Framework

Spring Boot

[1. 引言 1-2](#_Toc11055871)

[1.1. Spring 传统项目整合弊病 1-2](#_Toc11055872)

[1.2. Spring 家族主要成员构成 1-2](#_Toc11055873)

[1.3. Spring 应用中的技术趋势 1-3](#_Toc11055874)

[2. Spring Boot 简介 2-3](#_Toc11055875)

[2.1. Spring Boot概述 2-3](#_Toc11055876)

[2.2. Spring boot入门案例 2-4](#_Toc11055877)

[2.2.1. 初始环境配置 2-4](#_Toc11055878)

[2.2.2. 项目创建及结构分析 2-4](#_Toc11055879)

[2.2.3. 项目业务基本实现及测试 2-6](#_Toc11055880)

[2.2.4. Runner启动器应用分析及实践 2-7](#_Toc11055881)

[3. Spring Boot整合Web资源 3-8](#_Toc11055882)

[3.1. Web资源整合入门 3-8](#_Toc11055883)

[3.1.1. 项目创建及初始配置 3-8](#_Toc11055884)

[3.1.2. 项目业务编写测试 3-11](#_Toc11055885)

[3.2. Web资源整合进阶 3-13](#_Toc11055886)

[3.2.1. 定义静态页面 3-13](#_Toc11055887)

[3.2.2. 配置映射前缀后缀 3-13](#_Toc11055888)

[3.2.3. 控制层方法定义及测试 3-13](#_Toc11055889)

[4. Spring Boot 整合数据源 4-14](#_Toc11055890)

[4.1. 单数据源整合实现 4-14](#_Toc11055891)

[4.1.1. 添加数据源相关依赖 4-14](#_Toc11055892)

[4.1.2. 配置数据源 4-14](#_Toc11055893)

[4.1.3. 对数据源进行单元测试 4-15](#_Toc11055894)

[4.2. 多数据源整合实现（扩展） 4-15](#_Toc11055895)

[5. Spring Boot 整合ORM框架 5-16](#_Toc11055896)

[5.1. 整合mybatis框架 5-16](#_Toc11055897)

[5.1.1. 添加mybatis启动依赖 5-16](#_Toc11055898)

[5.1.2. 配置mybatis框架并测试 5-16](#_Toc11055899)

[5.1.3. 基本业务实现及单元测试 5-16](#_Toc11055900)

[6. Spring Boot 应用加强 6-17](#_Toc11055901)

[6.1. mybatis-plus插件应用（扩展） 6-17](#_Toc11055902)

[6.2. Lombok插件应用(扩展) 6-18](#_Toc11055903)

[7. 总结 7-18](#_Toc11055904)

[7.1. 重点难点 7-18](#_Toc11055905)

[7.2. FAQ分析 7-18](#_Toc11055906)

[7.3. Bug 7-19](#_Toc11055907)

# 引言

## Spring 传统项目整合弊病

在传统的spring应用项目中：

1. 配置相对复杂（例如大量xml）
2. 依赖冲突严重（maven依赖版本不同）
3. 扩展相对复杂

思考：如何让初级程序员从复杂的配置中解脱出来呢？

## Spring 家族主要成员构成

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。

## Spring 应用中的技术趋势

Spring 自5.x以后：

1. 只能向下兼容到JDK8，并支持Kotlin语言。
2. WebFlux异步编程技术突起，但全面落地还需要一些时日。
3. Portlet过时了，Velocity也不在维护，JasperReport不再流行等。
4. Spring boot，Spring cloud 异军突起。

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构建与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：

