



运维实用篇

1. 打包与运行

要使得SpringBoot项目打包成可执行jar，需要添加maven插件，

```
56     <build>
57         <plugins>
58             <plugin>
59                 <groupId>org.springframework.boot</groupId>
60                 <artifactId>spring-boot-maven-plugin</artifactId>
61             </plugin>
62         </plugins>
63     </build>
```

由此打出的jar，定义了主类和启动类，可以直接启动。

Linux下，后台运行命令，

```
nohup java -jar springboot_08_ssm-0.0.1-SNAPSHOT.jar > server.log 2>&1 &
```

ubuntu下配置oracle jdk8步骤

1. Oracle官网下载jdk8的tar包（命令行模式jdk8已无法安装）
2. 解压tar包至/usr/lib/jvm路径，
tar -zxvf ~/jdk-8u202-linux-x64.tar.gz -C /usr/lib/jvm
3. 添加环境变量，vim ~/.bashrc，

```
export JAVA_HOME=/usr/lib/jvm/jdk1.8.0_202
export JRE_HOME=${JAVA_HOME}/jre
export CLASSPATH=.:${JAVA_HOME}/lib:${JRE_HOME}/lib
export PATH=${JAVA_HOME}/bin:$PATH
```

4. 刷新环境变量，source ~/.bashrc
5. 添加
java
， sudo update-alternatives --install /usr/bin/java java /usr/lib/java/jdk1.8.0_202/bi
， 其中，300是优先级

2. 配置高级

启动时, 修改端口,

```
nohup java -jar springboot_08_ssm-0.0.1-SNAPSHOT.jar --server.port=8080> server.log
2>&1 &
```

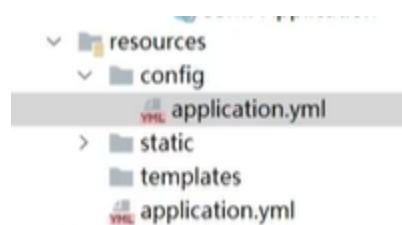
参考, [各项配置优先级](#)

1. Devtools global settings properties on your home directory (`~/spring-boot-devtools.properties` when devtools is active).
2. `@TestPropertySource` annotations on your tests.
3. `properties` attribute on your tests. Available on `@SpringBootTest` and the test annotations for testing a particular slice of your application.
4. Command line arguments.
5. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
6. `ServletConfig` init parameters.
7. `ServletContext` init parameters.
8. JNDI attributes from `java:comp/env`.
9. Java System properties (`System.getProperties()`).
10. OS environment variables.
11. A `RandomValuePropertySource` that has properties only in `random.*`.
12. Profile-specific application properties outside of your packaged jar (`application-{profile}.properties` and YAML variants).
13. Profile-specific application properties packaged inside your jar (`application-{profile}.properties` and YAML variants).
14. Application properties outside of your packaged jar (`application.properties` and YAML variants).
15. Application properties packaged inside your jar (`application.properties` and YAML variants).
16. `@PropertySource` annotations on your `@Configuration` classes.
17. Default properties (specified by setting `SpringApplication.setDefaultProperties`).

优先级由低到高。

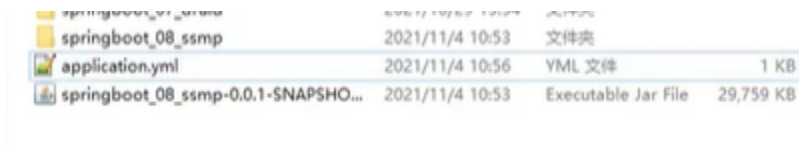
3. 四级配置文件

在resources下, 创建文件夹config, 可以存放配置文件, 优先级高于resources根目录下的配置文件, 可以作为统一的配置管理。



在正式上线时, 诸如数据库密码等是不可知的, 那应如何作配置?

