# 核心功能

1. 独立运行的Spring项目:SpringBoot可以以jar包形式独立运行；
2. 内嵌Servlet容器:SpringBoot 可选择内嵌Tomcat Jetty或者Undertow，这样无须以war包形式部署项目；
3. 提供Starter简化maven配置；
4. 自动配置Spring
5. 准生产的应用监控：Spring Boot提供基于http、ssh、telnet对运行时的项目进行监控；
6. 无代码生成和xml配置

Spring tool suite (Eclipse)

Spring Boot cli spring boot 控制台

可用spring boot cli 构建springboot项目

spring init --build=maven --java-version=1.8 --dependencies=web --packaging=jar --boot-version=1.3.0.M1 --groupId=com.demo --artifactId=demo

# 基本知识

https://blog.csdn.net/qq\_33530388/article/details/73291912

应用主类Application.java置于root package下，通常我们会在应用主类中做一些框架配置扫描等配置，我们放在root package下可以帮助程序减少手工配置来加载到我们希望被Spring加载的内容

## @SpringBootApplication

@SpringBootApplication源码如下：

@Target({ElementType.TYPE})

@Retention(RetentionPolicy.RUNTIME)

@Documented

@Inherited

@Configuration

@EnableAutoConfiguration

@ComponentScan

public @interface SpringBootApplication {

Class<?>[] exclude() default {};

String[] excludeName() default {};

@AliasFor(

annotation = ComponentScan.class,

attribute = "basePackages"

)

String[] scanBasePackages() default {};

@AliasFor(

annotation = ComponentScan.class,

attribute = "basePackageClasses"

)

Class<?>[] scanBasePackageClasses() default {};

}

从源代码中得知 **@SpringBootApplication** 被 **@Configuration**、**@EnableAutoConfiguration**、**@ComponentScan** 注解所修饰，换言之 Springboot 提供了统一的注解来替代以上三个注解，简化程序的配置。

### @EnableAutoConfiguration

spring文档说明：

Enable auto-configuration of the spring Application Context, attempting to guess and configure beans that you are likely to need. Auto-configuration classes are usually applied based on your classpath and what beans you have defined.

启用 Spring 应用程序上下文的自动配置，试图猜测和配置您可能需要的bean。自动配置类通常采用基于你的 classpath 和已经定义的 beans 对象进行应用。

The package of the class that is annotated with @EnableAutoConfiguration has specific significance and is often used as a ‘default’. For example, it will be used when scanning for@Entity classes. It is generally recommended that you place@EnableAutoConfiguration in a root package so that all sub-packages and classes can be searched.

被 @EnableAutoConfiguration 注解的类所在的包有特定的意义，并且作为默认配置使用。例如，当扫描 @Entity类的时候它将本使用。通常推荐将 @EnableAutoConfiguration 配置在 root 包下，这样所有的子包、类都可以被查找到。

Auto-configuration classes are regular Spring Configuration beans. They are located using the SpringFactoriesLoader mechanism (keyed against this class).Generally auto-configuration beans are @Conditional beans (most often using @ConditionalOnClass and @ConditionalOnMissingBean annotations).

Auto-configuration类是常规的 Spring 配置 Bean。它们使用的是 SpringFactoriesLoader 机制（以 EnableAutoConfiguration 类路径为 key）。通常 auto-configuration beans 是 @Conditional beans（在大多数情况下配合 @ConditionalOnClass 和 @ConditionalOnMissingBean 注解进行使用）。

### SpringFactoriesLoader 机制：

SpringFactoriesLoader会查询包含 META-INF/spring.factories 文件的JAR。

当找到spring.factories文件后，SpringFactoriesLoader将查询配置文件命名的属性。

**EnableAutoConfiguration**的key 值为

org.springframework.boot.autoconfigure.EnableAutoConfiguration。

根据此 key 对应的值进行 spring 配置。在 spring-boot-autoconfigure.jar文件中，包含一个 spring.factories 文件。

### @ComponentScan

spring文档说明：

Configures component scanning directives for use with @Configuration classes. Provides support parallel with Spring XML’s element.

