第三章 Spring Boot 实战

（Spring Cloud 初级）

一、 Spring Boot 回顾

1 什么是 Spring Boot？

Spring Boot 是在 Spring 的基础之上产生的(确切的说是在 Spring4.0 的版本的基础之上)，其中“Boot”的意思就是“引导”，意在简化开发模式，是开发者能够快速的开发出基于 Spring 的应用。Spring Boot 含有一个内嵌的 web 容器。我们开发的 web 应用不需要作为 war 包部署到 web 容器中，而是作为一个 jar 包，在启动时根据 web 服务器的配置进行加载。

2 在没有使用 Spring Boot 开发时项目时什么样的？

2.1在项目中存在大量的 xml 文件，配置相当繁琐

2.2整合第三方框架时的配置问题

2.3低效的开发效率与部署效率问题

3 Spring Boot 解决了什么？

3.1 Spring Boot 使配置简单

3.2 Spring Boot 使编码加单

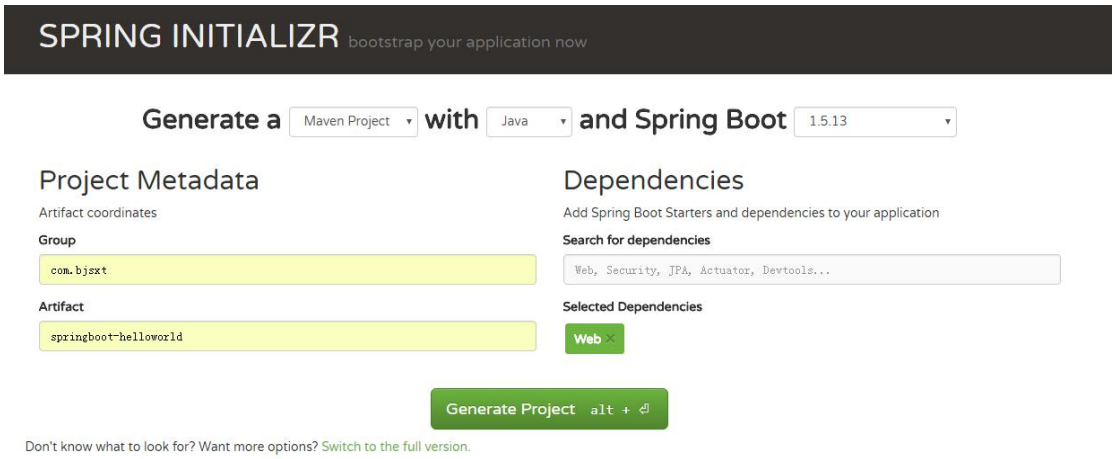
3.3 Spring Boot 使部署简单

3.4 Spring Boot 使监控简单

二、 Spring Boot 快速构建项目

1 打开 Spring Boot 的官网

https://projects.spring.io/spring-boot/
https://start.spring.io/ 构建 Spring Boot 的页面



