http://www.cnblogs.com/softidea/p/5759180.html

一般在一个项目中，总是会有好多个环境。比如：

开发环境 -> 测试环境 -> 预发布环境 -> 生产环境

每个环境上的配置文件总是不一样的，甚至开发环境中每个开发者的环境可能也会有一点不同，配置读取可是一个让人有点伤脑筋的问题。

Spring Boot提供了一种优先级配置读取的机制来帮助我们从这种困境中走出来。

常规情况下，我们都知道Spring Boot的配置会从application.properties中读取。实际上，从resource目录下的application.properties文件读取是Spring Boot配置链中的一环而已。

**外部化的配置**

在应用中管理配置并不是一个容易的任务，尤其是在应用需要部署到多个环境中时。通常会需要为每个环境提供一个对应的属性文件，用来配置各自的数据库连接信息、服务器信息和第三方服务账号等。通常的应用部署会包含开发、测试和生产等若干个环境。不同的环境之间的配置存在覆盖关系。测试环境中的配置会覆盖开发环境，而生产环境中的配置会覆盖测试环境。Spring 框架本身提供了多种的方式来管理配置属性文件。Spring 3.1 之前可以使用 PropertyPlaceholderConfigurer。  
Spring 3.1 引入了新的环境（Environment）和概要信息（Profile）API，是一种更加灵活的处理不同环境和配置文件的方式。不过 Spring 这些配置管理方式的问题在于选择太多，让开发人员无所适从。Spring Boot 提供了一种统一的方式来管理应用的配置，允许开发人员使用属性文件、YAML 文件、环境变量和命令行参数来定义优先级不同的配置值。

Spring Boot 所提供的配置优先级顺序比较复杂。按照优先级从高到低的顺序，具体的列表如下所示。

1. 命令行参数。
2. 通过 System.getProperties() 获取的 Java 系统参数。
3. 操作系统环境变量。
4. 从 java:comp/env 得到的 JNDI 属性。
5. 通过 RandomValuePropertySource 生成的“random.\*”属性。
6. **应用 Jar 文件之外的属性文件**。(通过spring.config.location参数)
7. 应用 Jar 文件内部的属性文件。
8. 在应用配置 Java 类（包含“@Configuration”注解的 Java 类）中通过“@PropertySource”注解声明的属性文件。
9. 通过“SpringApplication.setDefaultProperties”声明的默认属性。

Spring Boot 的这个配置优先级看似复杂，其实是很合理的。比如命令行参数的优先级被设置为最高。  
这样的好处是可以在测试或生产环境中**快速地修改配置参数值**，而不需要重新打包和部署应用。

SpringApplication 类默认会把以“**--**”开头的**命令行参数**转化成应用中可以使用的**配置参数**，如 “--name=Alex” 会设置配置参数 “name” 的值为 “Alex”。如果不需要这个功能，可以通过 “SpringApplication.**setAddCommandLineProperties**(false)” 禁用解析命令行参数。

RandomValuePropertySource 可以用来生成测试所需要的各种不同类型的随机值，从而免去了在代码中生成的麻烦。RandomValuePropertySource 可以生成数字和字符串。数字的类型包含 int 和 long，可以限定数字的大小范围。以“random.”作为前缀的配置属性名称由 RandomValuePropertySource 来生成，如[代码清单 3](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing3) 所示。

**清单 3. 使用 RandomValuePropertySource 生成的配置属性**

user.id=${random.value}

user.count=${random.int}

user.max=${random.long}

user.number=${random.int(100)}

user.range=${random.int[100, 1000]}

**属性文件**

属性文件是最常见的管理配置属性的方式。Spring Boot 提供的 SpringApplication 类会搜索并加载 application.properties 文件来获取配置属性值。SpringApplication 类会在下面位置搜索该文件。

