# Springboot的启动流程，启动时怎么启动tomcat

启动springboot使用的是SpringApplication.run方法，run方法中实现了启动流程

在run方法中创建SpringApplication对象，调用run方法，

创建SpringApplication对象，在构造函数的作用

1. 先把主类保存起来
2. 判断运行项目的类型
3. 扫描当前路径下META-INF/spring.factories文件的，加载ApplicationContextInitializer接口实例
4. 扫描当前路径下META-INF/spring.factories文件的，加载ApplicationListener接口实例

利用SPI机制扫描 META-INF/spring.factories 这个文件，并且加载 ApplicationContextInitializer、ApplicationListener 接口实例。

1、ApplicationContextInitializer 这个类当springboot上下文Context初始化完成后会调用

2、ApplicationListener 当springboot启动时事件change后都会触发

**上面就是SpringApplication初始化的代码，new SpringApplication()没做啥事情 ，利用SPI机制主要加载了META-INF/spring.factories 下面定义的事件监听器接口实现类**

执行run方法

在run方法中主要做的事情

1. 启动计时器StopWatch
2. 设置环境变量configureHeadlessProperty
3. 获取事件监听器SpringApplicationRunListener类型，并且执行starting()方法
4. 准备环境了，并且把环境跟spring上下文绑定好，并且执行environmentPrepared()方法
5. 判断一些环境的值，并设置一些环境的值
6. 打印banner
7. 创建上下文，根据项目类型创建上下文，**AnnotationConfigServletWebServerApplicationContext**
8. 获取异常报告事件监听
9. 准备上下文，执行完成后调用contextPrepared()方法,contextLoaded()方法
10. 这个是spring启动的代码了，这里就回去里面就回去扫描并且初始化单实列bean了

**refreshContext(context);**

1. 执行ApplicationRunListeners中的started()方法
2. 执行Runner（ApplicationRunner和CommandLineRunner）

实例化上下文**AnnotationConfigServletWebServerApplicationContext，**

refreshContext方法中执行refresh方法，refresh方法中执行上下文的refresh方法这个就是加载bean的方法，在这个方法中的onRefresh方法，一个钩子方法，它会钩到它子类重写onRefresh()方法，也就是由

AnnotationConfigServletWebServerApplicationContext的父类ServletWebServerApplicationContext来实现，在其内部执行createWebServer();方法，在这个方法中创建TomcatServletWebServerFactory的实例，它的方法getWebServer中就实现了tomcat的启动

**private** **void** refreshContext(ConfigurableApplicationContext context) {

refresh(context);

**if** (**this**.registerShutdownHook) {

**try** {

context.registerShutdownHook();

}

**catch** (AccessControlException ex) {

// Not allowed in some environments.

}

}

}

1、new了一个SpringApplication对象，使用SPI技术加载加载 ApplicationContextInitializer、ApplicationListener 接口实例

2、调用SpringApplication.run() 方法

3、调用createApplicationContext()方法创建上下文对象，创建上下文对象同时会注册spring的核心组件类（ConfigurationClassPostProcessor 、AutowiredAnnotationBeanPostProcessor 等）。

4、调用refreshContext() 方法启动Spring容器和内置的Servlet容器

自定义start组件

Resources目录下META-INF/spring.factories文件，在文件中配置了MybatisAutoConfiguration类

@SpringBootApplication注解使用了@EnableAutoConfiguration注解，这个注解

导入了AutoConfigurationImportSelector类

通过starter创建指定的bean,加载自定义starter

@SpringBootApplication中包含了@EnableAutoConfiguration注解，将会加载process-》getAutoConfigurationEntry-》getCandidateConfigurations

在这个方法中会扫描Meta-inf包下面的spring.factories配置文件，从配置文件中读取所有的AutoConfiguration类，在AutoConfiguration