为 @Configuration注解的类配置组件扫描指令。同时提供与 Spring XML’s 元素并行的支持。

Either basePackageClasses() or basePackages() (or its alias value()) may be specified to define specific packages to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

无论是 basePackageClasses() 或是 basePackages() （或其 alias 值）都可以定义指定的包进行扫描。如果指定的包没有被定义，则将从声明该注解的类所在的包进行扫描。

Note that the element has an annotation-config attribute; however, this annotation does not. This is because in almost all cases when using @ComponentScan, default annotation config processing (e.g. processing @Autowired and friends) is assumed. Furthermore, when using AnnotationConfigApplicationContext, annotation config processors are always registered, meaning that any attempt to disable them at the @ComponentScan level would be ignored.

注意， 元素有一个 annotation-config 属性（详情：[http://www.cnblogs.com/exe19/p/5391712.html](http://www.cnblogs.com/exe19/p/5391712.html" \t "https://blog.csdn.net/claram/article/details/_blank)），但是 @ComponentScan 没有。这是因为在使用 @ComponentScan 注解的几乎所有的情况下，默认的注解配置处理是假定的。此外，当使用 AnnotationConfigApplicationContext， 注解配置处理器总会被注册，以为着任何试图在 @ComponentScan 级别是扫描失效的行为都将被忽略。

通俗的讲，@ComponentScan 注解会自动扫描指定包下的全部标有 @Component注解 的类，并注册成bean，当然包括 @Component 下的子注解@Service、@Repository、@Controller。@ComponentScan 注解没有类似 、的属性。

## Spring boot 启动原理

https://www.cnblogs.com/shamo89/p/8184960.html

**SpringFactoriesLoader**

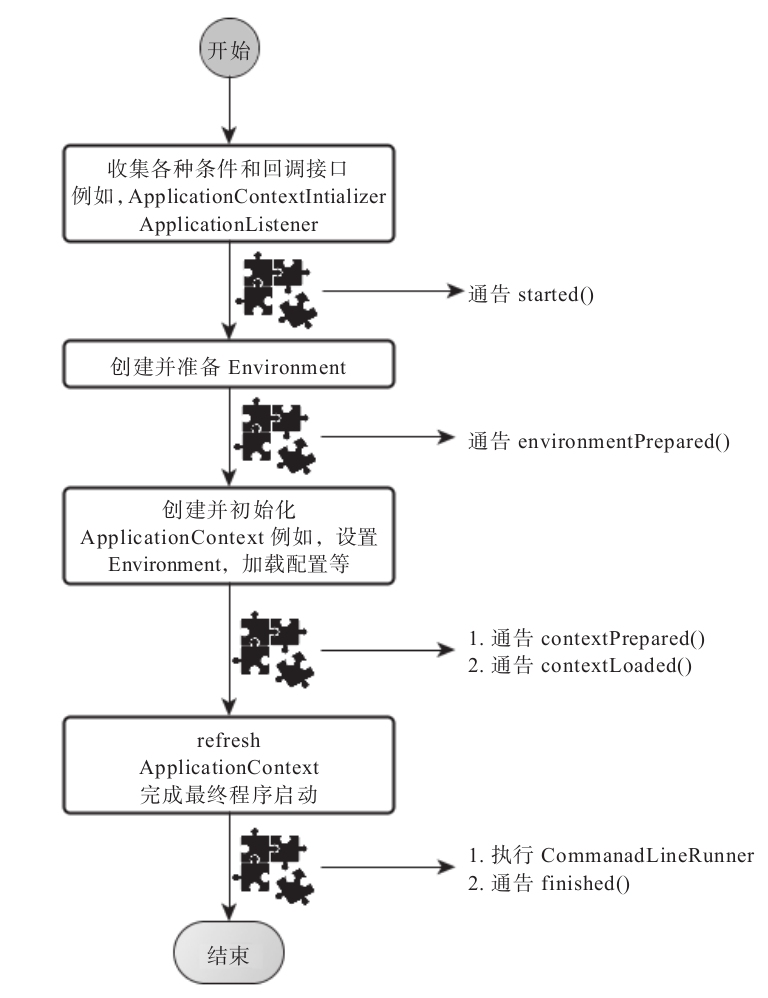
SpringFactoriesLoader属于Spring框架私有的一种扩展方案，其主要功能就是从指定的配置文件META-INF/spring.factories加载配置。

