

# learning spring boot

石廷鑫

Published  
with GitBook



# Table of Contents

Introduction	0
hello world	1
embed container	2
Externalizing Configuration	3
Spring Boot Actuator	4

## Spring boot 系列教程

## Spring Boot小试牛刀

1. 访问<http://start.spring.io>.

2. 在对应表单中填充如下内容:

Group : com.zjs

Artifact : hello-spring-boot

Name : hello-spring-boot

Description : Hello Spring Boot

Package Name : com.zjs

Type : Maven Project

Packaging : Jar

Language : Java

java Version : 1.8

Spring Boot Version : 1.3.0 RC1

3. 在项目依赖(Project dependencies)选择:

o Web

4. 点击项目生成按钮(Generate Project). 下载生成的项目

5. 将项目导入eclipse。 选择导入已存在的maven工程。

6. 在com.zjs.HelloSpringBootApplication.类增加@RestController 注解 同时增加如下的请求处理函数

```
@RequestMapping("/")
public String hello() {
    return "Hello World!";
}
```

7. 运行mvn install 安装需要的依赖包

8. 运行 mvn:spring-boot:run

9. 访问<http://127.0.0.1:8080>

恭喜你，完成第一个spring boot 应用

## 发布一个内嵌容器的**Web**应用

Spring Boot默认内嵌Apache Tomcat web容器

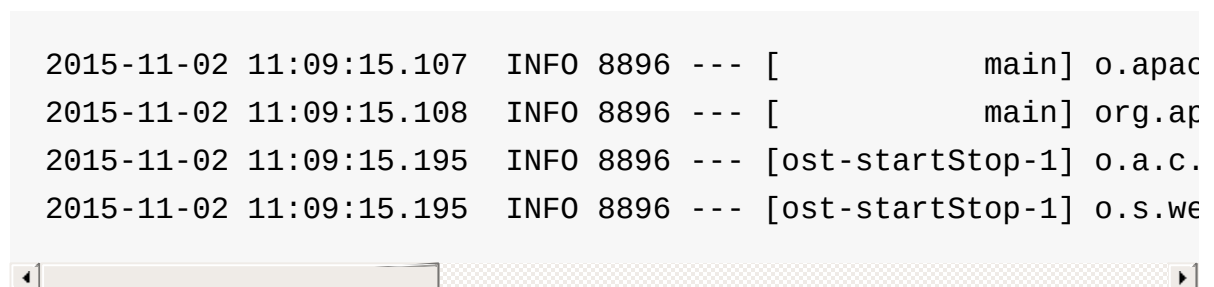
### 1. 打包应用

```
mvn package
```

### 2. 运行应用

```
java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

### 3. 你可以看到已经启动内嵌的tomcat容器，端口为8080



```
2015-11-02 11:09:15.107 INFO 8896 --- [main] o.apac
2015-11-02 11:09:15.108 INFO 8896 --- [main] org.ap
2015-11-02 11:09:15.195 INFO 8896 --- [ost-startStop-1] o.a.c.
2015-11-02 11:09:15.195 INFO 8896 --- [ost-startStop-1] o.s.we
```

## 将内潜容器改为**Eclipse Jetty**

打开pom.xml将

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

替换为

```
<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>
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jetty</artifactId>
</dependency>
```

执行打包命令：mvn package

运行应用 java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar



```
2015-11-02 11:02:52.743 INFO 8312 --- [main] e.j.JettyE
2015-11-02 11:02:52.746 INFO 8312 --- [main] org.eclips
2015-11-02 11:02:52.925 INFO 8312 --- [main] applicatio
2015-11-02 11:02:52.930 INFO 8312 --- [main] o.s.web.co
```

相同的应用:tomcat 启动时间是1207 ms, jetty的启动时间是1112 ms. 目前来看, jetty的启动比tomcat稍强。

## 通过配置文件来配置启动参数

1. 将 `src/main/resources/application.properties` 重新命名为 `application.yml`。并增加如下内容

```
greeting: Hello
```

2. 在 `com.zjs.HelloSpringBootApplication` 中增加一个 `greeting` 字段，并通过 IOC 注入。

```
@Value("${greeting}")  
String greeting;
```

3. 将 `hello()` 函数的内容改为

```
@RequestMapping("/")  
public String hello() {  
    return String.format("%s World!", greeting);  
}
```

4. `mvn package` 打包程序

5. 运行程序

```
java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

6. 访问 <http://localhost:8080>，输出为 Hello World!

## 通过环境变量来配置参数

1. 增加环境变量，并运行

```
set greeting=test&java -jar target/hello-spring-boot-0.0.1-SNA
```

2. 访问 <http://localhost:8080>，输出为 test World!