2 使用 Spring Boot 官网构建项目

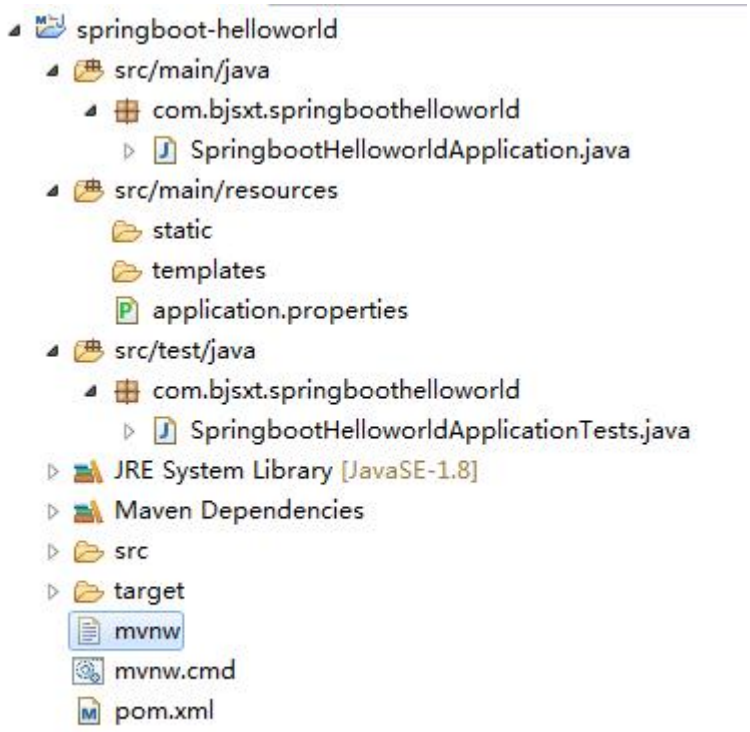
2.1会自动的帮助我们生成启动类

2.2会自动生成存放静态资源的目录，还会生成全局配置文件

2.3会自动生成测试代码，当然只是的一个结构。

2.4Spring Boot 官方推荐的 jdk 版本为 1.8 或者更高

3 构建项目目录结构



三、 Spring Boot 全局配置文件讲解

1 修改内嵌容器的端口号

```
server.port=8888
```

2 自定义属性配置

```
msg=Hello World
```

```
@Value("${msg}")
 private String msg;
```

3 配置变量引用

```
hello=bjsxt
msg=Hello World ${hello}
```

```
@Value("${msg}")
 private String msg;
```

4 随机值配置

4.1配置随机值

```
num=${random.int}
msg=Hello World ${num}
```

```
@Value("${msg}")
 private String msg;
```

用处：配置随机值，在程序中如果有一些运算需要一个随机值，那么可以使用该方式来生成。注意，只生成一次。

4.2配置随机端口

```
server.port=${random.int[1024,9999]}
```

用处：在 SpringCloud 的微服务中，我们是不需要记录 IP 与端口号的。那么我们也就不需要去维护服务的端口号。让他随机生成就可以了。

四、 yml 配置文件

是 Spring Boot 中新增的一种配置文件格式。特点：具备天然的树状结构

1 yml 配置文件与 properties 文件的区别

1.1 配置文件的扩展名有变化

1.2 配置文件中的语法有变化

2 yml 配置文件的语法

2.1 在 properties 文件中是以“.”进行分割，在 yml 中使用“：”进行分割

2.2 yml 的数据格式和 json 的格式很像，都是 K-V 结构的。并且是通过“：”赋值

2.3 在 yml 中缩进一定不能使用 TAB 件，否则会报错。

2.4 每个 K 的冒号后面一定要加一个空格

五、 logback 日志记录讲解

1 导入相关的 jar 包

2 添加 logback.xml 配置文件

```
    <?xml version="1.0" encoding="UTF-8" ?>
     <configuration>
    <!--定义日志文件的存储地址 勿在 LogBack 的配置中使用相对路径-->
        <property name="LOG_HOME" value="${catalina.base}/logs/"
/>
        <!-- 控制台输出 -->
```

```
        <appender name="Stdout"
class="ch.qos.logback.core.ConsoleAppender">
           <!-- 日志输出编码 -->
            <layout
class="ch.qos.logback.classic.PatternLayout">
                <!--格式化输出：%d 表示日期，%thread 表示线程
名，%-5level：级别从左显示 5 个字符宽度%msg：日志消息，%n 是换行符-->
                <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS}
[%thread] %-5level %logger{50} - %msg%n
                </pattern>
            </layout>
        </appender>
        <!-- 按照每天生成日志文件 -->
        <appender name="RollingFile"
class="ch.qos.logback.core.rolling.RollingFileAppender">
            <rollingPolicy
class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
                <!--日志文件输出的文件名-->

<FileNamePattern>${LOG_HOME}/server.%d{yyyy-MM-dd}.log</FileNa
mePattern>
                <MaxHistory>30</MaxHistory>
            </rollingPolicy>
            <layout
class="ch.qos.logback.classic.PatternLayout">
                <!--格式化输出：%d 表示日期，%thread 表示线程
名，%-5level：级别从左显示 5 个字符宽度%msg：日志消息，%n 是换行符-->
                <pattern>%d{yyyy-MM-dd HH:mm:ss.SSS}
[%thread] %-5level %logger{50} - %msg%n
                </pattern>
            </layout>
            <!--日志文件最大的大小-->
            <triggeringPolicy
class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
              <MaxFileSize>10MB</MaxFileSize>
            </triggeringPolicy>
        </appender>

        <!-- 日志输出级别 -->
        <root level="info">
            <appender-ref ref="Stdout" />
            <appender-ref ref="RollingFile" />
        </root>
```

```
    <!--日志异步到数据库 -->
    <!--     <appender name="DB"
class="ch.qos.logback.classic.db.DBAppender">
            日志异步到数据库
            <connectionSource
class="ch.qos.logback.core.db.DriverManagerConnectionSource">
              连接池
              <dataSource
class="com.mchange.v2.c3p0.ComboPooledDataSource">

<driverClass>com.mysql.jdbc.Driver</driverClass>

<url>jdbc:mysql://127.0.0.1:3306/databaseName</url>
                <user>root</user>
                <password>root</password>
              </dataSource>
            </connectionSource>
      </appender> -->

    </configuration>
```

