Spring Boot

运维实用篇

黑马程序员
www.itheima.com

传智教育旗下
高端IT教育品牌

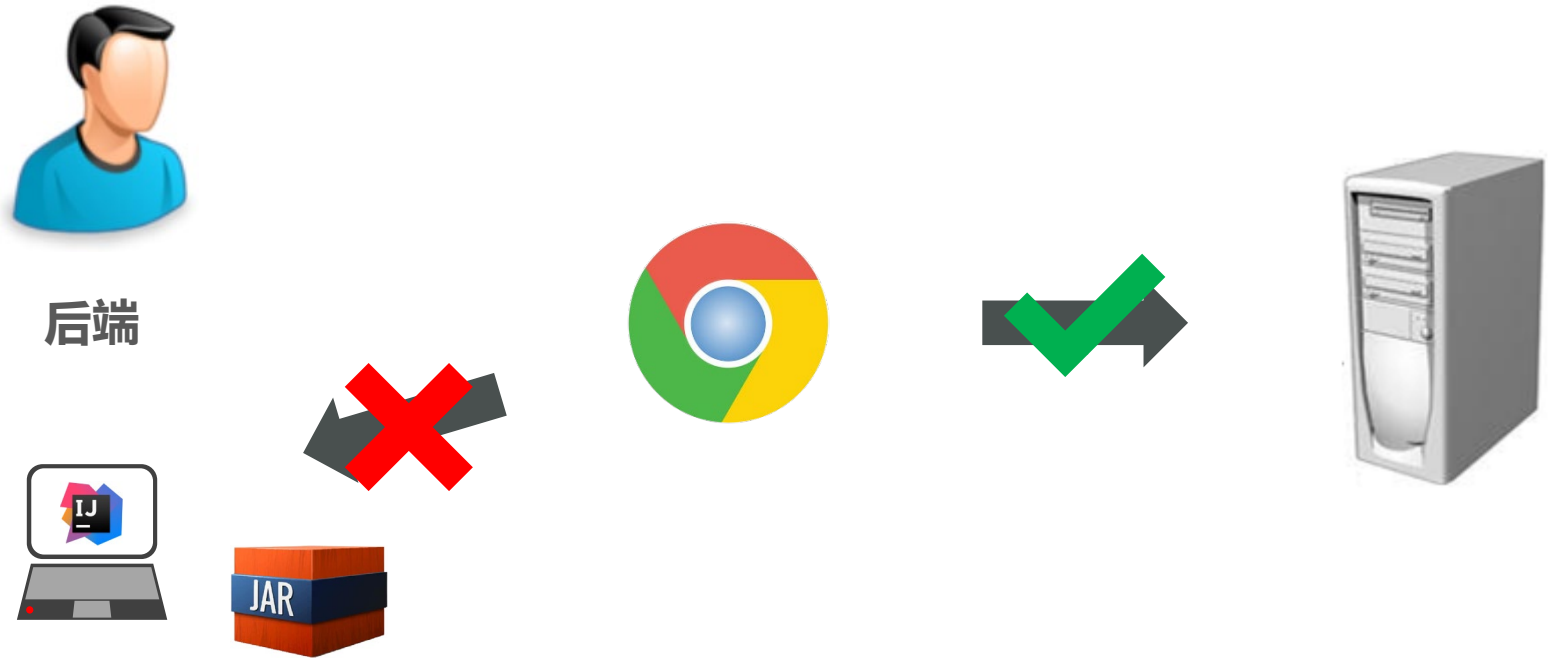
目录

Contents

打包与运行

- 程序打包与运行（Windows版）
- 程序运行（Linux版）

程序为什么要打包



步骤 SpringBoot项目快速启动（Windows版）

①：对SpringBoot项目打包（执行Maven构建指令package）

```
mvn package
```

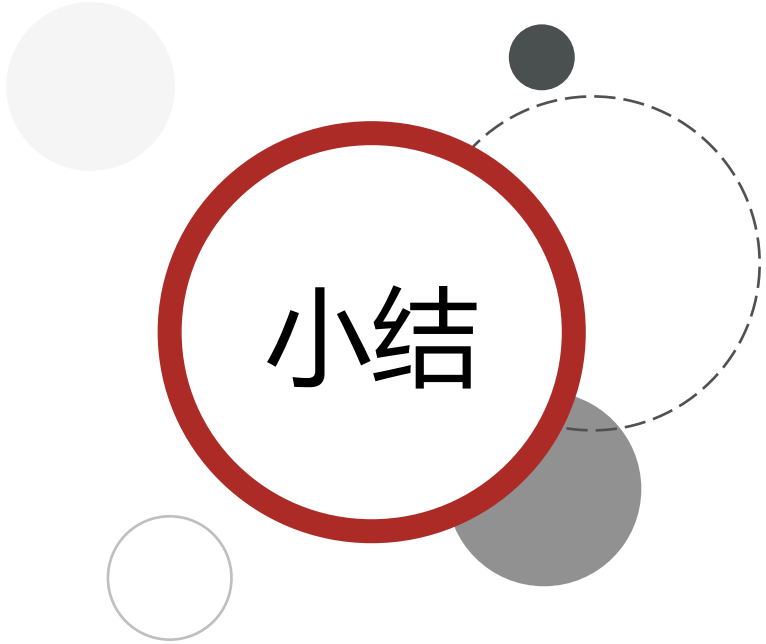
②：运行项目（执行启动指令）

```
java -jar springboot.jar
```

