springboot(十二)：springboot如何测试打包部署

 2017/05/09

有很多网友会时不时的问我，spring boot项目如何测试，如何部署，在生产中有什么好的部署方案吗？这篇文章就来介绍一下spring boot 如何开发、调试、打包到最后的投产上线。

**开发阶段**

**单元测试**

在开发阶段的时候最重要的是单元测试了，springboot对单元测试的支持已经很完善了。

1、在pom包中添加spring-boot-starter-test包引用

<dependency>

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

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

<scope>test</scope>

</dependency>

2、开发测试类

以最简单的helloworld为例，在测试类的类头部需要添加：@RunWith(SpringRunner.class)和@SpringBootTest注解，在测试方法的顶端添加@Test即可，最后在方法上点击右键run就可以运行。

**@RunWith(**SpringRunner**.**class**)**

**@SpringBootTest**

**public** **class** **ApplicationTests** **{**

**@Test**

**public** **void** **hello()** **{**

System**.**out**.**println**(**"hello world"**);**

**}**

**}**

实际使用中，可以按照项目的正常使用去注入dao层代码或者是service层代码进行测试验证，spring-boot-starter-test提供很多基础用法，更难得的是增加了对Controller层测试的支持。

*//简单验证结果集是否正确*

Assert**.**assertEquals**(**3**,** userMapper**.**getAll**().**size**());**

*//验证结果集，提示*

Assert**.**assertTrue**(**"错误，正确的返回值为200"**,** status **==** 200**);**

Assert**.**assertFalse**(**"错误，正确的返回值为200"**,** status **!=** 200**);**

引入了MockMvc支持了对Controller层的测试，简单示例如下：

**public** **class** **HelloControlerTests** **{**

**private** MockMvc mvc**;**

*//初始化执行*

**@Before**

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

mvc **=** MockMvcBuilders**.**standaloneSetup**(new** HelloController**()).**build**();**

**}**

*//验证controller是否正常响应并打印返回结果*

**@Test**

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

mvc**.**perform**(**MockMvcRequestBuilders**.**get**(**"/hello"**).**accept**(**MediaType**.**APPLICATION\_JSON**))**

**.**andExpect**(**MockMvcResultMatchers**.**status**().**isOk**())**

**.**andDo**(**MockMvcResultHandlers**.**print**())**

**.**andReturn**();**

**}**

*//验证controller是否正常响应并判断返回结果是否正确*

**@Test**

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

mvc**.**perform**(**MockMvcRequestBuilders**.**get**(**"/hello"**).**accept**(**MediaType**.**APPLICATION\_JSON**))**

**.**andExpect**(**status**().**isOk**())**

**.**andExpect**(**content**().**string**(**equalTo**(**"Hello World"**)));**

**}**

**}**

单元测试是验证你代码第一道屏障，要养成每写一部分代码就进行单元测试的习惯，不要等到全部集成后再进行测试，集成后因为更关注整体运行效果，很容易遗漏掉代码底层的bug.

**集成测试**

整体开发完成之后进入集成测试，spring boot项目的启动入口在 Application类中，直接运行run方法就可以启动项目，但是在调试的过程中我们肯定需要不断的去调试代码，如果每修改一次代码就需要手动重启一次服务就很麻烦，spring boot非常贴心的给出了热部署的支持，很方便在web项目中调试使用。

pom需要添加以下的配置：

<dependencies>

<dependency>

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

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

<optional>true</optional>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<fork>true</fork>

</configuration>

</plugin>

</plugins>

</build>

添加以上配置后，项目就支持了热部署，非常方便集成测试。

**投产上线**

其实我觉得这个阶段，应该还是比较简单一般分为两种；一种是打包成jar包直接执行，另一种是打包成war包放到tomcat服务器下。

**打成jar包**

如果你使用的是maven来管理项目，执行以下命令既可以

cd 项目跟目录（和pom.xml同级）

mvn clean package

*## 或者执行下面的命令*

*## 排除测试代码后进行打包*

mvn clean package -Dmaven.test.skip**=**true

打包完成后jar包会生成到target目录下，命名一般是 项目名+版本号.jar

启动jar包命令

java -jar target/spring-boot-scheduler-1.0.0.jar

这种方式，只要控制台关闭，服务就不能访问了。下面我们使用在后台运行的方式来启动:

nohup java -jar target/spring-boot-scheduler-1.0.0.jar &

也可以在启动的时候选择读取不同的配置文件

java -jar app.jar --spring.profiles.active**=**dev

也可以在启动的时候设置jvm参数

java -Xms10m -Xmx80m -jar app.jar &

**gradle**  
如果使用的是gradle,使用下面命令打包

gradle build

java -jar build/libs/mymodule-0.0.1-SNAPSHOT.jar

**打成war包**

打成war包一般可以分两种方式来实现，第一种可以通过eclipse这种开发工具来导出war包，另外一种是使用命令来完成，这里主要介绍后一种

1、maven项目，修改pom包

将

<packaging>jar</packaging>

改为

<packaging>war</packaging>

2、打包时排除tomcat.