六、 Spring Boot 的配置文件 - 多环境配置

profile:代表的就是一个环境变量
语法结构：application-{profile}.properties

1 需求：

application-dev.properties 开发环境
application-test.properteis 测试环境
application-prod.properteis 生产环境

2 运行项目：

java -jar xxx.jar --spring.profiles.active={profile}

3 完成的命令：

java -jar springboot-helloworld-0.0.1-SNAPSHOT.jar --spring.profiles.active=

test|dev|prod

七、 Spring Boot 核心注解讲解

@SpringBootApplication：代表是 SpringBoot 的启动类。
@SpringBootConfiguration：通过 bean 对象来获取配置信息
@Configuration：通过对 bean 对象的操作替代 spring 中 xml 文件
@EnableAutoConfiguration：完成一些初始化环境的配置。
@ComponentScan：来完成 spring 的组件扫描。替代之前我们在 xml 文件中配置组件扫描的配置<context:component-scan pacage=”....”>
@RestController:1,表示一个 Controller。2，表示当前这个 Controller 下的所有的方法都会以 json 格式的数据响应。

八、 回顾 SpringBoot 异常处理：

@ControllerAdvice+@ExceptionHandler 注解处理异常

代码

```
@ControllerAdvice
public class MyControllerAdvice {

    @ResponseBody
    @ExceptionHandler(value=java.lang.Exception.class)
    public Map<String, Object> myException(Exception ex) {
        Map<String, Object> map = new HashMap<>();
        map.put("code", 500);
        map.put("msg", "出错了。");
        return map;
    }

    @ResponseBody
    @ExceptionHandler(value=java.lang.NullPointerException.class)
    public Map<String, Object> myException2(Exception ex) {
        Map<String, Object> map = new HashMap<>();
        map.put("code", -500);
        map.put("msg", "空指针异常");
        return map;
    }
```

```
        @ResponseBody
        @ExceptionHandler(value=com.bjsxt.springboothelloworld.exception.ApplicationException.class)
        public Map<String, Object> myException3(Exception ex) {
            Map<String, Object> map = new HashMap<>();
            map.put("code", -800);
            map.put("msg", ex.getMessage());
            return map;
        }


    }
```

九、 如何监控 Spring Boot 的健康状况

1 使用 Actuator 检查与监控的步骤

1.1 在 pom 文件中添加 Actuator 的坐标

1.2 在全局配置文件中设置关闭安全限制

ID	描述	是否需要鉴权
actuator	为其他端点提供“发现页面”。要求 Spring HATEOAS 在 classpath 路径上。	需要
auditevents	陈列当前应用程序的审计事件信息。	需要
autoconfig	展示自动配置信息并且显示所有自动配置候选人以及他们“被不被”应用的原因。	需要
beans	显示应用程序中所有 Spring bean 的完整列表。	需要
configprops	显示所有配置信息。	需要
dump	dump 所有线程。	需要

