Spring Boot

一、Spring Boot 入门

简介、HelloWorld、原理分析

一、简介

Spring Boot来简化Spring应用开发，约定大于配置，
去繁从简，just run就能创建一个独立的，产品级别的应用

背景：

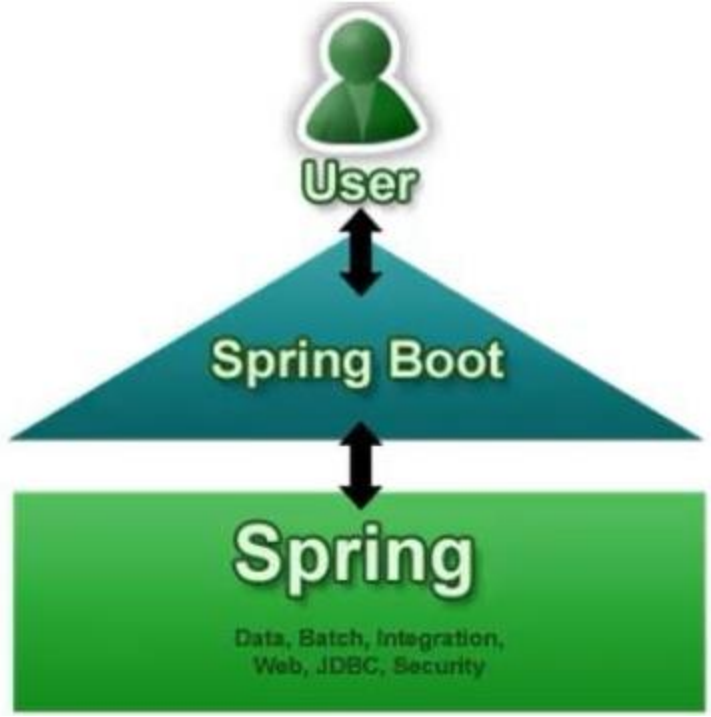
J2EE笨重的开发、繁多的配置、低下的开发效率、复杂的部署流程、第三方技术集成难度大。