* 当前目录的“/config”子目录。
* 当前目录。
* classpath 中的“/config”包。
* classpath

上面的顺序也表示了该位置上包含的属性文件的优先级。优先级按照从高到低的顺序排列。可以通过“spring.config.name”配置属性来指定不同的属性文件名称。也可以通过“**spring.config.location**”来添加**额外的属性文件的搜索路径**。如果应用中包含多个 profile，可以为每个 profile 定义各自的属性文件，按照“application-{profile}”来命名。

对于配置属性，可以在代码中通过“@Value”来使用，如[代码清单 4](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing4) 所示。

**清单 4. 通过“@Value”来使用配置属性**

@RestController

@EnableAutoConfiguration

public class Application {

@Value("${name}")

private String name;

@RequestMapping("/")

String home() {

return String.format("Hello %s!", name);

}

}

在[代码清单 4](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing4) 中，变量 name 的值来自配置属性中的“name”属性。

**YAML**

相对于属性文件来说，YAML 是一个更好的配置文件格式。YAML 在 Ruby on Rails 中得到了很好的应用。SpringApplication 类也提供了对 YAML 配置文件的支持，只需要添加对 SnakeYAML 的依赖即可。[代码清单 5](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing5) 给出了 application.yml 文件的示例。

**清单 5. 使用 YAML 表示的配置属性**

spring:

profiles: development

db:

url: jdbc:hsqldb:file:testdb

username: sa

password:

---

spring:

profiles: test

db:

url: jdbc:mysql://localhost/test

username: test

password: test

[代码清单 5](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing5) 中的 YAML 文件同时给出了 development 和 test 两个不同的 profile 的配置信息，这也是 YAML 文件相对于属性文件的优势之一。除了使用“@Value”注解绑定配置属性值之外，还可以使用更加灵活的方式。[代码清单 6](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing6) 给出的是使用[代码清单 5](http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html#listing5) 中的 YAML 文件的 Java 类。通过“@ConfigurationProperties(prefix="db")”注解，配置属性中以“db”为前缀的属性值会被自动绑定到 Java 类中同名的域上，如 url 域的值会对应属性“db.url”的值。只需要在应用的配置类中添加“**@EnableConfigurationProperties**”注解就可以启用该自动绑定功能。

**清单 6. 使用 YAML 文件的 Java 类**

@Component

@ConfigurationProperties(prefix="db")

public class DBSettings {

private String url;

private String username;

private String password;

}

 http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html

这意味着，如果Spring Boot在**优先级更高的位置找到了配置，那么它就会无视优先级低的配置**。

比如，我在application.properties目录中，写入本地的MySQL的配置：

db.jdbc.driver=com.mysql.jdbc.Driver

db.jdbc.url=jdbc:mysql://127.0.0.1:3306/test?useUnicode=true&characterEncoding=UTF-8

db.jdbc.username=username

db.jdbc.password=password

在自己项目调试的阶段，项目总是会使用本地的MySQL数据库。而一旦打包之后，在外部声明一个test\_evn.properties.

启动Jar包的时候, 指定一个外部配置文件：

java -jar demo.jar **--spring.config.location**=/path/test\_evn.properties

这样一来，我们在开发者的机器上总是使用自己的配置，而一到对应的环境，就会使用高级的位置所做的配置。

在代码中读取这些配置也是非常方便的，在代码的逻辑中，其实是无需去关心这个配置是从什么地方来的，只用关注能获取什么配置就够了。

[复制代码](javascript:void(0);)

public class ApplicationConfigure {

@Value("${db.jdbc.driver}")

private String jdbcDriver;

@Value("${db.jdbc.url}")

private String jdbcUrl;

@Value("${db.jdbc.username}")

private String jdbcUsername;

@Value("${db.jdbc.password}")

private String jdbcPassword;

// mysql config class

// .....