1. 自动配置(Auto Configuration)
2. 起步依赖(Starter Dependency)
3. 健康检查(Actator)
4. …

说明：Spring Boot地址：<https://spring.io/projects/spring-boot>

## Spring boot入门案例

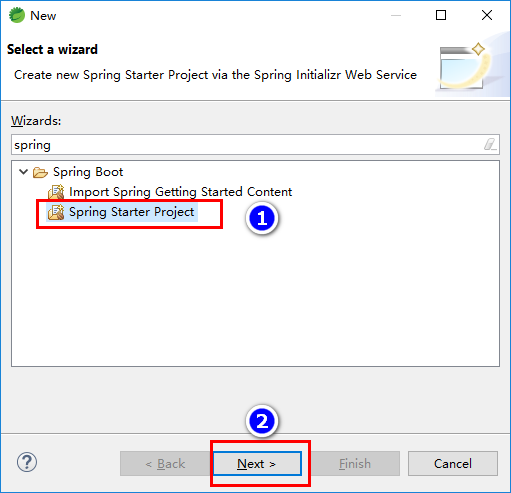
### 初始环境配置

1. 下载JDK1.8，并进行环境变量配置
2. 下载maven(apache-maven-3.6.1)，并进行相关配置(本地库，私服)。
3. 下载sts最新版(sts-4.2.2.RELEASE)，并进行maven配置.

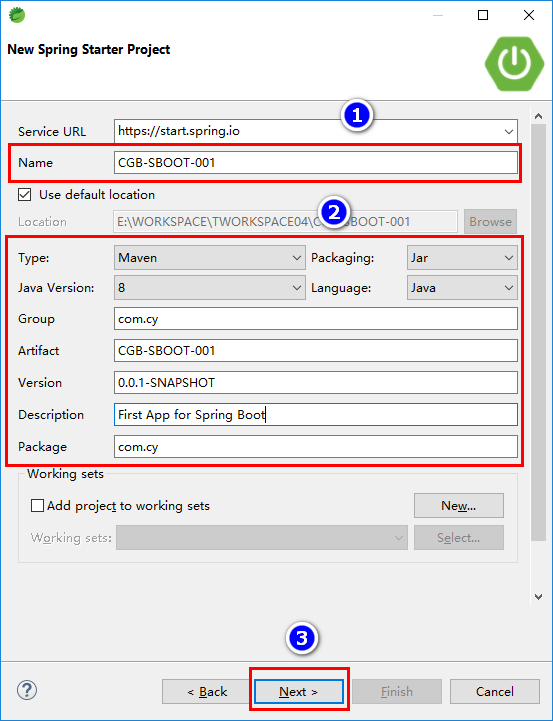
### 项目创建及结构分析

打开STS 集成开发工具，并基于如下步骤实现项目创建。

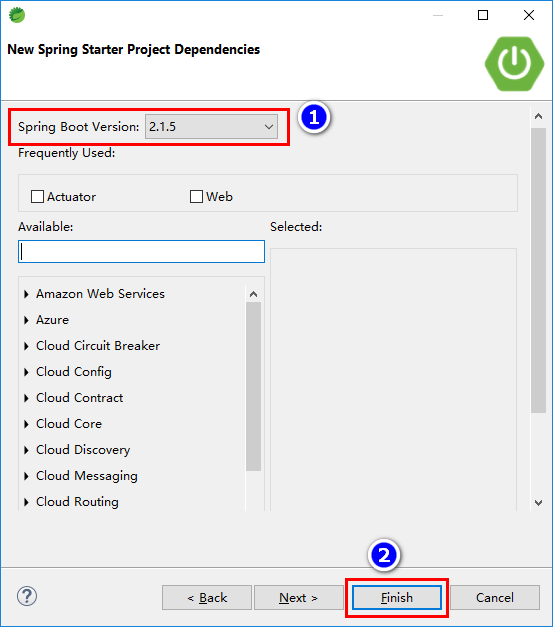
Step01:打开项目新建窗口（快捷键 ctrl+n）, 搜索spring，选择spring starter 项目。



Step02 填写项目基本信息。



Step03 选择Spring Boot版本。



### 项目业务基本实现及测试

业务描述：创建一个cache类然后交给spring 管理。

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

@Component

@Scope("singleton")

**public** **class** Cache {

**public** Cache() {

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

}

@PostConstruct

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

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

}

@PreDestroy

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

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

}

}

修改sringboot 启动类，进行bean的获取及测试。

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** CgbSboot01ApplicationTests {

@Autowired

**private** ApplicationContext ctx;

@Test

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

Cache cache=

ctx.getBean("cache", Cache.**class**);

System.out.println(cache);

}

}

### Runner启动器应用分析及实践

如果你想在Spring Boot启动的时候运行一些特定的代码，你可以实现接口 ApplicationRunner或者 CommandLineRunner，这两个接口实现方式一样，它们都只提供了一个run方法，其中：