通过上面例子可以看出，环境变量的配置会覆盖文件的配置

## 使用spring的profile进行配置

1. 在application.yml 文件中增加 test profile

```
greeting: Hello

---

spring:
  profiles: test

greeting: test
```

1. 重新打包
2. 默认运行

```
java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

输出为 hello world

指定profile运行:

```
```
set SPRING_PROFILES_ACTIVE=test&java -jar target/hello-spring-boot
```
```

输出为 test world

## 配置冲突

如果同时指定profile和环境变量出现的情况，测试如下情况：

```
set SPRING_PROFILES_ACTIVE=test&set greeting=test111&java -jar 1
```

输出为 test111 world。

# 使用Actuator进行性能监控

## 配置 Actuator

1. 在pom文件中增加

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

## 查看内置的监控点（Endpoints）

1. 打包应用

```
mvn package
```

2. 运行程序

```
java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

3. 尝试一下URL

<http://localhost:8080/beans>

输出所有在Spring context中的所有bean

<http://localhost:8080/autoconfig>

输出所有在应用启动时的所有自动配置内容

<http://localhost:8080/configprops>

输出所有目前应用的配置内容@ConfigurationProperties

<http://localhost:8080/env>

输出所有环境变量以及java的系统属性

<http://localhost:8080/mappings>

输出所有的URL mapping

<http://localhost:8080/dump>

线程dump

<http://localhost:8080/trace>

现在跟踪信息（默认是最近的HTTP 请求信息）

## 增加版本控制信息

在src/main/resources/application.yml中增加如下信息：

```
...  
info:  
  build:  
    artifact: @project.artifactId@  
    name: @project.name@  
    description: @project.description@  
    version: @project.version@  
...
```

通过这种方法将项目的maven信息映射到/info endpoint。Spring Boot的plugin将在build的时候自动进行替换。

## 健康指示器

Spring Boot提供了一个health(<http://localhost:8080/health>)允许查看不同的健康指标。

1. 通常情况下，出于安全原因考虑，/health 端点只显示运行（Up）或停止（down）我们可以打开安全限制

```
endpoints:  
health:  
    sensitive: false
```

## 2. 创建一个简单的Health示例，用于显示动态的健康情况

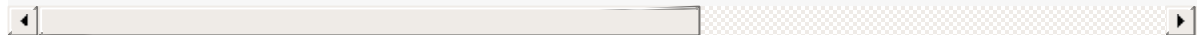
```
``` package com.zjs;
```

```
import org.springframework.boot.actuate.health.Health; import  
org.springframework.boot.actuate.health.HealthIndicator; import  
org.springframework.stereotype.Component;
```

```
import java.util.Random;
```

```
@Component public class FlappingHealthIndicator implements  
HealthIndicator{
```

```
    private Random random = new Random(System.currentTimeMillis())  
  
    @Override  
    public Health health() {  
        int result = random.nextInt(100);  
        if (result < 50) {  
            return Health.down().withDetail("flapper", "failure").  
        } else {  
            return Health.up().withDetail("flapper", "ok").withDet  
        }  
    }  
}
```



```
}
```

```
```
```

## 1. 打包运行：

```
java -jar target/hello-spring-boot-0.0.1-SNAPSHOT.jar
```

2. 访问 <http://localhost:8080/health> 将显示现象的健康情况。

```
{
  "status": "DOWN",
  "flapping":
  {
    "status": "DOWN",
    "flapper": "failure",
    "random": 49
  },
  "diskSpace":
  {
    "status": "UP",
    "total": 446114476032,
    "free": 444336259072,
    "threshold": 10485760
  }
}
```

## 度量

Spring Boot 提供了一个metrics端点 (<http://localhost:8080/metrics>) 为应用自动收集性能指标。当然也可以定义或创建自个关注的指标

1. 创建一个简单的度量示例。代码如下：

...

```
package com.zjs;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.actuate.metrics.CounterService;
import org.springframework.stereotype.Component;

@Component
public class GreetingService {

    @Autowired
    CounterService counterService;

    @Value("${greeting}")
    String greeting;

    public String getGreeting() {
        counterService.increment("counter.services.greeting.incr");
        return greeting;
    }
}
```

...

在例子中我们自动注入CounterService 来计算对getGreeting()方法调用了多少次。

1. 重构HelloSpringBootApplication, 代码如下：

```
@Autowired
private GreetingService greetingService;

@RequestMapping("/")
public String hello() {
    return String.format("%s World!", greetingService.getGreet
}

public static void main(String[] args) {
    SpringApplication.run(HelloSpringBootApplication.class, ar
}
```

现在讲对hello()的调用代理到我们创建的服务GreetingService

- i. 打包运行程序
- ii. 用两个浏览器分别打开<http://localhost:8080> 和 <http://localhost:8080/metrics>。当不停的访问<http://localhost:8080>时，[counter.services.greeting.invoked](#)的调用指标也在增加。

```
"counter.services.greeting.invoked": 2
```