}

[复制代码](javascript:void(0);)

有时候我们在项目启动的时候，总是需要先启动一些初始化的类，以前比较常见的做法是写再static块中，Spring Boot提供了一个CommandLineRunner接口，实现这个接口的类总是会被优先启动，并优先执行CommandLineRunner接口中提供的run()方法。

[复制代码](javascript:void(0);)

public class ApplicationConfigure implements CommandLineRunner {

@Value("${db.jdbc.driver}")

private String jdbcDriver;

@Value("${db.jdbc.url}")

private String jdbcUrl;

@Value("${db.jdbc.username}")

private String jdbcUsername;

@Value("${db.jdbc.password}")

private String jdbcPassword;

// mysql config class

// .....

@Override

public void run(String... strings) throws Exception {

// 预先加载的一些方法，类，属性。

}

}

[复制代码](javascript:void(0);)

如果有多个CommandLineRunner接口实现类，那么可以通过注解@Order来规定所有实现类的运行顺序。

通过这一系列API的帮助，Spring Boot让环境配置变得轻松很多。

http://www.cnblogs.com/whthomas/p/5270917.html

http://www.ibm.com/developerworks/cn/java/j-lo-spring-boot/index.html

Tomcat   
Tomcat为Spring Boot的默认容器，下面是几个常用配置：

**[java]** [view plaincopyprint?](http://www.cnblogs.com/duyinqiang/p/5696342.html)

1. # tomcat最大线程数，默认为200
2. server.tomcat.max-threads=800
3. # tomcat的URI编码
4. server.tomcat.uri-encoding=UTF-8
5. # 存放Tomcat的日志、Dump等文件的临时文件夹，默认为系统的tmp文件夹（如：C:\Users\Shanhy\AppData\Local\Temp）
6. server.tomcat.basedir=H:/springboot-tomcat-tmp
7. # 打开Tomcat的Access日志，并可以设置日志格式的方法：
8. #server.tomcat.access-log-enabled=**true**
9. #server.tomcat.access-log-pattern=
10. # accesslog目录，默认在basedir/logs
11. #server.tomcat.accesslog.directory=
12. # 日志文件目录
13. logging.path=H:/springboot-tomcat-tmp
14. # 日志文件名称，默认为spring.log
15. logging.file=myapp.log

Jetty   
如果你要选择Jetty，也非常简单，就是把pom中的tomcat依赖排除，并加入Jetty容器的依赖，如下：

<dependencies>

<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>

<dependencies>

打包   
打包方法：   
CMD进入项目目录，使用 mvn clean package 命令打包，以我的项目工程为例：

E:\spring-boot-sample>mvn clean package

可以追加参数**-Dmaven.test.skip=true** (-DskipTests)跳过测试。   
打包后的文件存放于项目下的target目录中，如：spring-boot-sample-0.0.1-SNAPSHOT.jar   
如果pom配置的是war包，则为spring-boot-sample-0.0.1-SNAPSHOT.war

**二、部署到JavaEE容器**

1. 修改启动类，继承 **SpringBootServletInitializer** 并重写 configure 方法

public class SpringBootSampleApplication extends SpringBootServletInitializer{

private static final Logger logger = LoggerFactory.getLogger(SpringBootSampleApplication.class);

@Override

protected SpringApplicationBuilder **configure**(SpringApplicationBuilder builder) {

return builder.sources(this.getClass());

}

}

1. 修改pom文件中jar 为 war

<!-- <packaging>jar</packaging> -->

<packaging>war</packaging>

1. 修改pom，排除**tomcat插件**

<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>

1. 打包部署到容器   
   使用命令 mvn clean package 打包后，同一般J2EE项目一样部署到web容器。

**三、使用Profile区分环境**

spring boot 可以在 “配置文件”、“Java代码类”、“日志配置” 中来配置profile区分不同环境执行不同的结果

1、配置文件   
使用配置文件application.yml 和 application.properties 有所区别   
以application.properties 为例，通过文件名来区分环境 **application-{profile}.properties**   
application.properties

