[第一章 了解微服务 2](#_Toc27308)

[第二章 Spring框架的本质 2](#_Toc27138)

[javaconfig 2](#_Toc9309)

[第三章 Springboot的工作机制 2](#_Toc28833)

[@SpringBootApplication 2](#_Toc27783)

[SpringBoot程序启动 3](#_Toc7857)

[springApplication执行流程 3](#_Toc32464)

[自动配置 4](#_Toc529)

[基于条件的自动配置 4](#_Toc14113)

[调整自动配置的顺序 4](#_Toc7781)

[第四章 了解spring-boot-starter 4](#_Toc19566)

[应用日志与spring-boot-starter-logging 4](#_Toc29001)

[快速Web应用开发与spring-boot-starter-web 4](#_Toc9866)

[项目结构层面的约定 5](#_Toc18749)

[SpringMVC框架层面的约定和定制 5](#_Toc20462)

[嵌入式Web容器层面的约定和定制 5](#_Toc3313)

[数据访问与spring-boot-starter-jdbc 5](#_Toc19030)

[SpringBoot应用的数据库版本化管理 5](#_Toc15823)

[spring-boot-starter-aop及其使用场景说明 5](#_Toc1437)

[spring-boot-starter-aop构建spring-boot-starter-metrics？？？ 6](#_Toc6135)

[应用安全与spring-boot-starter-security 6](#_Toc29260)

[SpringSecurity基本设计 6](#_Toc783)

[定制spring-boot-starter-security 6](#_Toc13742)

[应用监控与spring-boot-starter-actuator 6](#_Toc25231)

[自定义应用的健康状态检查 7](#_Toc29539)

[访问endpoint 7](#_Toc2419)

[章节小结 7](#_Toc209)

[SpringBoot微服务实践 8](#_Toc29816)

[使用SpringBoot构建微服务 8](#_Toc14570)

[基于Dubbo的微服务 8](#_Toc18260)

[使用springboot快速构建Web API 8](#_Toc6920)

[使用springboot构建其他形式的微服务 9](#_Toc11393)

[SpringBoot微服务的发布与部署 9](#_Toc30214)

[spring-boot-starter的发布和部署方式 10](#_Toc29848)

[基于RPM的发布与部署 10](#_Toc10422)

[基于Docker的发布与部署 11](#_Toc31053)

[SpringBoot微服务的注册与发现 11](#_Toc2913)

[SpringBoot微服务的监控与运维 11](#_Toc10638)

[SpringBoot微服务的安全与防护 11](#_Toc29177)

# 了解微服务

SOA，面向服务的架构。

微服务的好处：独立性、隔离性；多语言生态。

# Spring框架的本质

IoC，inversion of controll；

两种实现方式：DI(Dependency Injection)，DL(Dependency Lookup)。

Spring IoC，spring框架实现的IoC Container。

依赖注入分两阶段：注册与收集、分析和组装。new+setter

### javaconfig

基于java代码和Annotation元信息的依赖关系绑定描述方式。两种方式：

Xml方式，beans--bean--property注入值。

java代码方式，@Configuration--@Bean--调用对应@Bean方法注入值。

spring通过拦截配置类，实现单例依赖注入。

#### 常用Annotation

@Configuration，声明配置类。+@Bean

@ComponentScan，组件扫描；basePackage属性，指定扫描路径。

@PropertySource，加载\*.properties文件；java8可同时使用多个@PropertySource

@PropertySources，java8前，同时加载多个@PropertySource

@PropertySources({

@PropertySource(“classpath:1.properties”),

@PropertySource(“classpath:2.properties”)})

@Import，导入JavaConfig形式的IoC容器配置。@Import(User.class)

@ImportResource，导入xml形式的配置。@ImportResource(“...”)

# Springboot的工作机制

习惯优先配置，convention over configuration。

## @SpringBootApplication

主要包含：

@Configuration，声明配置类

@EnableAutoConfiguration，自动装配。

@ComponentScan，开启组件自动扫描。

### @EnableAutoConfiguration分析

通过@Import(EnableAutoConfigurationImportSelector.class)，导入EnableAutoConfigurationImportSelector类，将当前SpringBoot应用中所有@Configuration配置类加载到IoC容器中。

通过Spring框架工具类SpringFactoriesLoader(Spring私有扩展方案)，查找classpath下所有META-INF/springfactories(key-value文件)，自动加载org.springframework.boot.autoconfigure.EnableAutoConfiguration后的@Configuration配置类。