注意事项

jar支持命令行启动需要依赖maven插件支持，请确认打包时是否具有SpringBoot对应的maven插件

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```



1. SpringBoot工程可以基于java环境下独立运行jar文件启动服务
2. SpringBoot工程执行mvn命令package进行打包
3. 执行jar命令：java -jar 工程名.jar

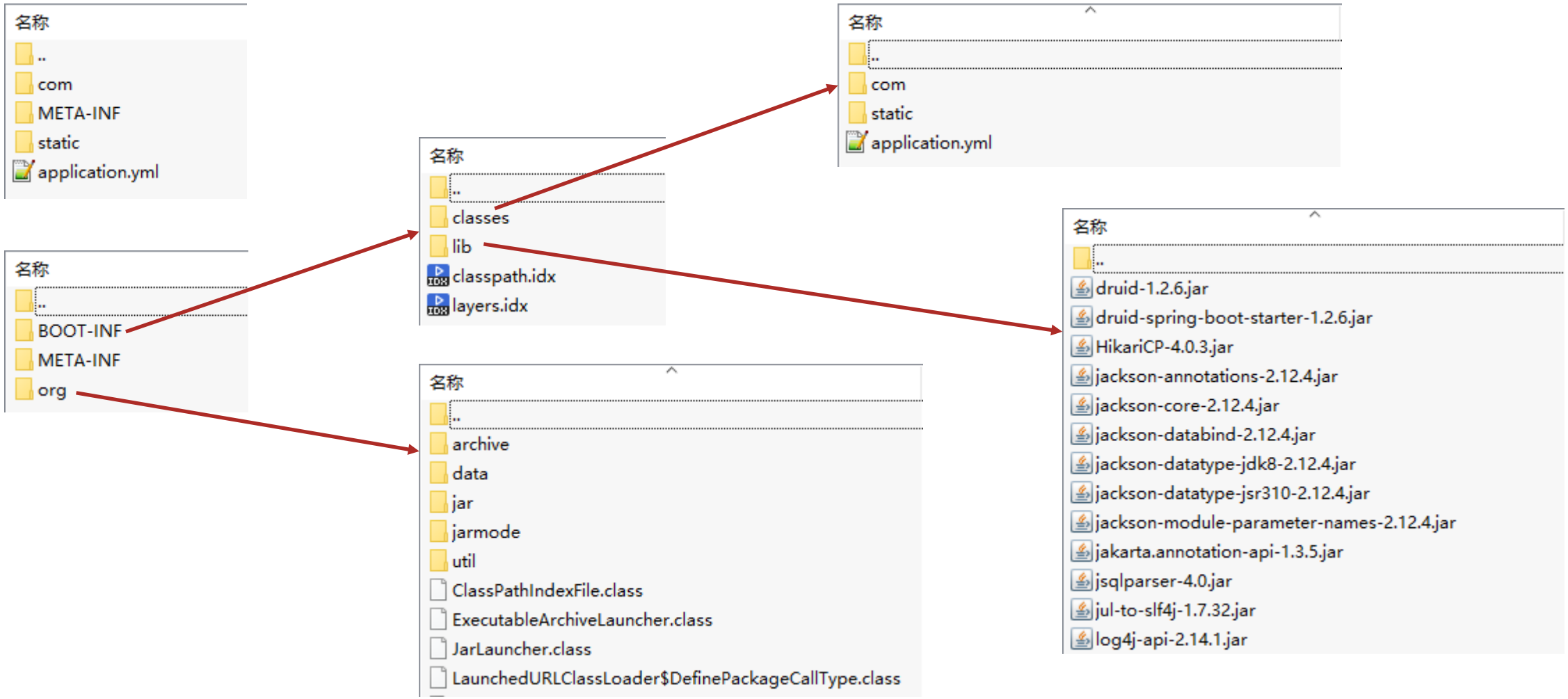
SpringBoot项目快速启动

SpringBoot打包插件

- 使用SpringBoot提供的maven插件可以将工程打包成可执行jar包

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

可执行jar包目录结构



jar包描述文件（MANIFEST.MF）

- 普通工程

```
Manifest-Version: 1.0
Implementation-Title: springboot_08_ssmp
Implementation-Version: 0.0.1-SNAPSHOT
Build-Jdk-Spec: 1.8
Created-By: Maven Jar Plugin 3.2.0
```

- 基于spring-boot-maven-plugin打包的工程

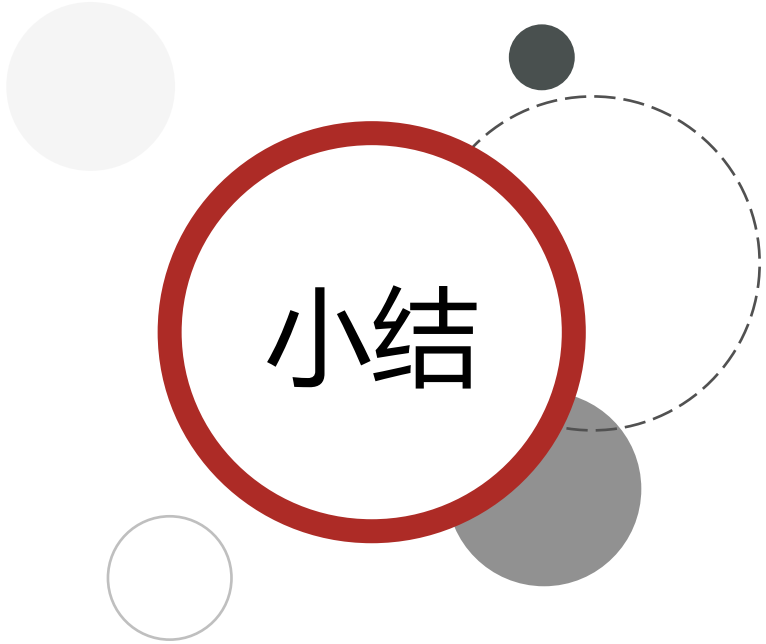
```
Manifest-Version: 1.0
Spring-Boot-Classpath-Index: BOOT-INF/classpath.idx
Implementation-Title: springboot_08_ssmp
Implementation-Version: 0.0.1-SNAPSHOT
Spring-Boot-Layers-Index: BOOT-INF/layers.idx
Start-Class: com.itheima.SSMPApplication
Spring-Boot-Classes: BOOT-INF/classes/
Spring-Boot-Lib: BOOT-INF/lib/
Build-Jdk-Spec: 1.8
Spring-Boot-Version: 2.5.4
Created-By: Maven Jar Plugin 3.2.0
Main-Class: org.springframework.boot.loader.JarLauncher
```

jar启动器

命令行启动常见问题及解决方案

- Windonws端口被占用

```
# 查询端口
netstat -ano
# 查询指定端口
netstat -ano |findstr "端口号"
# 根据进程PID查询进程名称
tasklist |findstr "进程PID号"
# 根据PID杀死任务
taskkill /F /PID "进程PID号"
# 根据进程名称杀死任务
taskkill -f -t -im "进程名称"
```

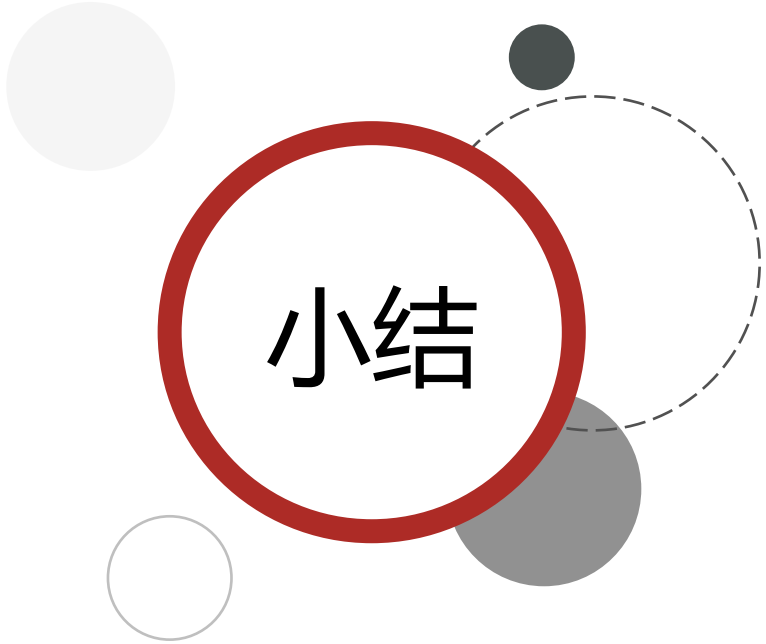


1. spring-boot-maven-plugin插件作用

SpringBoot项目快速启动

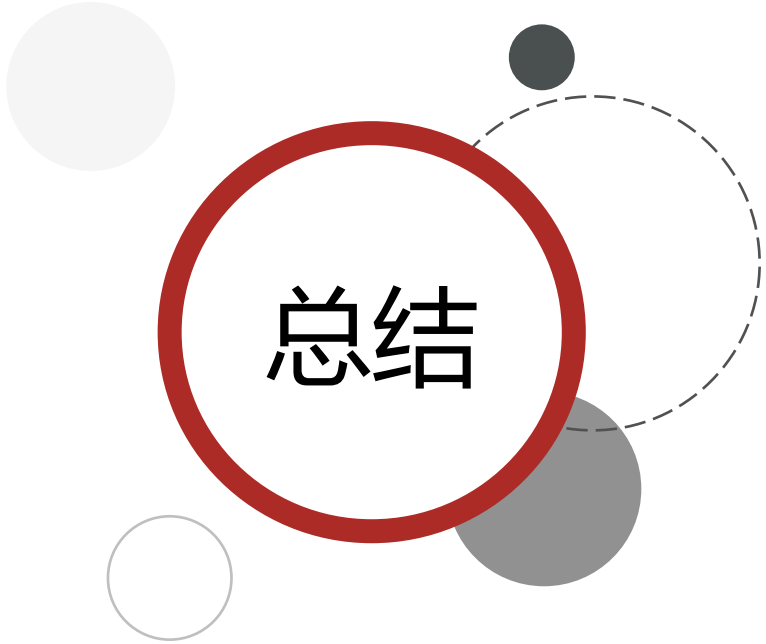
SpringBoot项目快速启动（Linux版）

- 基于Linux（CenterOS7）
- 安装JDK，且版本不低于打包时使用的JDK版本
- 安装包保存在/usr/local/自定义目录中或$HOME下
- 其他操作参照Windows版进行



1．上传安装包

2．执行jar命令：java -jar 工程名.jar



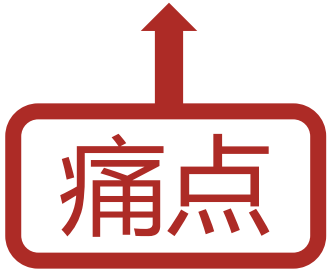
1. Boot程序打包依赖SpringBoot对应的Maven插件即可打包出可执行的jar包
2. 运行jar包使用jar命令进行
3. Windows与Linux下执行Boot打包程序流程相同，仅需确保运行环境有效即可