env	陈列所有的环境变量。	需要
flyway	Shows any Flyway database migrations that have been applied.	需要
health	显示应用程序运行状况信息	不需要
info	显示应用信息。	不需要
loggers	显示和修改应用程序中的 loggers 配置。	需要
liquibase	显示已经应用的任何 Liquibase 数据库迁移。	需要
metrics	显示当前应用程序的“指标”信息。	需要
mappings	显示所有@RequestMapping 的 url 整理列表。	需要
shutdown	关闭应用（默认情况下不启用）。	需要
trace	显示跟踪信息（默认最后 100 个 HTTP 请求）。	需要

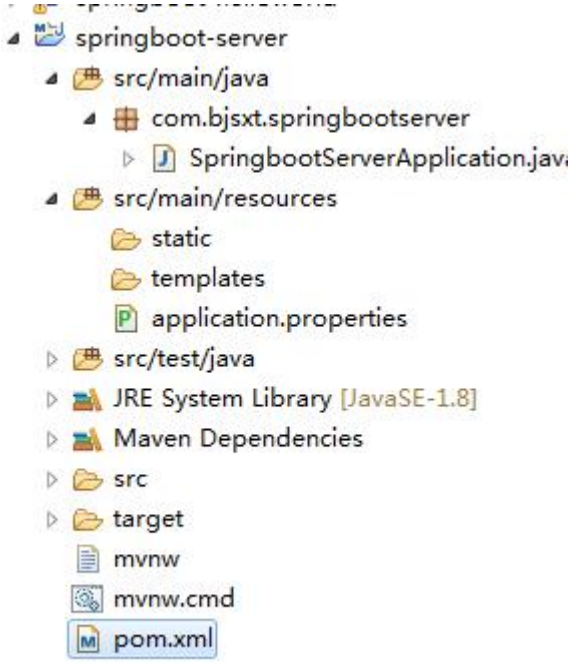
2 使用可视化的监控报表-Spring Boot Admin

2.1使用 Spring Boot Admin 的步骤

2.1.1搭建服务端

服务端其实也是一个 SpringBoot 项目

2.1.1.1 创建项目



2.1.1.2 访问 Spring Boot Admin 的首页

https://github.com/codecentric/spring-boot-admin

修改 pom 文件添加 Spring Boot Admin 坐标

```
<dependency>
        <groupId>de.codecentric</groupId>

<artifactId>spring-boot-admin-starter-server</artifactId>
        <version>1.5.7</version>
    </dependency>
```

2.1.1.3 修改启动类，添加@EnableAdminServer

```
@SpringBootApplication
@EnableAdminServer
public class SpringbootServerApplication {

   public static void main(String[] args) {

SpringApplication.run(SpringbootServerApplication.class,
args);
    }
}
```

