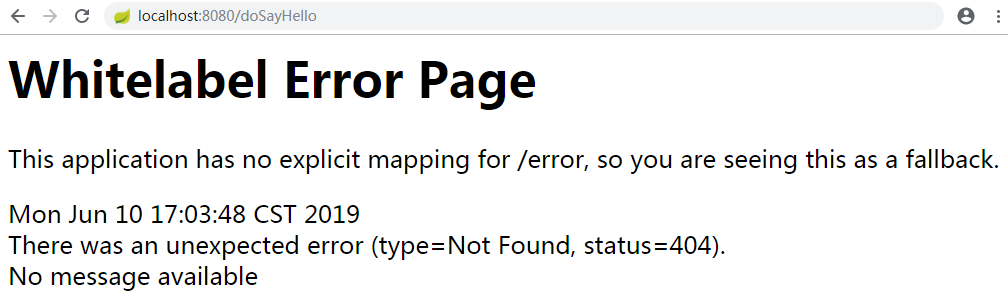


图-21

其解决方案:

1. 检查tomcat启动是否OK。
2. 检查url对应的服务端资源是否匹配(客户端url与controller定义url)。
3. 检查controller的包结构(所在包应在主启动类所在包的同包或子包中)。
4. 检查项目中是否已添加thymeleaf依赖及配置(内置视图解析器)。

* **视图解析失败，如图-22所示**

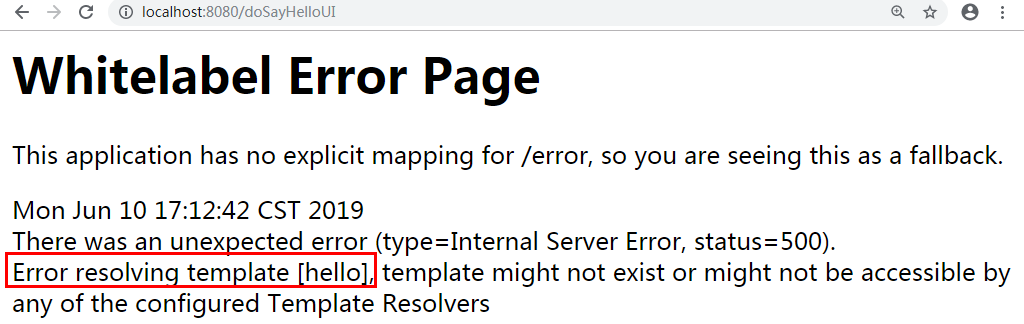


图-22

其解决方案：

1. 视图解析器的配置有问题（有可能方法没有实用@ReponseBody修饰）。
2. 解决方案:检查视图解析器前缀,后缀的配置。