**配合@EnableAutoConfiguration使用**

[根据@EnableAutoConfiguration的完整类名](mailto:根据@EnableAutoConfiguration的完整类名org.springframework.boot.autoconfigure.EnableAutoConfiguration)

[org.springframework.boot.autoconfigure.EnableAutoConfiguration](mailto:根据@EnableAutoConfiguration的完整类名org.springframework.boot.autoconfigure.EnableAutoConfiguration)

作为查找的Key,获取对应的一组@Configuration类；



## 配置Bean

### @ConfigurationProperties

读取配置文件中 属性key 为prefix 的value，并配置对象或被@Bean修饰的方法中

@ConfigurationProperties(prefix = "section3")

### @EnableConfigurationProperties

向IOC容器中注册Bean3

@EnableConfigurationProperties({Bean3.class})

### Yml

例子：

appilcation.yml

oneuser:

username: 猴子

配置类

@Configuration

public class BeanConfiguration {

@Bean

@ConfigurationProperties(prefix = "oneuser")

public User oneuser(){

return new User();

}

}

## 基于JavaConfig的配置

@Configuration:

该注解锁标记类吗，即为IOC 容器的JavaConfig配置类

例：

@Configurationpublic class MockConfiguration{

@Bean

public MockService mockService(){

return new MockServiceImpl();

}

}

与下方 xml等价

<bean id="mockService" class="..MockServiceImpl">

<propery name ="dependencyService" ref="dependencyService" />

</bean>

<bean id="dependencyService" class="DependencyServiceImpl"></bean>

效果等价：

@Configuration <==> beans

@Bean <==> bean

## 定制Banner

1. 在src/main/resources下新建一个banner.txt文档   
   2.通过[http://patorjk.com/software/taag](http://patorjk.com/software/taag/" \t "https://blog.csdn.net/u012702547/article/details/_blank)网站生成需要的字符，将字符拷贝到步骤1所创建的txt文档中。

### 关闭Banner

//修改Banner的模式为OFF

SpringApplicationBuilder builder = new SpringApplicationBuilder(Test19SpringBoot2Application.class);

builder.bannerMode(Banner.Mode.OFF).run(args);

## Starter pom

Spring-boot为我们提供了简化企业级开发绝大多数场景的starter pom，只要使用了应用场景所需要的starter pom，相关的技术配置将会消失，就可以得到Spring boot为我们提供的自动配置的Bean。

除官方提供 starter，还有第三方为Spring Boot所写的starter pom

如 Handlerbar,Vaadin,Apach Camel,WRO4J,Spring Batch,HDIV，Jade 4J,Actitivi

## IOC容器:ConfigurableApplicationContext

实现 org.springframework.context 的 ApplicationContext接口；

即在使用 Spring Boot 后的 IOC 容器，

代替原本的

BeanFactory、

FileSystemXmlApplicationContext、

ClassPathXmlApplicationContext、

XmlWebApplicationContext

常用方法：

<T>T getBean(...)

Object getBean(String beanname)

String[] getBeanDefinitionNames()

# 日志配置

默认情况下Spring Boot使用Logback作为日志框架，也就是我们前面几篇博客中用到的打印日志方式，当然如果有需要我们可以手动配置日志级别以及日志输出位置，相比于我们在Spring容器中写的日志输出代码，这里的配置简直就是小儿科了，只需要在application.properties中添加如下代码：

logging.file=/home/sang/workspace/log.log

logging.level.org.springframework.web=debug

上面表示配置日志输出位置，下面配置日志级别。

# .yml文件配置

Yml配置更简洁，目前STS 3.7.0已开始提供支持yaml语言的配置，而IDEA则只对springboot的properties配置提供了自动提示功能，且@PropertySourece注解也不支持加载yaml文件。在日常开发中，我们习惯用properties来配置。故目前推荐使用property文件配置。

## 注意

1、原有的key，例如[spring](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "https://blog.csdn.net/testcs_dn/article/details/_blank).jpa.properties.[hibernate](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "https://blog.csdn.net/testcs_dn/article/details/_blank).dialect，按“.”分割，都变成树状的配置；