类中创建对应组件的bean实例

定义一个properties类用来封装要创建组件的参数，具体的配置就是在resources目录下的application.properties文件中，

在类上使用@ConfigurationProperties(prefix = "tom")，参数prefix指定配置前缀

定义AutoConfiguration类，在类上使用注解

@Configuration

@ConditionalOnClass(MyAuto.class)指定创建的组件，ConditionalOnClass注解是表示当给定的类名在类路径上存在，则实例化当前Bean

@EnableConfigurationProperties(MyAutoProperties.class)

@ConditionalOnProperty(prefix = "tom", value = "enable", matchIfMissing = true)

public class MyAutoConfiguration

{

在类中创建bean

@Bean

@ConditionalOnMissingBean

public MyAuto myAuto(MyAutoProperties myAutoProperties)

{

MyAuto myAuto = new MyAuto();

myAuto.setCode(myAutoProperties.getCode());

myAuto.setName(myAutoProperties.getName());

System.out.println("init");

return myAuto;

}

}

在resources下创建META-INF文件夹，在文件夹中创建spring.factories文件来配置AutoConfiguration的全类名

MyAutoConfiguration就是一个spring的配置类，找到这个配置类后交给spring来执行bean注解修饰的方法从而来实例化对应的组件

AnnotationConfigServletWebServerApplicationContext

Servletwebapplicationcontext

Mybatis的starter启动

在pom引入mybatis starter的依赖

在mybatis-spring-boot-autoconfiguration中有两个类

分别是mybatisautoconfiguration和mybatisproperties

Mybatisautoconfiguration类上使用了@configuration注解，表示这是一个spring的java配置类

这个配置类中创建的bean有sqlsessiontemplate，SqlSessionFactoryBean，这2个都是spring集成mybatis时配置的bean;这里就帮我们配置好了；

Mybatisproperties就是mybatis需要的配置信息

在classpath目录下的META-Info文件夹中的spring.factories配置mybatisautoconfiguration的全类名

Springbootapplication注解中有enableautoconfiguration注解，

Enableautoconfiguration注解包含了@AutoConfigurationPackage注解和import注解

Import注解导入的类AutoConfigurationImportSelector中会去扫描spring.factories文件加载MybatisAutoconfiguration这个配置类

[AutoConfigurationPackage注解包含@Import(AutoConfigurationPackages.Registrar.class)](mailto:AutoConfigurationPackage注解包含@Import(AutoConfigurationPackages.Registrar.class))，在Registrar中将配置类的bean注册到spring

容器中

Spring Boot使开发独立的，产品级别的基于Spring的应用变得非常简单，你只需"just run"。 我们为Spring平台及第三方库提供开箱即用的设置，这样你就可以有条不紊地开始。多数Spring Boot应用需要很少的Spring配置。

你可以使用Spring Boot创建Java应用，并使用java -jar启动它或采用传统的war部署方式。

spring Boot 是为 spring 服务的，是用来简化新 spring 应用的初始搭建以及开发过程的。Spring Boot是Spring开源组织下的子项目，是Spring组件一站式解决方案，主要是简化了使用Spring的难度，简省了繁重的配置，提供了各种启动器，开发者能快速上手。

### SSM的问题

* 依赖太多了, 且存在版本问题
* 配置太多了且每次都一样, 大部分工程, 配置每次都是一样的, 从一个地方拷贝到另外一个地方. 且Spring发展10多年, 各种配置版本太多, 对于很多程序员来说, 分不清哪个是有效, 哪个无效.
* 部署太麻烦. 需要tomcat部署, 项目结构也需要照着[Java EE](http://lib.csdn.net/base/17)的目录结构来写.