配置高级

- 临时属性设置
- 配置文件分类
- 自定义配置文件

临时属性设置

后端



临时属性设置

- 带属性数启动SpringBoot

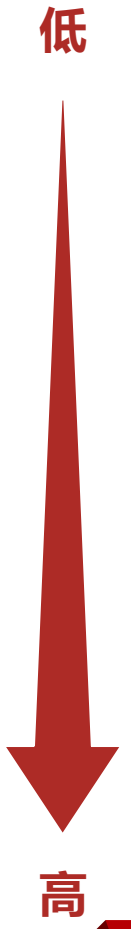
```
java -jar springboot.jar --server.port=80
```

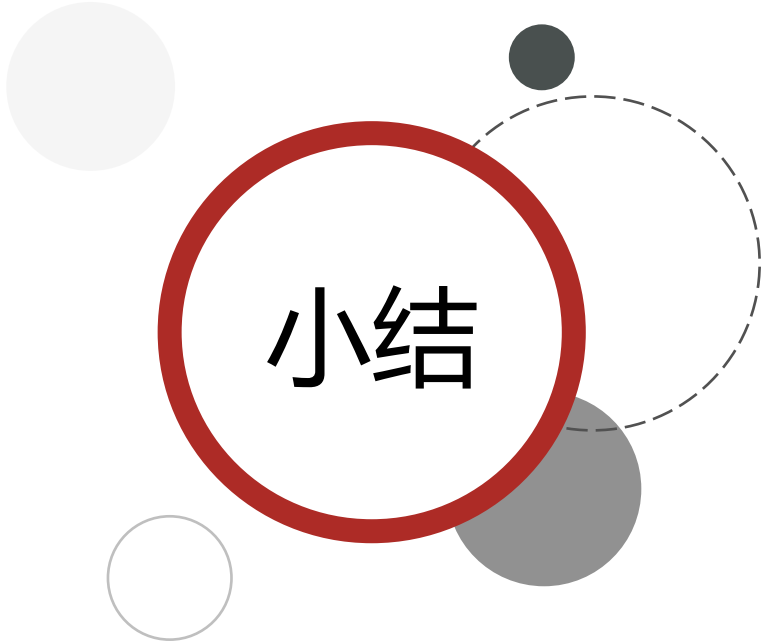
- 携带多个属性启动SpringBoot，属性间使用空格分隔

属性加载优先顺序

1. 参看https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-features.html#boot-features-external-config

1. Default properties (specified by setting `SpringApplication.setDefaultProperties`).
2. `@PropertySource` annotations on your `@Configuration` classes. Please note that such property sources are not added to the `Environment` until the application context is being refreshed. This is too late to configure certain properties such as `logging.*` and `spring.main.*` which are read before refresh begins.
3. Config data (such as `application.properties` files)
4. A `RandomValuePropertySource` that has properties only in `random.*`.
5. OS environment variables.
6. Java System properties (`System.getProperties()`).
7. JNDI attributes from `java:comp/env`.
8. `ServletContext` init parameters.
9. `ServletConfig` init parameters.
10. Properties from `SPRING_APPLICATION_JSON` (inline JSON embedded in an environment variable or system property).
11. Command line arguments.
12. `properties` attribute on your tests. Available on `@SpringBootTest` and the test annotations for testing a particular slice of your application.
13. `@TestPropertySource` annotations on your tests.
14. Devtools global settings properties in the `$HOME/.config/spring-boot` directory when devtools is active.





1．使用jar命令启动SpringBoot工程时可以使用临时属性替换配置文件中的属性

2．临时属性添加方式：java -jar 工程名.jar --属性名=值

3．多个临时属性之间使用空格分隔

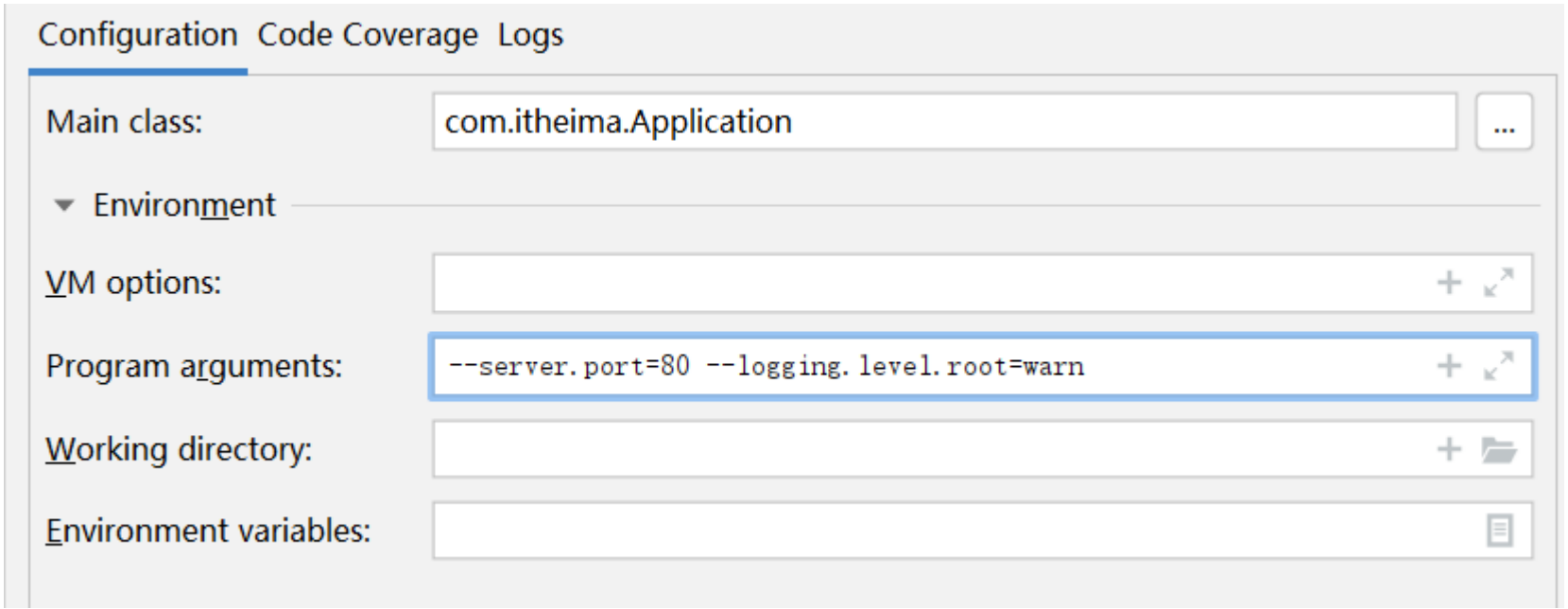
4．临时属性必须是当前boot工程支持的属性，否则设置无效

临时属性设置

后端

临时属性设置（开发环境）

- 带属性启动SpringBoot程序，为程序添加运行属性



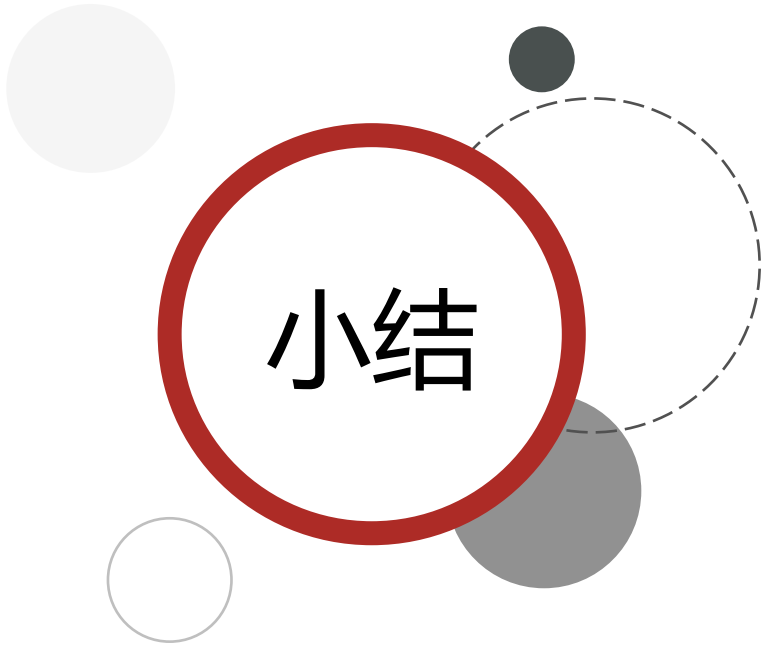
临时属性设置（开发环境）

- 通过编程形式带参数启动SpringBoot程序，为程序添加运行参数

```
public static void main(String[] args) {
    String[] arg = new String[1];
    arg[0] = "--server.port=8080";
    SpringApplication.run(SSMPApplication.class, arg);
}
```