<dependency>

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

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

</dependency>

<dependency>

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

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

<scope>provided</scope>

</dependency>

在这里将scope属性设置为provided，这样在最终形成的WAR中不会包含这个JAR包，因为Tomcat或Jetty等服务器在运行时将会提供相关的API类。

3、注册启动类

创建ServletInitializer.java，继承SpringBootServletInitializer ，覆盖configure()，把启动类Application注册进去。外部web应用服务器构建Web Application Context的时候，会把启动类添加进去。

**public** **class** **ServletInitializer** **extends** SpringBootServletInitializer **{**

**@Override**

**protected** SpringApplicationBuilder **configure(**SpringApplicationBuilder application**)** **{**

**return** application**.**sources**(**Application**.**class**);**

**}**

**}**

最后执行

mvn clean package -Dmaven.test.skip**=**true

会在target目录下生成：项目名+版本号.war文件，拷贝到tomcat服务器中启动即可。

**gradle**

如果使用的是gradle,基本步奏一样，build.gradle中添加war的支持，排除spring-boot-starter-tomcat：

...

apply plugin: 'war'

...

dependencies **{**

compile**(**"org.springframework.boot:spring-boot-starter-web:1.4.2.RELEASE"**){**

exclude mymodule:"spring-boot-starter-tomcat"

**}**

**}**

...

再使用构建命令

gradle build

war会生成在build\libs 目录下。

**生产运维**

**查看JVM参数的值**

可以根据java自带的jinfo命令：

jinfo -flags pid

来查看jar 启动后使用的是什么gc、新生代、老年代分批的内存都是多少，示例如下：

-XX:CICompilerCount**=**3 -XX:InitialHeapSize**=**234881024 -XX:MaxHeapSize**=**3743416320 -XX:MaxNewSize**=**1247805440 -XX:MinHeapDeltaBytes**=**524288 -XX:NewSize**=**78118912 -XX:OldSize**=**156762112 -XX:+UseCompressedClassPointers -XX:+UseCompressedOops -XX:+UseFastUnorderedTimeStamps -XX:+UseParallelGC

* -XX:CICompilerCount ：最大的并行编译数
* -XX:InitialHeapSize 和 -XX:MaxHeapSize ：指定JVM的初始和最大堆内存大小
* -XX:MaxNewSize ： JVM堆区域新生代内存的最大可分配大小
* …
* -XX:+UseParallelGC ：垃圾回收使用Parallel收集器

**如何重启**

**简单粗暴**

直接kill掉进程再次启动jar包

ps -ef|grep java

*##拿到对于Java程序的pid*

kill -9 pid

*## 再次重启*

Java -jar xxxx.jar

当然这种方式比较传统和暴力，所以建议大家使用下面的方式来管理

**脚本执行**

如果使用的是maven,需要包含以下的配置

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

<configuration>

<executable>true</executable>

</configuration>

</plugin>

如果使用是gradle，需要包含下面配置

springBoot **{**

executable **=** true

**}**

启动方式：

1、 可以直接./yourapp.jar 来启动

2、注册为服务

也可以做一个软链接指向你的jar包并加入到init.d中，然后用命令来启动。

init.d 例子:

ln -s /var/yourapp/yourapp.jar /etc/init.d/yourapp

chmod +x /etc/init.d/yourapp

这样就可以使用stop或者是restart命令去管理你的应用。

/etc/init.d/yourapp start|stop|restart

或者

service yourapp start|stop|restart

到此 springboot项目如何测试、联调和打包投产均已经介绍完，以后可以找时间研究一下springboot的自动化运维，以及spring boot 和docker相结合的使用。

[**示例代码-github**](https://github.com/ityouknow/spring-boot-examples)

[**示例代码-码云**](https://gitee.com/ityouknow/spring-boot-examples)