1. **CommandLineRunner**：启动获取命令行参数。
2. **ApplicationRunner**：启动获取应用启动的时候参数。

SpringBoot在项目启动后会遍历所有实现CommandLineRunner的类并执行run方法。

例如：

@SpringBootApplication

**public** **class** CgbSpringbootApplication **implements** CommandLineRunner{

@Autowired

**private** ApplicationContext ctx;

@Override

**public** **void** run(String... args) **throws** Exception {

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

}

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

SpringApplication.*run*(CgbSpringbootApplication.**class**, args);

}

}

说明：

如果启动的时候有多个ApplicationRunner和CommandLineRunner，想控制它们的启动顺序，可以实现 org.springframework.core.Ordered接口或者使用 org.springframework.core.annotation.Order注解。

# Spring Boot 整合数据源

## 单数据源整合实现

### 添加数据源相关依赖

编辑项目中pom.xml，基于基本业务添加如下依赖：

添加mysql数据库驱动依赖

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

添加druid启动依赖

<dependency>

<groupId>com.alibaba</groupId>

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

<version>1.1.10</version>

</dependency>

添加spring对象jdbc支持

<dependency>

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

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

</dependency>

### 配置数据源

打开application.properties配置文件，添加如下内容。

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

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

spring.datasource.username=root

spring.datasource.password=root

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

简化实现

spring.datasource.url=jdbc:mysql:///jtsys?useSSL=false&serverTimezone=GMT%2B8

spring.datasource.username=root

spring.datasource.password=root

### 对数据源进行单元测试

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** CgbSbootApplicationTests {

@Autowired

**private** DataSource dataSource;

@Autowired

**private** SysLogDao sysLogDao;

@Test

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

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

}

}

## 多数据源整合实现（扩展）

自己课后扩展

# Spring Boot 整合ORM框架

## 整合mybatis框架

### 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>

### 配置mybatis框架并测试

在mybatis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

### 基本业务实现及单元测试

定义业务接口

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

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

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

@Mapper //3.4

**public** **interface** SysLogDao {

@Delete("delete from sys\_logs where id=#{id}")

**int** deleteObject(Integer id);

}

定义测试类

@RunWith(SpringRunner.**class**)

@SpringBootTest

**public** **class** CgbSboot02ApplicationTests {

@Autowired

**private** SysLogDao sysLogDao;

@Test

**public** **void** testSysLogDao() {

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

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

}

}

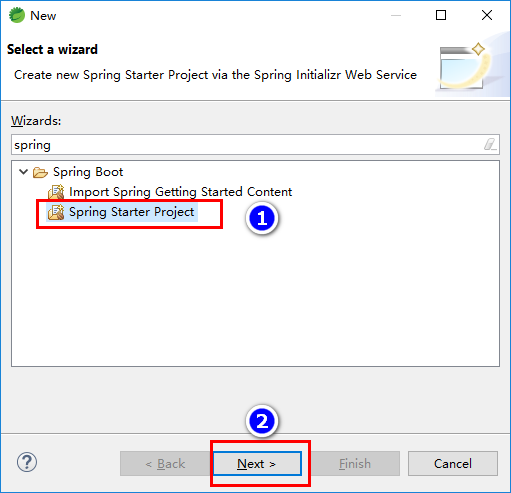
# Spring Boot整合Web资源

## Web资源整合入门

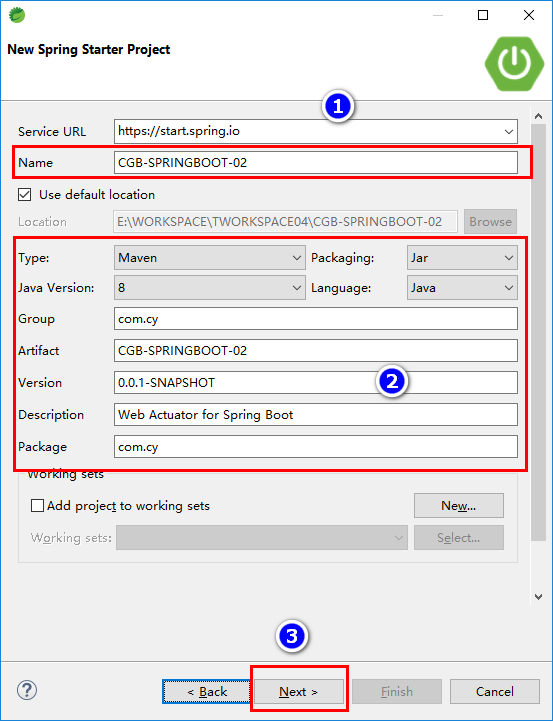
基于STS创建WEB应用项目（打包方式选择jar包方式），实现简易Spring Boot项目访问。