- 不携带参数启动SpringBoot程序

```
public static void main(String[] args) {
    SpringApplication.run(SSMPApplication.class);
}
```



1．启动SpringBoot程序时，可以选择是否使用命令行属性为SpringBoot程序传递启动属性

临时属性设置（开发环境）

经理

研发

```yaml
server:
  port: 6471
spring:
  datasource:
    druid:
      username: admin
      password: &3aX%-46>9S
```

需求不同

```yaml
server:
  port: 80
spring:
  datasource:
    druid:
      username: root
      password: root
```

配置文件分类

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】

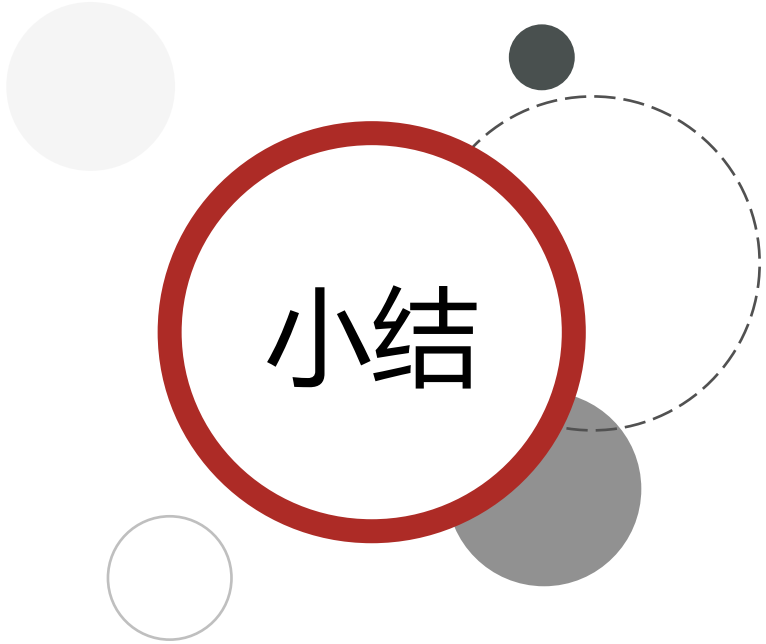
2. 作用：

   - 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

思　考

如果yml与properties在不同层级中共存会是什么效果?

例：类路径application.properties属性是否覆盖文件系统config目录中application.yml属性



1．配置文件分为4种

- 项目类路径配置文件：服务于开发人员本机开发与测试
- 项目类路径config目录中配置文件：服务于项目经理整体调控
- 工程路径配置文件：服务于运维人员配置涉密线上环境
- 工程路径config目录中配置文件：服务于运维经理整体调控

2．多层级配置文件间的属性采用叠加并覆盖的形式作用于程序

配置文件分类

经理

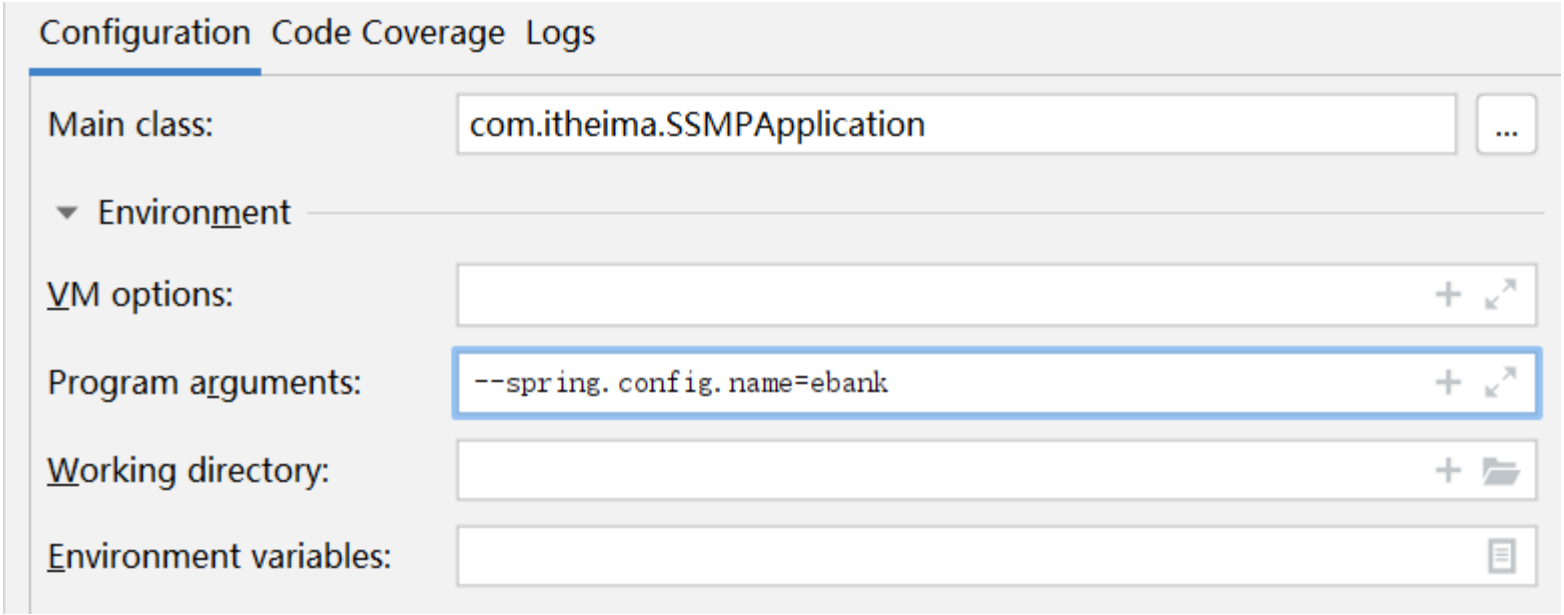
```yaml
server:
  port: 6471
spring:
  datasource:
    druid:
      username: admin
      password: &3aX%-46>9S
```

研发

```yaml
server:
  port: 80
  servlet:
    context-path: /ebank
spring:
  datasource:
    druid:
      username: root
      password: root
```