SpringBoot项目的jar包会自动读取与它同目录的配置文件，可以做诸如数据库密码等的配置管理。



springboot_08_ssm	2021/11/4 10:53	文件夹	
application.yml	2021/11/4 10:56	YML 文件	1 KB
springboot_08_ssm-0.0.1-SNAPSHOT.jar	2021/11/4 10:53	Executable Jar File	29,759 KB

更高级别的配置管理，可以存放在jar包同路径下的config文件夹下，由此形成了四级配置文件管理。

1. SpringBoot中4级配置文件

1级: file : config/application.yml **【最高】**

2级: file : application.yml

3级: classpath: config/application.yml

4级: classpath: application.yml **【最低】**

2. 作用:

- ◆ 1级与2级留做系统打包后设置通用属性，1级常用于运维经理进行线上整体项目部署方案调控
- ◆ 3级与4级用于系统开发阶段设置通用属性，3级常用于项目经理进行整体项目属性调控

4. 自定义配置文件

SpringBoot可以自定义配置文件名，

在启动参数中传入spring.config.name，指定配置文件名，

```
--spring.config.name=books
```

CLI arguments to your application. Alt+R

通过启动参数spring.config.location，指定配置文件路径，

```
--spring.config.location=classpath:/books.yml
```

多个配置可以以逗号分隔。

多服务器项目使用自定义配置文件需求较高，将所有配置放置在一个目录中，统一管理。基于

SpringCloud技术，所有的服务器将不再设置配置文件，而是通过配置中心进行设定，动态加载配置信息。

5. 多环境开发

生产环境、测试环境和开发环境配置不同，如何在多环境下进行配置管理呢？

配置文件可使用如下配置，active用于指定应用的环境配置，on-profile用于声明环境名称，

```
# 应用环境
spring:
  profiles:
    active: test
---
# 生产
spring:
  config:
    activate:
      on-profile: prod
server:
  port: 80
---
# 开发
spring:
  config:
    activate:
      on-profile: dev
server:
  port: 81
---
# 测试
spring:
  config:
    activate:
      on-profile: test
server:
  port: 82
```

也可以在application配置文件的基础上，额外添加多个配置文件application-name，这个name可以自定义，在application中指定active为name即可。

properties文件的多环境配置，仅支持多文件类型。

另外，可以按功能将配置文件中的信息拆分为多个配置文件，例如

- application-devDB.yml
- application-devRedis.yml
- ...

然后，再主配置文件中使用spring-profiles-include声明其他配置文件，如下，

```
spring:
  profiles:
    active: dev
    include: devDB,devRedis,devMVC
```

这样的配置，使用多组的话，修改环境配置会很麻烦，因此，SpringBoot有了新的配置，

- 从SpringBoot2.4版开始使用group属性替代include属性，降低了配置书写量
- 使用group属性定义多种主环境与子环境的包含关系

```
spring:
  profiles:
    active: dev
    group:
      "dev": devDB,devRedis,devMVC
      "pro": proDB,proRedis,proMVC
      "test": testDB,testRedis,testMVC
```

Maven也可以设置多环境，并且优先级是高于再SpringBoot中的设置的，

```

<profiles>
  <profile>
    <id>dev_env</id>
    <properties>
      <profile.active>dev</profile.active>
    </properties>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
  </profile>
  <profile>
    <id>pro_env</id>
    <properties>
      <profile.active>pro</profile.active>
    </properties>
  </profile>
  <profile>
    <id>test_env</id>
    <properties>
      <profile.active>test</profile.active>
    </properties>
  </profile>
</profiles>

```

SpringBoot引用Maven属性,

```

spring:
  profiles:
    active: @profile.active@

```

执行完成Maven打包后, 可以看到jar包中的配置信息, 已经按照Maven配置修改了。

6. 日志

日志级别默认是INFO, 开启DEBUG级别, 需要在配置文件声明,

```

logging:
  level:
    root: debug

```

可以对某个包设置日志级别, 另外, 也可以将包分组, 对组设置日志级别, 如下,

```
logging:
    # 设置日志组
    group:
        # 自定义组名，设置当前组中所包含的包
        ebank: com.itheima.controller
    level:
        root: warn
        # 为对应组设置日志级别
        ebank: debug
        # 为对包设置日志级别
        com.itheima.controller: debug
```

日志格式自定义,

- 设置日志输出格式

```
logging:
    pattern:
        console: "%d - %m%n"
```

- ◆ %d: 日期
- ◆ %m: 消息
- ◆ %n: 换行

日志文件系统的设置,

- 设置日志文件

```
logging:
  file:
    name: server.log
```

- 日志文件详细配置

```
logging:
  file:
    name: server.log
  logback:
    rollingpolicy:
      max-file-size: 3KB
      file-name-pattern: server.%d{yyyy-MM-dd}.%i.log
```