### 项目创建及初始配置

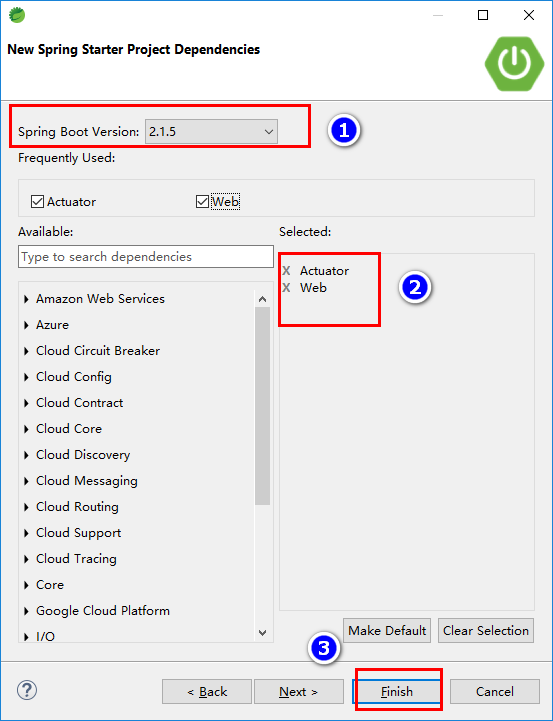
Step01:打开项目新建窗口（快捷键 ctrl+n）, 搜索spring，选择spring starter 项目。



Step02：填写项目基本信息



Step03:选择spring boot版本,并添加web和actuator依赖。其中actuator在spring项目中主要负责健康检查以及一些监控功能。



### 项目业务编写测试

Step01:编写HelloController类并将其交给spring管理。

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

@Controller

**public** **class** HelloController {

@RequestMapping("doSayHello")

@ResponBody

**public** String doSayHello() {

**return** "hello spring";

}

}

}

启动项目（WEB项目内置tomcat，启动时会自动启动tomcat）

**package** com.cy;

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

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

@SpringBootApplication

**public** **class** CgbSpringApplication{

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

SpringApplication.*run*(CgbSpringApplication.**class**, args);

}

}

浏览器测试方式：

打开chrome浏览器，在地址栏直接输入如下地址

http://localhost:8080/doSayHello

健康检查分析：在浏览器中输入

<http://localhost:8080/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=\*

## Web资源整合进阶

### 定义静态页面

在src/main/resources的templates目录下创建pages目录,并在目录中创建hello.html文件，其内容自己定义。

### 配置映射前缀后缀

添加thymeleaf模板引擎依赖（有此依赖可以去除web依赖）

<dependency>

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

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

</dependency>

在application.properties文件中添加视图解析器

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

spring.thymeleaf.suffix=.html

### 控制层方法定义及测试

@RequestMapping("doSayHelloUI")