自定义配置文件

- 通过启动参数加载配置文件（无需书写配置文件扩展名）

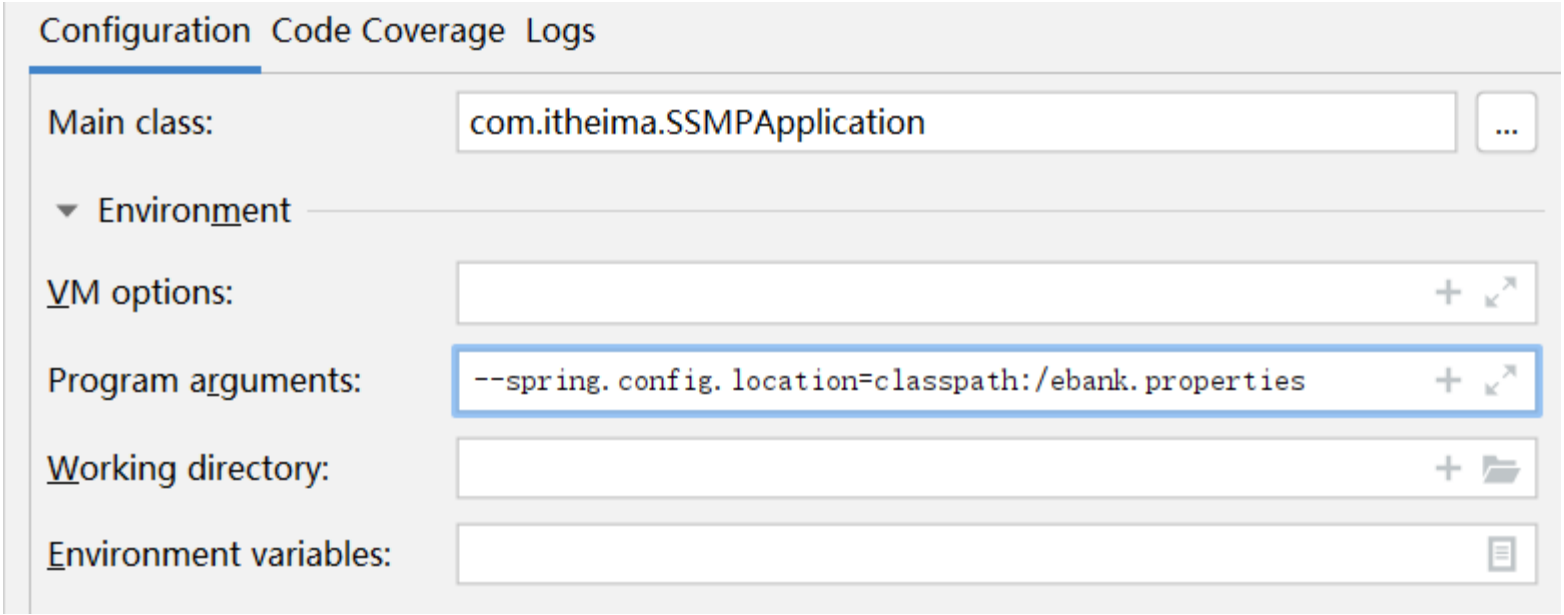


注意事项

properties与yml文件格式均支持

自定义配置文件

- 通过启动参数加载指定文件路径下的配置文件

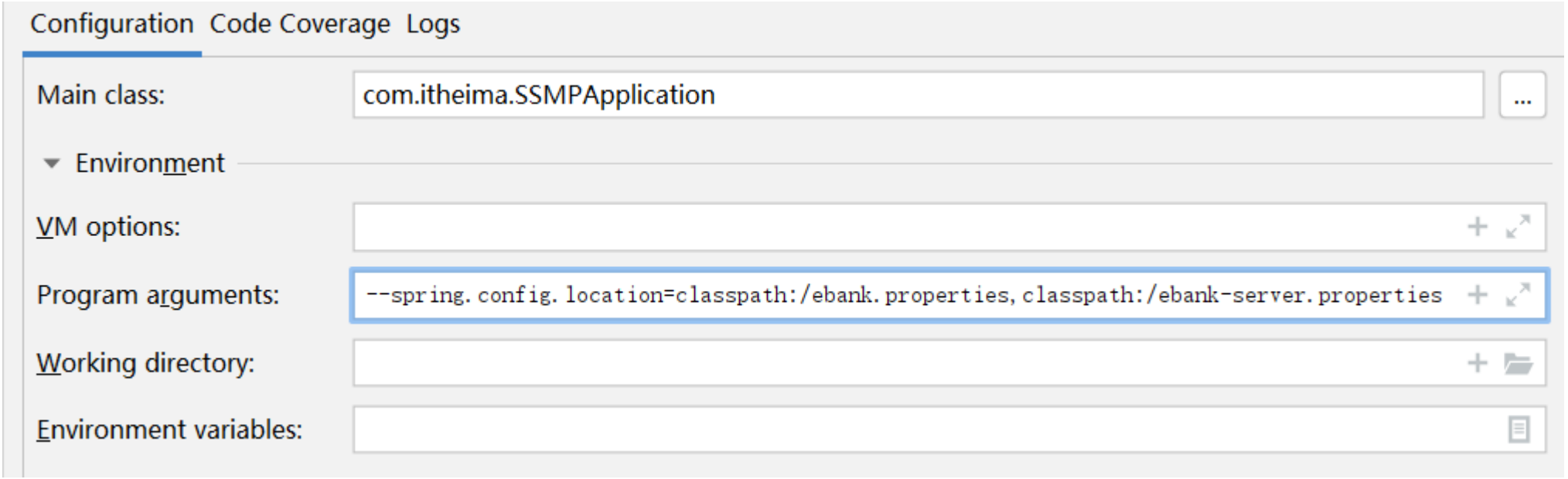


注意事项

properties与yml文件格式均支持

自定义配置文件

- 通过启动参数加载指定文件路径下的配置文件时可以加载多个配置

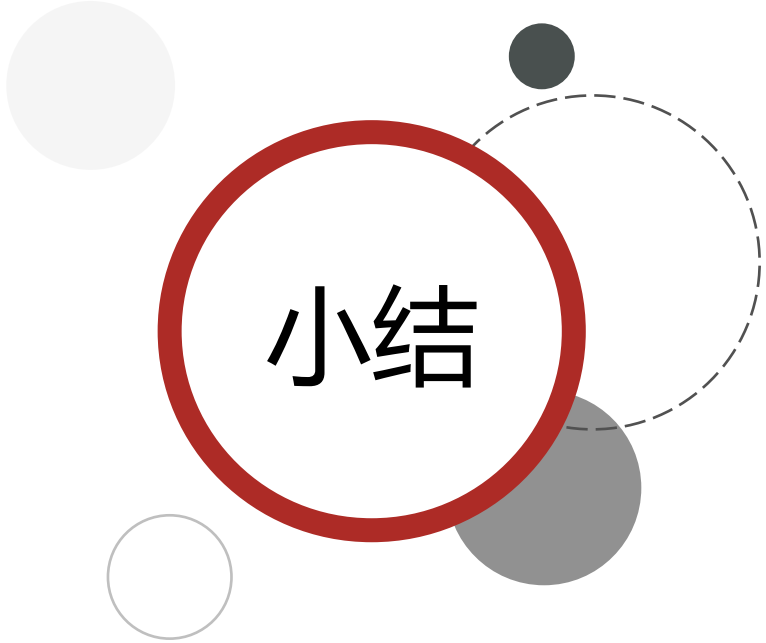


注意事项

多配置文件常用于将配置进行分类，进行独立管理，或将可选配置单独制作便于上线更新维护

自定义配置文件——重要说明

- 单服务器项目：使用自定义配置文件需求较低
- 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



1．配置文件可以修改名称，通过启动参数设定

2．配置文件可以修改路径，通过启动参数设定

3．微服务开发中配置文件通过配置中心进行设置

总结

1. SpringBoot在开发和运行环境均支持使用临时参数修改工程配置
2. SpringBoot支持4级配置文件，应用于开发与线上环境进行配置的灵活设置
3. SpringBoot支持使用自定义配置文件的形式修改配置文件存储位置
4. 基于微服务开发时配置文件将使用配置中心进行管理

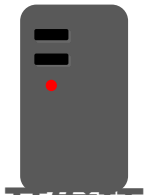
多环境开发

- 多环境开发（YAML版）
- 多环境开发（Properties版）
- 多环境开发控制

多环境

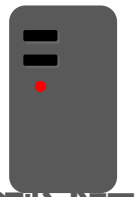
生产环境

```
jdbc:
  url: jdbc:mysql://21.49.35.241:3306/ccb
  user: ccb_admin
  password: 8Fm#_!@aSdm93]4k
```



开发环境

```
jdbc:
  url: jdbc:mysql://127.0.0.1:4092/ccb_svms
  user: root
  password: root
```



测试环境

```
jdbc:
  url: jdbc:mysql://21.49.27.66:4092/ccb_svms
  user: ccb_admin_test
  password: noBUGnoBugnoBUG
```

多环境开发（YAML版）

```yaml
spring:
  profiles:
    active: pro
---
spring:
  profiles: pro

server:
  port: 80
---
spring:
  profiles: dev

server:
  port: 81
---
spring:
  profiles: test

server:
  port: 82
```