2、编辑yml文件时不能使用TAB键来缩进，否则启动会报错；

3、键值之间需要一个空格，比如enabled: true 即true前面其实是有一个空格的，否则会报错；

4、‘#‘ 为注释；

5、优先配置 application.properties

## 一、example:h2数据库的配置：

h2:

console:

enabled: true

path: '/h2'

datasource:

driver-class-name: org.h2.Driver

url: 'jdbc:h2:tcp://localhost/~/oai'

username: sa

password: ''

continue-on-error: true

tomcat.max-wait: 5000

tomcat.test-on-borrow: true

schema: 'classpath:ddl/oai.ddl'

## 二、Jpa（Java持久层API）

设置运行时控制台是否实时显示sql操作，以及是否运行通过ddl文件自动更新表结构到数据库。

jpa:

show-sql: true

hibernate:

ddl-auto: update

## 切换不同配置文件 profiles

application.yml :

#切换不同的配置文件  
#配置springmvc的视图

spring:

mvc:

view:

prefix: /WEB-INF/jsp/

suffix: .jsp

#切换不同的配置文件

profiles:

active: dev

application-dev.yml :

spring:

mvc:

view:

prefix: /WEB-INF/jsp/

suffix: .jsp

server:

port: 90

可以定义多个application-xxx.yml，需要引用到该文件就在application.yml中的profiles: active: xxx(对应前面的xxx,如：application-test.yml,这里active：后面的xxx则为test; active:test)

### **YAML缺点**

YAML文件不能通过@PropertySource注解加载，如果需要使用该方式，那就必须使用properties文件。

# 单元测试

<https://blog.csdn.net/tengxing007/article/details/73332038>

过时方法 正常

@RunWith(SpringJUnit4ClassRunner.class)

@SpringApplicationConfiguration(Application.class)

以下出错 不解

Pom文件

<dependency>

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

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

<scope>test</scope>

</dependency>

@RunWith(SpringJUnit4ClassRunner.class)

@SpringBootTest(classes=Application.class) // 指定spring-boot的启动类

在spring的其他项目中一般加载

@RunWith(SpringRunner.class)

@ContextConfiguration(locations={"classpath:spring-servlet.xml", "classpath:spring-dao-test.xml", "classpath:spring-service-test.xml"})

# 配置SpringMVC

## 配置流程

1、pom文件是继承自spring-boot-starter-parent）；

2、引入spring-boot-starter-web；

<dependency>

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

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

</dependency>

1. 在application.properties文件中对springmvc的一些参数进行配置；

spring.mvc.view.prefix=/WEB-INF/jsp

spring.mvc.view.suffix=.jsp

在application.yml中配置

server:

port: 8090 #配置端口

session-timeout: 30

tomcat.max-threads: 0

tomcat.uri-encoding: UTF-8

context-path: /hc-web #配置项目路径

#配置springmvc的视图

spring:

mvc:

view:

prefix: /WEB-INF/jsp/

suffix: .jsp

| **SpringBoot版本** | **配置** |
| --- | --- |
| 1.x | server.context-path=/demo |
| 2.x | server.servlet.context-path=/demo |

## 使用JSP

添加引用

<!-- jsp解析器 -->

<dependency>

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

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

<scope>provided</scope>

</dependency>

Spring boot 内置Tomcat，该Tomcat不包含Jsp相关依赖；

此外spring-boot-maven-plugin插件将在打包时自动将添加jar，故该依赖设置scope为provided

<!--排除内置tomcat-->

<dependency>

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

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

<exclusions>

<exclusion>

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

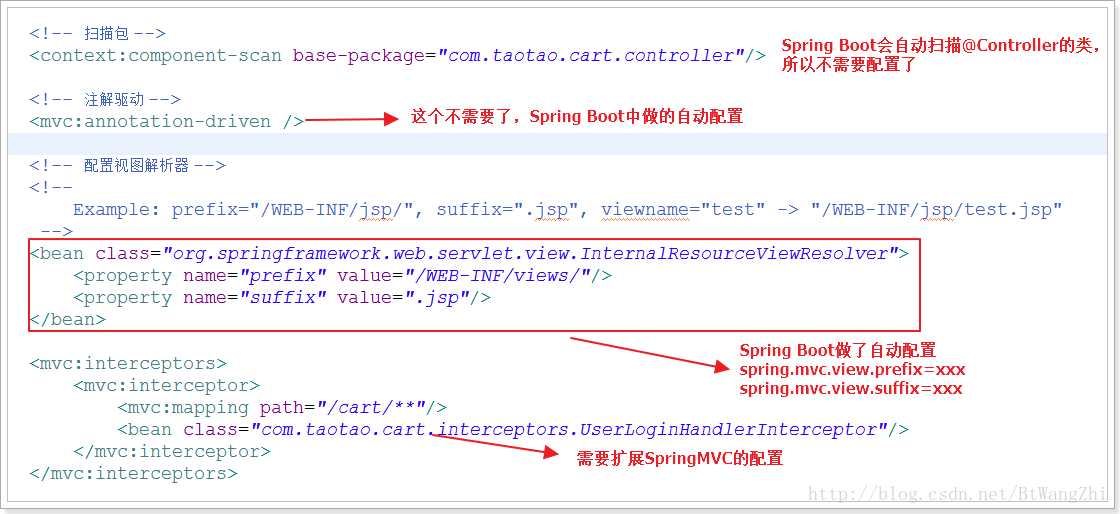
<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