**public** String doSayHelloUI() {

**return** "hello";

}

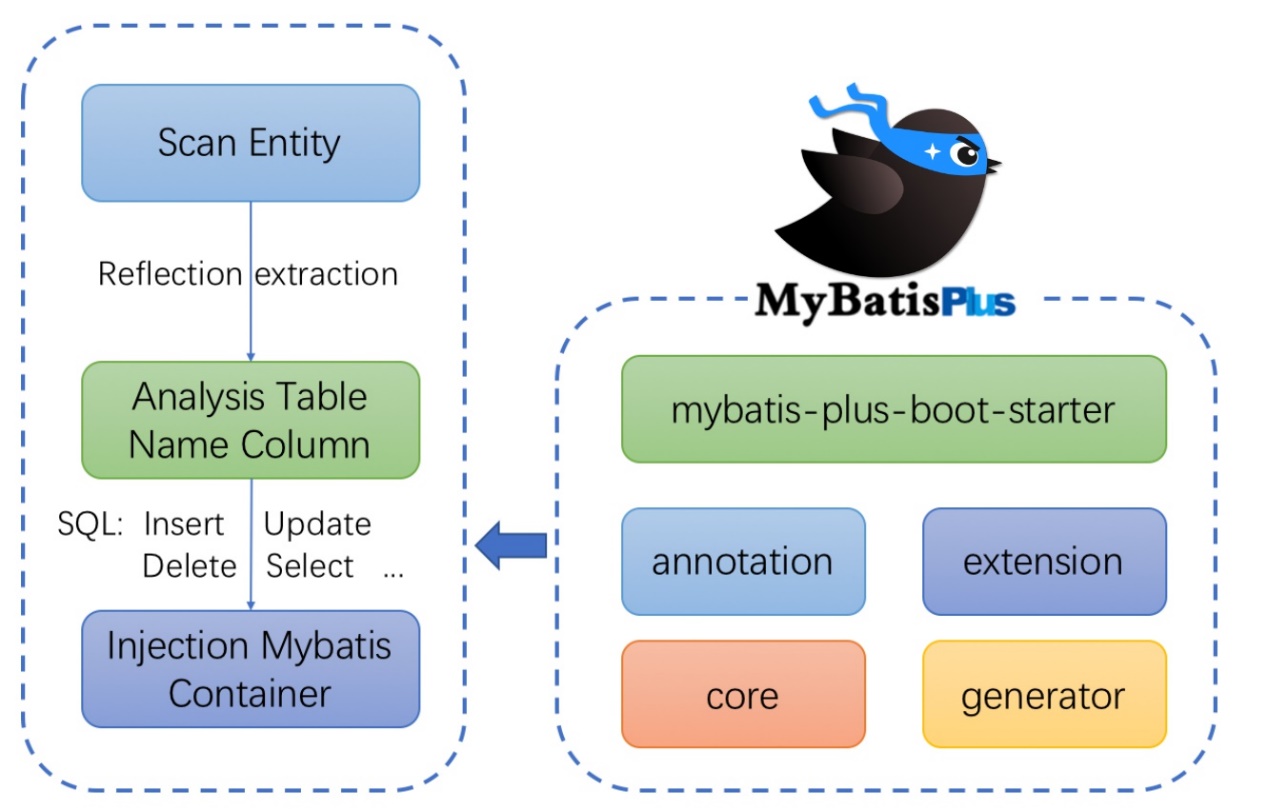
启动项目，然后在在浏览器输入：

http://localhost:8080/doSayHelloUI

# Spring Boot 应用加强

## mybatis-plus插件应用（扩展）

[MyBatis-Plus](https://github.com/baomidou/mybatis-plus)（简称 MP）是一个 [MyBatis](http://www.mybatis.org/mybatis-3/) 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。



## Lombok插件应用(扩展)

愿景分析：能够省略pojo类中set/get/toString方法的编写。

Step01：添加依赖

<dependency>

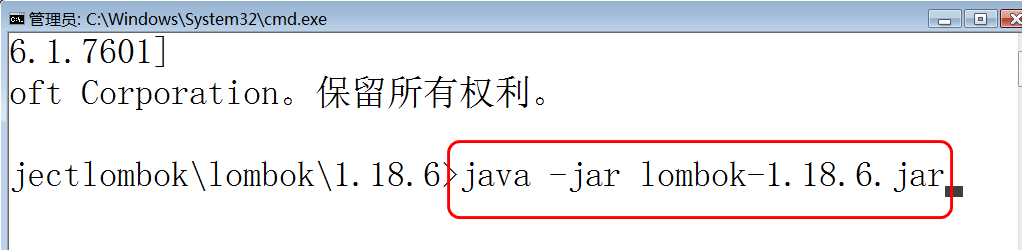
<groupId>org.projectlombok</groupId>

<artifactId>lombok</artifactId>

</dependency>

Step02: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。

Step03:利用java –jar 指令启动lombok插件，例如



然后在弹出的框中选择sts安装文件。

# 总结

## 重点难点

## FAQ分析

## Bug