启动指定环境

设置生产环境

生产环境具体参数设定

设置开发环境

开发环境具体参数设定

设置测试环境

测试环境具体参数设定

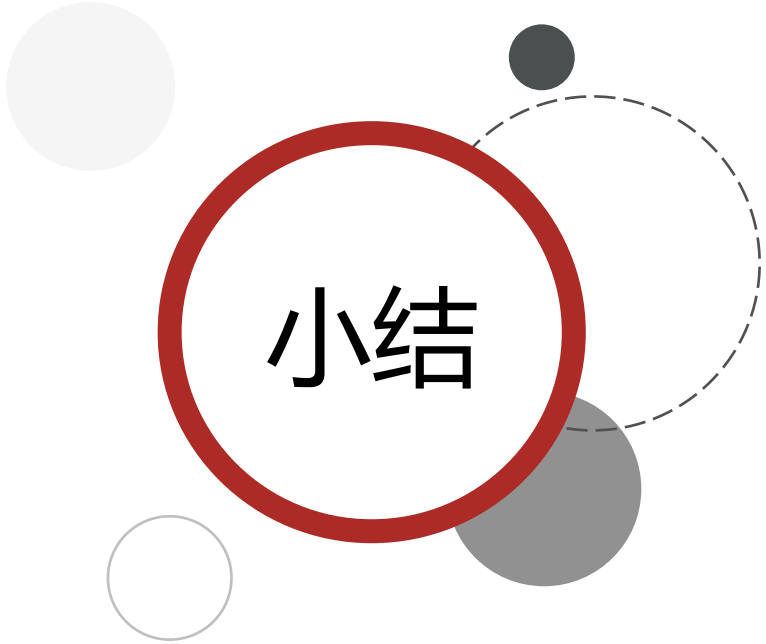
多环境开发（YAML版）

```yaml
spring:
  profiles:
    active: pro
---
server:
  port: 80

spring:
  ~~profiles~~: pro
```

过时格式

```yaml
spring:
  profiles:
    active: pro
---
server:
  port: 80

spring:
  config:
    activate:
      on-profile: pro
```

推荐格式



1．多环境开发需要设置若干种常用环境，例如开发、生产、测试环境

2．yaml格式中设置多环境使用---区分环境设置边界

3．每种环境的区别在于加载的配置属性不同

4．启用某种环境时需要指定启动时使用该环境

多环境开发（YAML版）

```yaml
spring:
  profiles:
    active: pro
---
spring:
  profiles: pro
jdbc:
  url: jdbc:mysql://21.49.35.241:3306/ccb
  user: ccb_admin
  password: 8Fm#_!@aSdm93]4k
---
spring:
  profiles: test
jdbc:
  url: jdbc:mysql://21.49.27.66:4092/ccb_svms
  user: ccb_admin_test
  password: noBUGnoBugnoBUG
```

暴露配置

多环境开发（YAML版）多配置文件格式

1. 主启动配置文件application.yml

```yaml
spring:
  profiles:
    active: dev
```

2. 环境分类配置文件application-**pro**.yml

```yaml
server:
  port: 80
```

3. 环境分类配置文件application-**dev**.yml

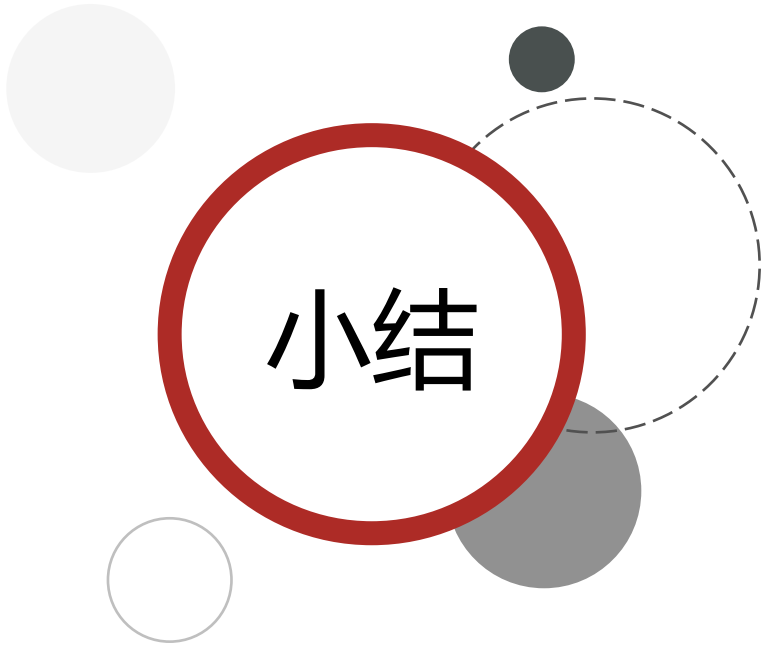
```yaml
server:
  port: 81
```

4. 环境分类配置文件application-**test**.yml

```yaml
server:
  port: 82
```

多环境开发配置文件书写技巧（一）

- 主配置文件中设置公共配置（全局）
- 环境分类配置文件中常用于设置冲突属性（局部）



1．可以使用独立配置文件定义环境属性

2．独立配置文件便于线上系统维护更新并保障系统安全性

多环境开发（Properties版）多配置文件格式

- 主启动配置文件application.properties

```
spring.profiles.active=pro
```

- 环境分类配置文件application-**pro**.properties

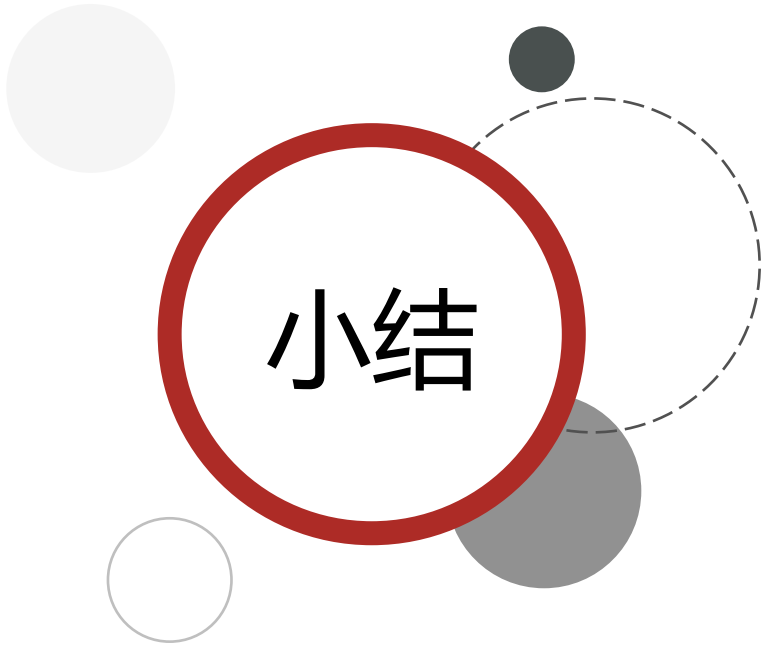
```
server.port=80
```

- 环境分类配置文件application-**dev**.properties

```
server.port=81
```

- 环境分类配置文件application-**test**.properties

```
server.port=82
```



1. properties文件多环境配置仅支持多文件格式

多环境开发独立配置文件书写技巧（二）

- 根据功能对配置文件中的信息进行拆分，并制作成独立的配置文件，命名规则如下
  - application-devDB.yml
  - application-devRedis.yml
  - application-devMVC.yml
- 使用include属性在激活指定环境的情况下，同时对多个环境进行加载使其生效，多个环境间使用逗号分隔