个人理解：@Import加载自定义@Configuration配置类；SpringFactoriesLoader加载依赖包的@Configuration配置类。

## SpringBoot程序启动

可通过SpringApplication实例的setter方法，扩展SpringApplication。

### springApplication执行流程

1. 初始化SpringApplication实例。确定ApplicationContext类型，SpringFactoriesLoader查找classpath()并实例化所有ApplicationContextInitializer、ApplicationListener，设置main方法定义类。
2. 调用实例的run()方法。配置环境、初始化context、执行CommandLineRunner。

遍历执行查找到的SpringApplicationRunListener的started()。

创建并配置Environment，包括PropertySource、Profile。

遍历执行SpringApplicationRunListener的environmentPrepared()。

创建ApplicationContext实例context，配置environment。

遍历执行查找到的ApplicationContextInitializer的initialize(context)。

遍历执行SpringApplicationRunListener的contextLoaded()。

执行ApplicationContext的refresh()，完成IoC。

查找执行ApplicationContext的CommandLineRunner。

遍历执行SpringApplicationRunListener的finished()；有异常时传入异常继续执行。

#### SpringApplicationRunListener

ApplicationListener实例。

监听SpringApplication实例run()方法执行，并在发布事件。

#### ApplicationListener

为SpringBoot添加ApplicationListener有两种方式：

1. SpringApplication.addListeners()/SpringApplication.setListeners()
2. 修改META-INF/spring.factories中key：org.springframework.context.ApplicationListener

#### ApplicationContextInitializer

处理ApplicationContext实例。

#### CommandLineRunner

SpringBoot应用特定的回调扩展接口。所有的CommandLineRunner都在ApplicationContext初始化后执行。

实现CommandLineRunner接口时，使用@Order(org.springframework.core.annotation.Order)、或实现org.springframework.core.Order接口，调整执行顺序。

## 自动配置

### 基于条件的自动配置

@Conditional，meta annotation

复合annotation：

@ConditionalOnClass

@ConditionalOnBean

@ConditionalOnMissingClass

### 调整自动配置的顺序

@AutoConfigureBefore

@AutoConfigureAfter

@Order(int)

# 了解spring-boot-starter

Springboot配置优先级：

命令行参数

系统环境变量

文件系统中的配置文件

Classpath中的配置文件

代码中的配置项

## 应用日志与spring-boot-starter-logging

导入spring-boot-starter-logging后，SpringBoot应用自动使用logback作为应用日志框架。SpringBoot应用启动时，由Spring-boot-starter-logging.Logging-Application-Listener根据情况初始化并使用。

修改默认配置方式：

1，在classpath中添加logback.xml配置文件。

2，通过配置文件--logging.config指向文件系统中logback.xml所在文件夹。

其他应用日志：spring-boot-starter-log4j，spring-boot-starter-log4j2

## 快速Web应用开发与spring-boot-starter-web

导入spring-boot-starter-web后，可通过mvn spring-boot:run 直接启动应用。

### 项目结构层面的约定

src/main/resources/static，存放静态文件；js、css、静态HTML

src/main/resources/templates，存放模板文件。

### SpringMVC框架层面的约定和定制

spring-boot-starter-web默认自动配置MVC组件。

可在IoC容器中注册行的同类型bean定义来替换默认值。

### 嵌入式Web容器层面的约定和定制

spring-boot-starter-web默认嵌入式tomcat作为web容器，端口8080。

修改web容器：导入spring-boot-starter-jetty，spring-boot-starter-undertow。

修改容器属性：修改配置文件，server.属性名；如server.port,server.address

## 数据访问与spring-boot-starter-jdbc

DataSource自动配置模块？

Springboot会自动配置DataSource、JdbcTemplate、DataSourceTransactionManager等，可通过@Autowited、@Resource注入。

自动配置的DataSource通过@Autowired注入，默认方式为byType，有多个DataSource同时存在时会报错。

配置连接连接多个数据库的多个DataSource时，有两种方式：

1. 修改启动类@SpringBootApplication，禁用DataSource自动配置。

@SpringBootApplication(exclude = { DataSourceAutoConfiguration.class ,

DataSourceTransactionManagerAutoConfiguration.class})