2.1.2搭建客户端

其实客户端就是我们需要监控的工程。

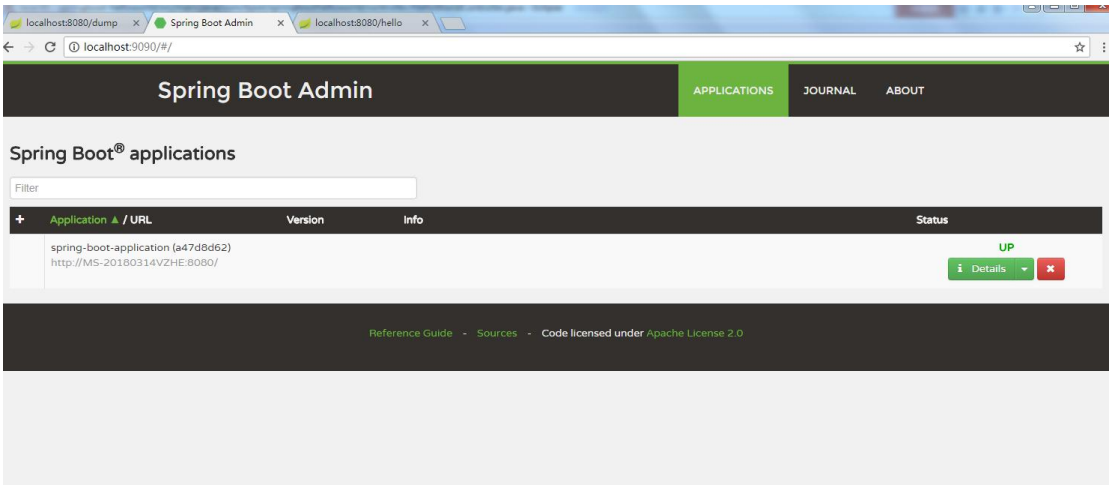
2.1.2.1 修改客户端的 pom 文件添加依赖

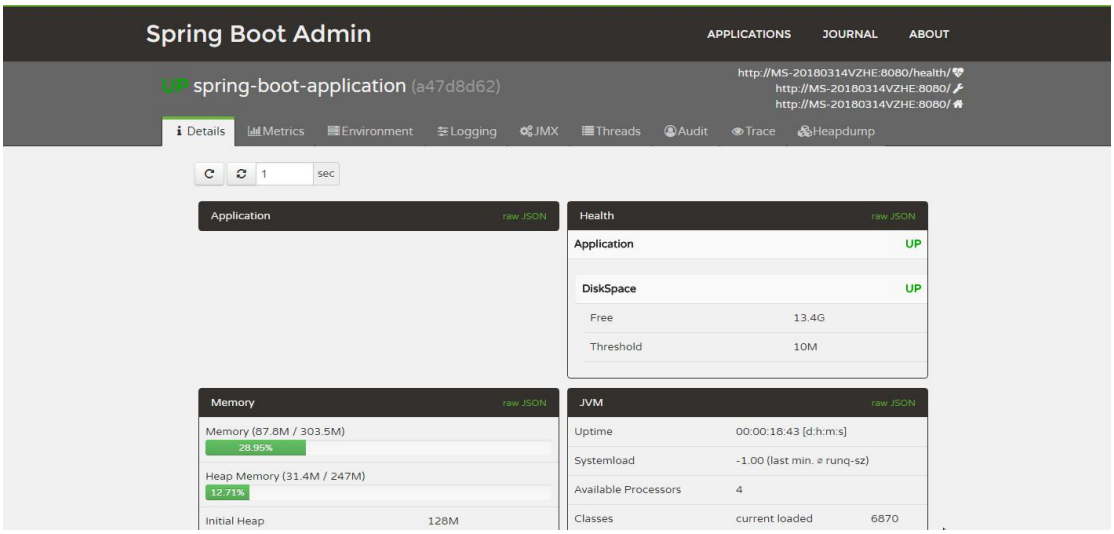
```
    <dependency>
            <groupId>de.codecentric</groupId>

    <artifactId>spring-boot-admin-starter-client</artifactId>
            <version>1.5.7</version>
        </dependency>
```

2.1.2.2 修改客户端的 application.properteis 配置文件

```
    management.security.enabled=false
    #http://localhost:9090 表示是 Spring Boot Admin 服务单的 IP 地址以及端口号
    spring.boot.admin.url: http://localhost:9090
```

2.2 监控信息讲解




Spring Boot Admin
APPLICATIONS
JOURNAL
ABOUT
UP spring-boot-application (a47d8d62)
http://MS-20180314VZHE:8080/health/
http://MS-20180314VZHE:8080/
http://MS-20180314VZHE:8080/
Details
Metrics
Environment
Logging
JMX
Threads
Audit
Trace
Heapdump
1
sec
Application
raw JSON
Health
raw JSON
Application
UP
DiskSpace
UP
Free
13.4G
Threshold
10M
Memory
raw JSON
Memory (87.8M / 303.5M)
28.95%
Heap Memory (31.4M / 247M)
12.71%
Initial Heap
128M
JVM
raw JSON
Uptime
00:00:18:43 [d:h:m:s]
Systemload
-1.00 (last min. ⌀ runq-sz)
Available Processors
4
Classes
current loaded
6870