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

注意事项

当主环境dev与其他环境有相同属性时，主环境属性生效；其他环境中有相同属性时，最后加载的环境属性生效

多环境开发独立配置文件书写技巧（二）

- 从Spring2.4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

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

1．多环境开发使用group属性设置配置文件分组，便于线上维护管理

多环境开发控制

profile

profile

步骤 Maven与SpringBoot多环境兼容

①：Maven中设置多环境属性

```
<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>
```

步骤 Maven与SpringBoot多环境兼容

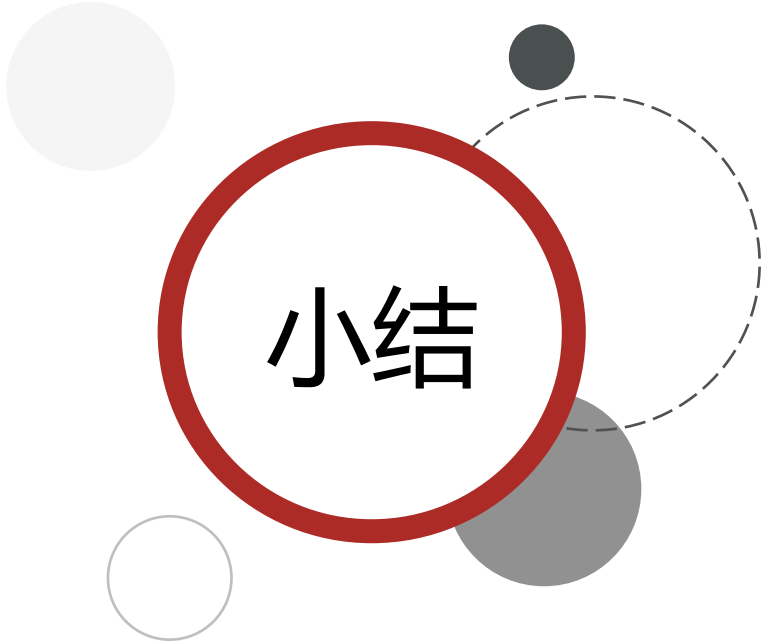
②：SpringBoot中引用Maven属性

```yaml
spring:
  profiles:
    active: @profile.active@
```

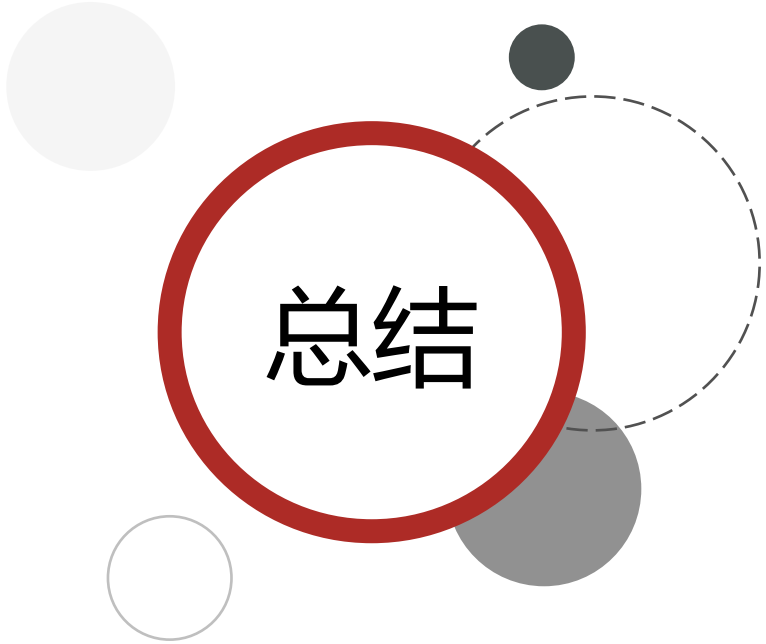
```xml
<profile>
    <id>dev_env</id>
    <properties>
        <profile.active>dev</profile.active>
    </properties>
    <activation>
        <activeByDefault>true</activeByDefault>
    </activation>
</profile>
```

步骤 Maven与SpringBoot多环境兼容

③：执行Maven打包指令，并在生成的boot打包文件.jar文件中查看对应信息



1. 当Maven与SpringBoot同时对多环境进行控制时，以Mavn为主，SpringBoot使用@..@占位符读取Maven对应的配置属性值
2. 基于SpringBoot读取Maven配置属性的前提下，如果在Idea下测试工程时pom.xml每次更新需要手动compile方可生效



1．多环境开发（YAML版）

2．多环境开发（Properties版）

3．Maven与SpringBoot多环境冲突现象解决方案

日志

- 日志基础
- 日志输出格式控制
- 日志文件

日志基础操作

- 日志（log）作用
  - 编程期调试代码
  - 运营期记录信息
    - 记录日常运营重要信息（峰值流量、平均响应时长……）
    - 记录应用报错信息（错误堆栈）
    - 记录运维过程数据（扩容、宕机、报警……）

步骤 代码中使用日志工具记录日志

①：添加日志记录操作

```
@RestController
@RequestMapping("/books")
public class BookController extends BaseController {
    private static final Logger log = LoggerFactory.getLogger(BookController.class);
    @GetMapping
    public String getById(){
        System.out.println("springboot is running...");
        log.debug("debug ...");
        log.info("info ...");
        log.warn("warn ...");
        log.error("error ...");
        return "springboot is running...";
    }
}
```

日志基础操作

- 日志级别
  - TRACE：运行堆栈信息，使用率低
  - DEBUG：程序员调试代码使用
  - INFO：记录运维过程数据
  - WARN：记录运维过程报警数据
  - ERROR：记录错误堆栈信息
  - FATAL：灾难信息，合并计入ERROR

步骤 代码中使用日志工具记录日志

②：设置日志输出级别

```yaml
# 开启debug模式，输出调试信息，常用于检查系统运行状况
debug: true

# 设置日志级别，root表示根节点，即整体应用日志级别
logging:
  level:
    root: debug
```

步骤 代码中使用日志工具记录日志

③：设置日志组，控制指定包对应的日志输出级别，也可以直接控制指定包对应的日志输出级别

```yaml
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
```

小结

1．日志用于记录开发调试与运维过程消息

2．日志的级别共6种，通常使用4种即可，分别是DEBUG，INFO,WARN,ERROR

3．可以通过日志组或代码包的形式进行日志显示级别的控制

步骤 代码中使用日志工具记录日志

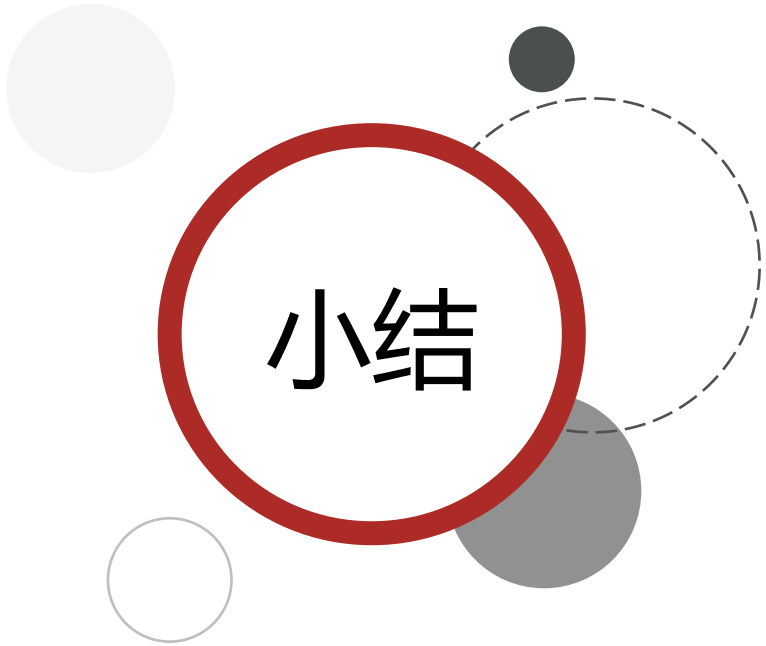
①：Maven中设置多环境属性

```
@RestController
@RequestMapping("/books")
public class BookController extends BaseController {
    private static final Logger log = LoggerFactory.getLogger(BookController.class);
    @GetMapping
    public String getById(){
        System.out.println("springboot is running...");
        log.debug("debug ...");
        log.info("info ...");
        log.warn("warn ...");
        log.error("error ...");
        return "springboot is running...";
    }
}
```

教你一招：优化日志对象创建代码

- 使用lombok提供的注解@Slf4j简化开发，减少日志对象的声明操作

```
@Slf4j
@RestController
@RequestMapping("/books")
public class BookController {
    @GetMapping
    public String getById(){
        System.out.println("springboot is running...");
        log.debug("debug info...");
        log.info("info info...");
        log.warn("warn info...");
        log.error("error info...");
        return "springboot is running...";
    }
}
```