1. 为DataSource bean添加@Primary注解，优先注入。

使用jpa、mongodb时也是类似处理。

### SpringBoot应用的数据库版本化管理

SpringBoot提供针对Flyway，Liquibase的自动配置功能：

org.springframework.boot.autoconfigure.flyway.FlywayAutoConfiguration

org.springframework.boot.autoconfigure.liquibase.LiquibaseAutoConfiguration

## spring-boot-starter-aop及其使用场景说明

AOP，Aspect Oriented Programming；面向切面编程。

spring-boot-starter-aop自动配置行为由两部分组成：

1. AopAutoConfiguration类，提供@Configuration配置类和相应配置项(spring.aop.auto ,

spring.aop.proxy-target-class)。

1. spring-boot-starter-aop自身提供针对spring-aop，aspectjrt，aspectweaver的依赖。

### spring-boot-starter-aop构建spring-boot-starter-metrics？？？

## 应用安全与spring-boot-starter-security

spring-boot-starter-security默认提供基于HTTP Basic认证的安全防护策略，默认用户名user，密码打印于控制台。

可在配置文件修改用户名、密码；security.user.name,security.user.password

### SpringSecurity基本设计

Spring Security的Web安全方案基于Java的Servlet API规范构建。

核心类：

AuthenticationManager，登录

AccessDecisionManager，权限

AbstractSecurityInterceptor，拦截验证

AbstractSecurityInterceptor包括三部分：FilterSecurityInterceptor，MethodSecurityInterceptor，AspectjSecurityInterceptor。

Spring Security通过DelegatingFilterProxy、FilterChainProxy实现filter的注入、安全管理。

DelegatingFilterProxy代理的Filter是定义在spring容器中的FilterChainProxy类的bean；

 FilterChainProxy，从Spring IOC容器中取出SecurityFilterChain bean，实现安全管理。

<http://blog.csdn.net/zhangbojun888/article/details/41344249>

### 定制spring-boot-starter-security

继承WebSecurityConfigurerAdapter

[添加注解：@Configuration、@Order(SecurityProperties.ACCESS\_OVERRIDE\_ORDER)](mailto:使用@Configuration、@Order(SecurityProperties.ACCESS_OVERRIDE_ORDER))

重写方法实现定制。

## 应用监控与spring-boot-starter-actuator

Actuator的Endpoint分为两类：监视类Sensor、控制类actuator。

Sensor类Endpoint：

autoconfig，自动配置报告

beans，容器中所有bean的信息

configprops，处理ConfigurationProperties信息，给出汇总信息

info，给出当前springboot应用的信息

health，给出当前springboot应用的健康信息

env，给出当前springboot应用的环境信息

metrics，给出当前springboot应用的metrics信息

trace，给出当前springboot应用的trace信息

mapping，给出当前springboot应用的RequestMapping信息。

Actuator类Endpoint：

shutdown，关闭当前springboot应用

dump，执行线程dump操作

默认情况下，shutdown为关闭状态，其他打开。

修改状态：

endpoints.enable=false #关闭所有Endpoint

endpoints.info.enable=true

### 自定义应用的健康状态检查

Springboot提供常见服务的监控检查支持：

DataSourceHealthIndicator

DiskSpaceHealthIndicator

RedisHealthIndicator

自定义健康检查，继承AbstractHealthIndicator，实现doHealthCheck()方法。

实现自动配置

### 访问endpoint

Springboot应用为web应用时，默认使用：address:port/endpointName访问。

修改访问路径：

management.context-path；设置路径

management.address；设置监听地址

management.port；设置监听端口

## 章节小结

一个典型spring-boot-starter的两点：

1，提供一个@Configuration配置类，并在org.springframework.boot.autoconfigure包META-INF/spring.factories中注册为org.springframework.boot.autoconfigure.EnableAutoConfiguration标志key的一个值，从而加载到ApplicationContext。

2，为实现@Configuration要提供的功能，需添加相关依赖；通过传递依赖，项目中也可以使用这些依赖包。

# SpringBoot微服务实践

## 使用SpringBoot构建微服务

### 基于Dubbo的微服务

#### 声明接口

建立maven工程currency-rate-api

声明接口CurrencyRateService，方法quote()

mvn install/deploy，发布到本地/远程仓库

#### 实现接口

建立maven工程currency-rate-service，导包currency-rate-api、dubbo、spring-context、spring-jdbc

建类CurrencyRateServiceImpl，实现接口

在dubbo的服务描述文件中对外开放服务

<dubbo:service interface=”groupID.currency-rate-api.CurrencyRateService”

Ref=”serviceImpl” registry=”N/A”/> #registry生产环境中需更改

<bean id=”serviceImpl” class=”groupID.currency-rate-service.CurrencyRateServiceImpl”/>