app.name=MyApp

server.port=8080

spring.profiles.active=dev

application-dev.properties  
server.port=8081

application-stg.properties  
server.port=8082

在启动程序的时候通过添加 –spring.profiles.active=**{profile}** 来指定具体使用的配置   
例如我们执行 java -jar demo.jar **–spring.profiles.active=dev** 那么上面3个文件中的内容将被如何应用？   
Spring Boot 会先加载默认的配置文件，然后使用具体指定的profile中的配置去覆盖默认配置。

app.name 只存在于默认配置文件 application.properties 中，因为指定环境中不存在同样的配置，所以该值不会被覆盖

server.port 默认为8080，但是我们指定了环境后，将会被覆盖。如果指定stg环境，server.port 则为 8082

**spring.profiles.active** 默认指定dev环境，如果我们在运行时指定 –spring.profiles.active=stg 那么将应用stg环境，最终 server.port 的值为8082

2、Java类中@Profile注解   
下面2个不同的类实现了同一个接口，@Profile注解指定了具体环境

// 接口定义

public interface SendMessage {

// 发送短信方法定义

public void send();

}

// Dev 环境实现类

@Component

@Profile("dev")

public class DevSendMessage implements SendMessage {

@Override

public void send() {

System.out.println(">>>>>>>>Dev Send()<<<<<<<<");

}

}

// Stg环境实现类

@Component

@Profile("stg")

public class StgSendMessage implements SendMessage {

@Override

public void send() {

System.out.println(">>>>>>>>Stg Send()<<<<<<<<");

}

}

// 启动类

@SpringBootApplication

public class ProfiledemoApplication {

@Value("${app.name}")

private String name;

@Autowired

private SendMessage sendMessage;

@PostConstruct

public void init(){

sendMessage.send();// 会根据profile指定的环境实例化对应的类

}

}

3、logback-spring.xml也支持有节点来支持区分

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<include resource="org/springframework/boot/logging/logback/base.xml" />

<logger name="org.springframework.web" level="INFO"/>

<springProfile name="default">

<logger name="org.springboot.sample" level="TRACE" />

</springProfile>

<springProfile name="dev">

<logger name="org.springboot.sample" level="DEBUG" />

</springProfile>

<springProfile name="staging">

<logger name="org.springboot.sample" level="INFO" />

</springProfile>

</configuration>

再说一遍文件名不要用logback.xml 请使用logback-spring.xml

**四、指定外部的配置文件**

有些系统，关于一些数据库或其他第三方账户等信息，由于安全问题，其配置并不会提前配置在项目中暴露给开发人员。   
对于这种情况，我们在运行程序的时候，可以通过参数指定一个外部配置文件。   
以 demo.jar 为例，方法如下：

java -jar demo.jar **--spring.config.location**=/opt/config/application.properties

其中文件名随便定义，无固定要求。

**五、创建一个Linux 应用的sh脚本**

下面几个脚本仅供参考，请根据自己需要做调整   
start.sh

#!/bin/sh

rm -f tpid

nohup java -jar /data/app/myapp.jar **--spring.profiles.active**=stg > /dev/null 2>&1 &

echo $! > tpid

stop.sh  
  
tpid=`cat tpid | awk '{print $1}'`

tpid=`ps -aef *| grep $tpid | awk '{print $2}' |grep $tpid`*

*if* [ *${tpid} ]; then*

*kill -9 $tpid*

*fi*

check.sh

tpid=`cat tpid | awk '{print $1}'`

tpid=`ps -aef | grep $tpid | awk '{print $2}' |grep $tpid`

if [ ${tpid} ]; then

echo App is running.

else

echo App is NOT running.

fi

kill.sh

#!/bin/sh  
# kill -9 `ps -ef|grep 项目名称|awk '{print $2}'`  
kill -9 `ps -ef|grep demo|awk '{print $2}'`

http://www.cnblogs.com/duyinqiang/p/5696342.html