### SpringBoot特点

* 创建独立的Spring应用程序
* 嵌入的Tomcat，无需部署WAR文件
* 简化Maven配置
* 自动配置Spring
* 提供生产就绪型功能，如指标，健康检查和外部配置
* 绝对没有代码生成和对XML没有要求配置

### SpringBoot功能

* **自动配置(auto-configuration)**

一项简化配置的功能，比如在classpath中发现有spring security的jar包，则自动创建相关的bean等

* **starters(简化依赖)**

这个比较关键，方便spring去集成各类组件，比如redis、mongodb等等。

**Spring Boot 的配置文件有哪几种格式？它们有什么区别？**

.properties 和 .yml，它们的区别主要是书写格式不同。

1).properties，以keyvalue的形式

app.user.name = javastack

2).yml ：**yml每个k的冒号后面一定都要加一个空格**

app: user: name: javastack

yml是跨语言的:可以在包括JAVA，go，python等大量的语言中使用，比如做云计算使用go和java的时候，可以通过配置中心使用同一份配置！

支持列表:区别于properties只支持键值对数据，yml配置文件支持列表

## 静态资源访问

静态资源：js, css, html, 图片，音视频等

静态资源路径：是指系统可以直接访问的路径，且路径下的所有文件均可被用户直接读取。

Spring Boot默认提供静态资源目录位置需置于classpath下，目录名需符合如下规则：

/static

/public

/resources

/META-INF/resources

在resources目录下面建立static文件夹，在文件夹里面任意放张图片。

命名为：enjoy.jpg

## Spring Boot 热加载/部署

热部署不会用在生产环境，但对于程序员开发的效率，还是有一定帮助的，所谓的热部署，就是在应用程序在不停止的情况下，实现新的部署

spring-boot-devtools 是一个为开发者服务的一个模块，其中最重要的功能就是自动应用代码更改到最新的App上面去。原理是在发现代码有更改之后，重新启动应用，但是速度比手动停止后再启动还要更快，更快指的不是节省出来的手工操作的时间。

其深层原理是使用了两个ClassLoader，一个Classloader加载那些不会改变的类（第三方Jar包），另一个ClassLoader加载会更改的类，称为 restart ClassLoader

,这样在有代码更改的时候，原来的restart ClassLoader 被丢弃，重新创建一个restart ClassLoader，由于需要加载的类相比较少，所以实现了较快的重启时间

修改pom文件，增加

<dependency>

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

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

</dependency>

修改properties文件

如果使用的 Thymeleaf 模板，那么请直接在application.properties中添加

spring.thymeleaf.cache=false

如果使用的 FreeMarker 模板，那么请直接在application.properties中添加

spring.freemarker.cache=false

## Actuator监控管理

Actuator是spring boot的一个附加功能,可帮助你在应用程序生产环境时监视和管理应用程序。可以使用HTTP的各种请求来监管,审计,收集应用的运行情况.特别对于微服务管理十分有意义

缺点：没有可视化界面(Spring cloud 还会用到这功能，就可以看到界面了)

修改pom文件，添加依赖

<dependency>

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

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

</dependency>

修改application.properties文件，启动监控端点

# 加载所有的端点/默认只加载了 info / health

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

# 描述信息

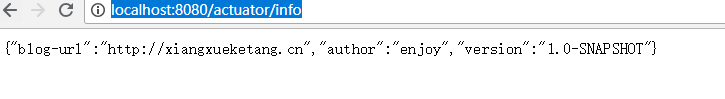
info.blog-url=http://xiangxueketang.cn

info.author=enjoy

info.version=@project.version@

重新启动，在地址栏输入

<http://localhost:8080/actuator/info>



在界面看到这说明监控成功

Actuator访问路径

通过actuator/+端点名就可以获取相应的信息。

|  |  |
| --- | --- |
| 路径 | 作用 |
| /actuator/beans | 显示应用程序中所有Spring bean的完整列表。 |
| /actuator/configprops | 显示所有配置信息。 |
| /actuator/env | 陈列所有的环境变量。 |
| /actuator/mappings | 显示所有@RequestMapping的url整理列表。 |
| /actuator/health | 显示应用程序运行状况信息 up表示成功 down失败 |
| /actuator/info | 查看自定义应用信息 |