#### 通过启动类启动服务

Dubbo框架服务不依赖J2EE容器对外提供服务，而是以独立进程的形式对外服务。

创建启动类Bootstrap，在main()方法中通过ClassPathXmlApplicationContext启动服务。

#### 构建关闭服务类

JMX

#### 封装为spring-boot-starter-dubbo包

建立springboot maven工程spring-boot-starter-dubbo，导包spring-boot-starter，dubbo

实现CommandLineRunner接口CLRDubbo，在run()方法中实现服务启动。

将CLRDubbo注册到springboot容器中(@Configuration+@Bean)，+@Order最后执行

更改spring.boot.autoconfigure包META-INF/springfactories文件，添加自动配置。

mvn install/deploy，发布到本地/远程仓库。而后直接引用即可

##com.alibaba.dubbo.rpc.Exporter；暴露给外界的类

### 使用springboot快速构建Web API

spring-boot-starter-jersey，构建RESTful风格的Web API。

spring-boot-starter-hateoas，构建有关联性和智能的Web API。

#### 定义Web API规范

json rpc over http，request、response都为json

rpc over http，仅response为json，request不做要求

一般选rpc over http，客户端各种工具都能支持。

自定义响应格式：

{

code : 1,

error : “exception info”,

data : {...}

}

#### 根据规范构建Web API

spring-boot-starter-web默认提供一系列HttpMessageConverter，用于请求参数和响应结果的类型转换。

更改响应格式方式：

1. 显式的强类型封装方式(explicit type wrapper)。不推荐。

创建自定义格式实现类A，controller返回A类的实例。

1. 隐式的自动转换方式(implicit conversion)。
   1. 继承AbstractHttpMessageConverter<Object>，创建转换类，覆盖默认配置。不推荐。
   2. 创建配置类，继承WebMvcConfigurerAdapter，重写extendMessageConverters()，添加自定义HttpMessageConverter实现类。

#### Web API的短板和补足

Web API没有强类型支持，使用者需与Web API提供者沟通后才能知道详细信息，如参数列表、响应格式、请求方式等。可使用根据代码元信息生成API文档的方式来补足。

创建spring-boot-starter-webapi自动配置模块：

1. 创建maven项目spring-boot-starter-webapi；导包spring-boot-starter-web、

springfox-swagger2、springfox-swagger-ui、javax-servlet-api

1. 创建配置类。
2. 修改spring.boot.autoconfigure包META-INF/springfactories文件。
3. mvn install/deploy，发布到本地/远程仓库

### 使用springboot构建其他形式的微服务

构建只响应外部事件的微服务。

## SpringBoot微服务的发布与部署

简单流程：

mvn clean package，打包为jar包。

Scp/rsync等方式部署到服务器

Java -jar myProject.jar，启动应用

发布：打包项目，放到指定地点。

部署：将打包好的项目分发到目标环境的指定资源池，并启动服务。

### spring-boot-starter的发布和部署方式

发布：

1. 获取Nexus服务器信息；或自行构建。
2. 在pom.xml中添加distributionManagement节点。

<distributionManagement>

<repository>

<id>deployment</id> #登录Nexus的username

<name>release</name> #稳定版

<url>url1</url>

<repository>

<snapshotRepository>

<id>deployment</id> #登录Nexus的username

<name>snapshot</name> #不稳定版

<url>url2</url>

<snapshotRepository>

1. 在本地仓库setting.xml中添加Nexus服务器认证信息。

<server>

<id>deployment</id>

<username>deployment</username>

<password>password</password>

</server>

1. mvn deploy，发布

spring-boot-starter为java类库，随项目部署。

### 基于RPM的发布与部署

RPM，Linux centos系统的包管理器。

安装rpmbuild，用于构建rpm包。

构建过程：

1. mvn package，打包项目。
2. 根据规范，构建标准发布格式的rpm包。构建目录体系。
3. 构建标准发布格式到具体目标环境的映射。指定个性化参数。
4. 通过rpm脚本构建rpm包；rpmbuild -bb {projectId}.spec
5. scp/sftp方式发布。

### 基于Docker的发布与部署

## SpringBoot微服务的注册与发现

Dubbo框架注册与发现机制：Redis、Zookeeper

Web API注册与发现机制：基于DNS

通过服务代理(如zuul)访问服务实例，而不是直接访问服务实例；降低用户学习成本。

## SpringBoot微服务的监控与运维

集中式的日志采集、存储和分析平台。

ELK技术栈，ElasticSearch Logstash Kibana。

采集日志、内置”传感器”MBean、metrics数据；白盒模式。

actuator模块，从外部访问应用；黑盒模式。

## SpringBoot微服务的安全与防护