1．基于lombok提供的@Slf4j注解为类快速添加日志对象

日志输出格式控制

```
2021-11-02 12:25:39.392  INFO 2336 --- [           main] com.itheima.Springboot10LogApplication    : Starting Springboot10LogApplication using Java 1.8.0_172 or
2021-11-02 12:25:39.395  INFO 2336 --- [           main] com.itheima.Springboot10LogApplication    : No active profile set, falling back to default profiles: de
2021-11-02 12:25:40.065  INFO 2336 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat initialized with port(s): 8080 (http)
2021-11-02 12:25:40.071  INFO 2336 --- [           main] o.apache.catalina.core.StandardService    : Starting service [Tomcat]
2021-11-02 12:25:40.071  INFO 2336 --- [           main] org.apache.catalina.core.StandardEngine   : Starting Servlet engine: [Apache Tomcat/9.0.54]
2021-11-02 12:25:40.113  INFO 2336 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]        : Initializing Spring embedded WebApplicationContext
2021-11-02 12:25:40.113  INFO 2336 --- [           main] w.s.c.ServletWebServerApplicationContext  : Root WebApplicationContext: initialization completed in 672
2021-11-02 12:25:40.326  INFO 2336 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer   : Tomcat started on port(s): 8080 (http) with context path ''
2021-11-02 12:25:40.334  INFO 2336 --- [           main] com.itheima.Springboot10LogApplication    : Started Springboot10LogApplication in 1.281 seconds (JVM ru
```

时间　　级别 PID　　所属线程　　所属类/接口名　　日志信息

- PID：进程ID，用于表明当前操作所处的进程，当多服务同时记录日志时，该值可用于协助程序员调试程序
- 所属类/接口名：当前显示信息为SpringBoot重写后的信息，名称过长时，简化包名书写为首字母，甚至直接删除

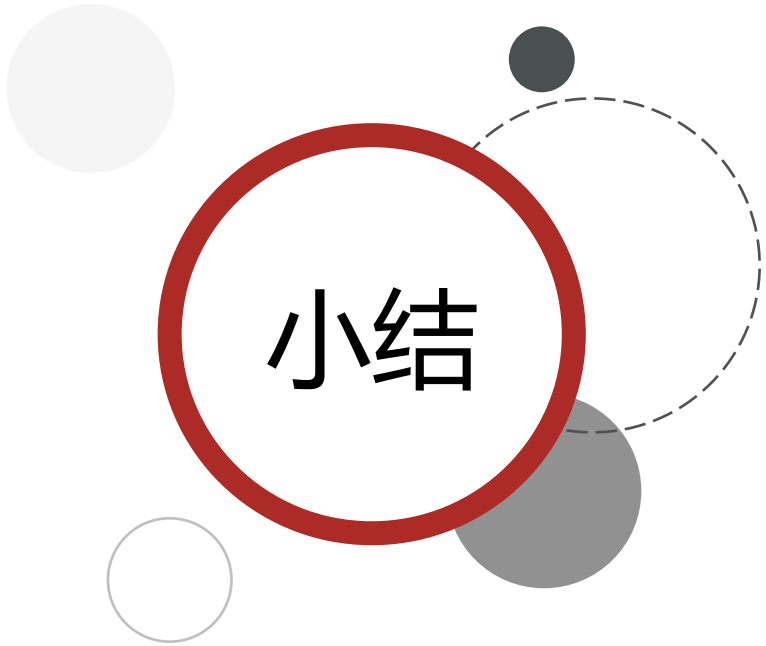
日志输出格式控制

- 设置日志输出格式

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

  - %d：日期
  - %m：消息
  - %n：换行

```yaml
logging:
  pattern:
    console: "%d %clr(%p) --- [%16t] %clr(%-40.40c){cyan} : %m %n"
```



1．日志输出格式设置规则

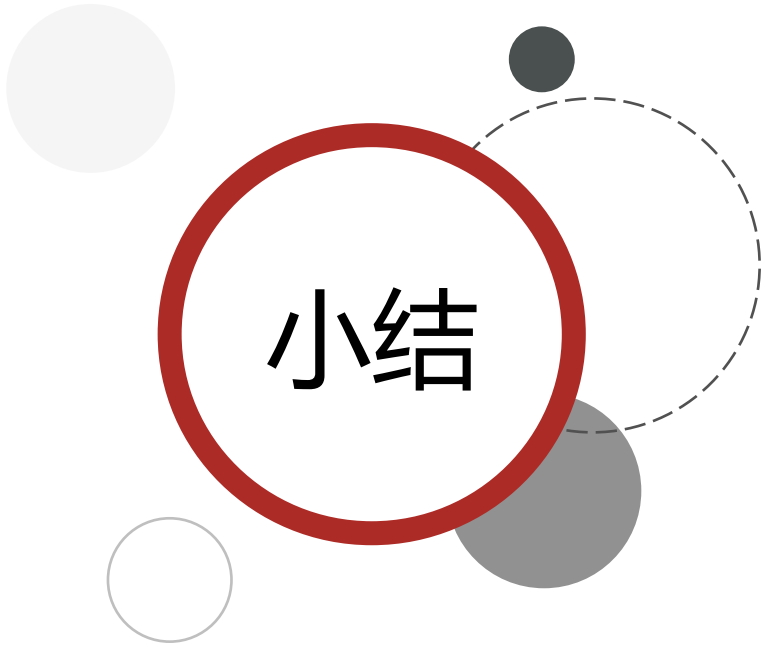
日志文件

- 设置日志文件

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

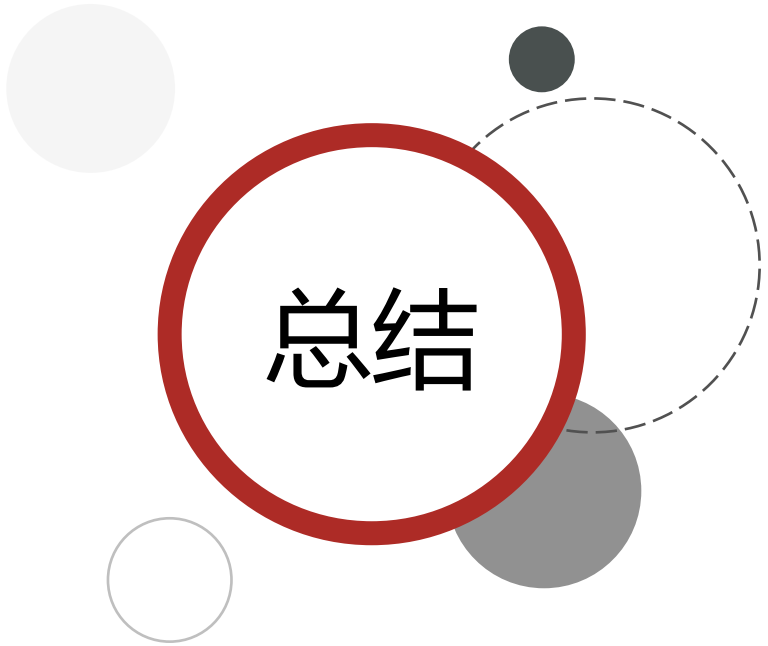
- 日志文件详细配置

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



1．日志记录到文件

2．日志文件格式设置



1．日志基础使用规则

2．编辑日志输出格式

3．日志文件设置

日志基础操作

1. 基础篇
2. 实用篇
   - 运维实用篇
     - 能够掌握SpringBoot程序多环境开发
     - 能够基于Linux系统发布SpringBoot工程
     - 能够解决线上灵活配置SpringBoot工程的需求
   - 开发实用篇
3. 原理篇

日志基础操作

1. 基础篇
2. 实用篇
   - 运维实用篇
     - 能够掌握SpringBoot程序多环境开发
     - 能够基于Linux系统发布SpringBoot工程
     - 能够解决线上灵活配置SpringBoot工程的需求
   - 开发实用篇
     - 能够基于SpringBoot整合任意第三方技术
3. 原理篇

传智教育旗下高端IT教育品牌