## 对比传统SSM项目



# Spring boot整合数据库

https://www.cnblogs.com/yql1986/p/6819097.html

## 配置JdbcTemplate

pom文件

<!-- spring boot 使用jdbctemplate 连接数据库-->

<dependency>

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

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

</dependency>

<!-- mysql 驱动 -->

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

spring:

#配置数据源

datasource :

#数据源类型

#type: com.zaxxer.hikari.HikariDataSource

driver-class-name: com.mysql.jdbc.Driver

url: jdbc:mysql://localhost:3306/hc

username: root

password: root

max-idle: 10

max-wait: 10000

min-idle: 5

initial-size: 5

## 配置MyBatis

配置 整合包：

<!-- springboot整合MyBatis -->

<dependency>

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

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

<version>1.3.1</version>

</dependency>

**配置yml**

mybatis:

#config-locations: classpath:mybatis/mybatis-config.xml

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

type-aliases-package: hc.web.bean

**配置类**

@Configuration

@MapperScan("hc.web.mapper") //或在Mapper接口表姐 @Mapper注解

public class MyBatisConfiguration {

}

### 死配法： 配置类

@Configuration

@MapperScan("hc.web.mapper")

@EnableTransactionManagement //开启事务支持

@ConfigurationProperties(prefix = "mybatis")

public class MyBatisConfiguration {

private String MYBATIS\_CONFIG = "mybatis/mybatis-config.xml";

private String MAPPER\_PATH = "mybatis/mapper/\*\*.xml";

@Autowired

private DataSource dataSource;

public String typeAliasPackage = "hc.web.bean" ;

/\*\*

\*创建sqlSessionFactoryBean 实例

\* 并且设置configtion 如驼峰命名.等等

\* 设置mapper 映射路径

\* 设置datasource数据源

\* @return

\*/

@Bean

public SqlSessionFactoryBean createSqlSessionFactoryBean() throws IOException {

SqlSessionFactoryBean sqlSessionFactoryBean = new SqlSessionFactoryBean();

/\*\* 设置mybatis configuration 扫描路径 \*/

sqlSessionFactoryBean.setConfigLocation(new ClassPathResource(MYBATIS\_CONFIG));

/\*\* 添加mapper 扫描路径 \*/

PathMatchingResourcePatternResolver pathMatchingResourcePatternResolver = new PathMatchingResourcePatternResolver();

String packageSearchPath = ResourcePatternResolver.CLASSPATH\_ALL\_URL\_PREFIX + MAPPER\_PATH;

sqlSessionFactoryBean.setMapperLocations(pathMatchingResourcePatternResolver.getResources(packageSearchPath));

/\*\* 设置datasource \*/

sqlSessionFactoryBean.setDataSource(dataSource);

/\*\* 设置typeAlias 包扫描路径 \*/

sqlSessionFactoryBean.setTypeAliasesPackage(typeAliasPackage);

return sqlSessionFactoryBean;

}

}

### 配置多数据源

上述静态多个数据库的这种配置是不支持分布式事务的，也就是同一个事务中，不能操作多个数据库。

@MapperScan(

basePackages = { "com.cml.springboot.sample.db" },

sqlSessionFactoryRef = "sqlSessionFactory")

### MapperScannerRegistrar将扫描 basePackages 下的mapper

### AOP动态配置数据源