解决：

“Spring全家桶”时代。

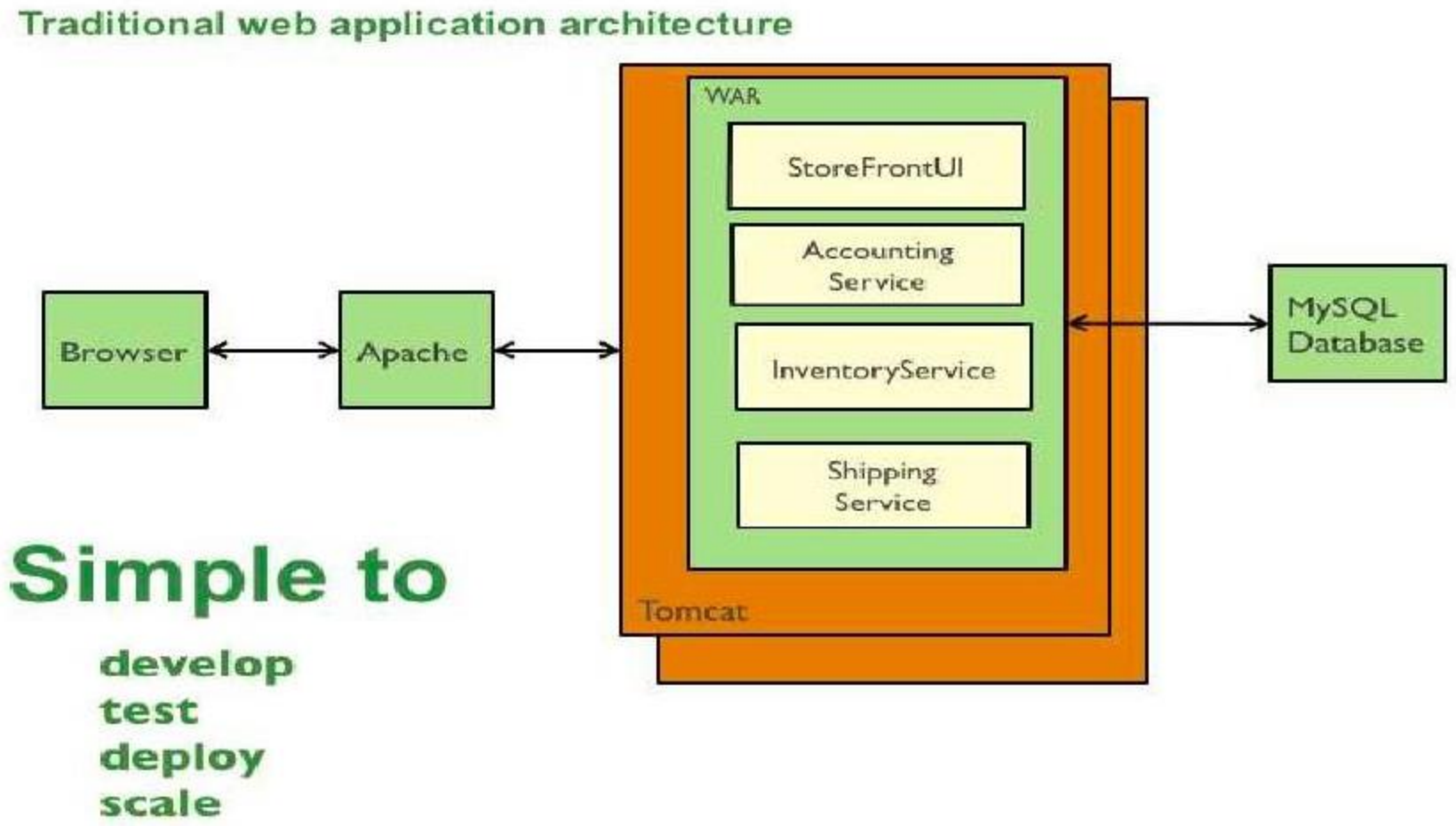
Spring Boot → J2EE一站式解决方案

Spring Cloud → 分布式整体解决方案



- 优点：
  - 快速创建独立运行的Spring项目以及与主流框架集成
  - 使用嵌入式的Servlet容器，应用无需打成WAR包
  - starters自动依赖与版本控制
  - 大量的自动配置，简化开发，也可修改默认值
  - 无需配置XML，无代码生成，开箱即用
  - 准生产环境的运行时应用监控
  - 与云计算的天然集成

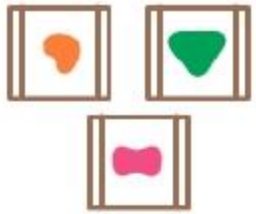
单体应用


Traditional web application architecture
WAR
StoreFrontUI
Accounting Service
InventoryService
Shipping Service
Browser
Apache
MySQL Database
Tomcat
Simple to
develop
test
deploy
scale

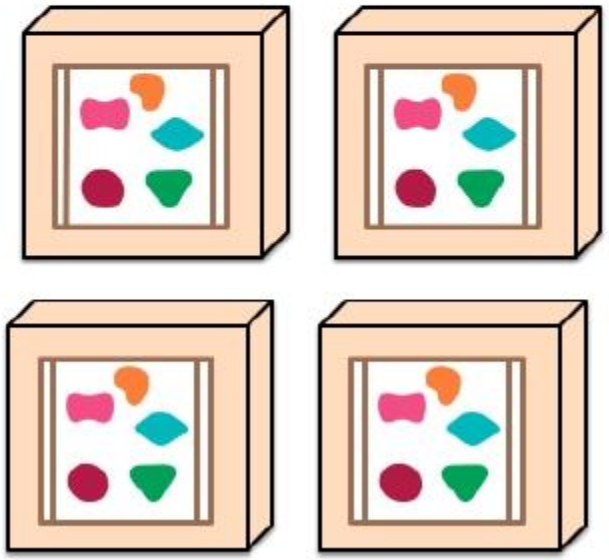
微服务

一个单体应用程序把它所有的功能放在一个单一进程中...

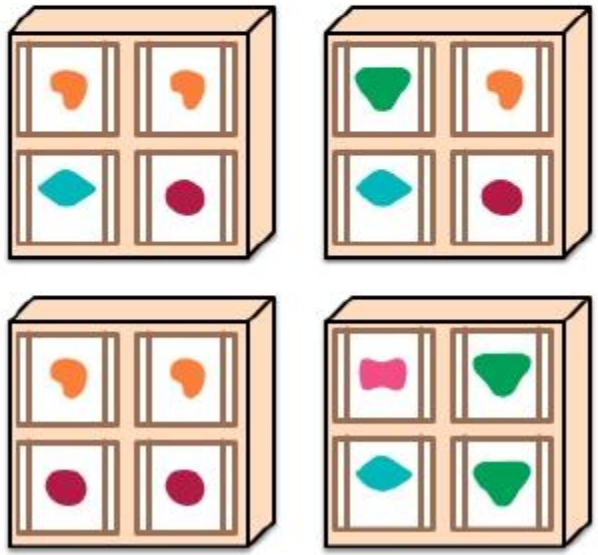
一个微服务架构把每个功能元素放进一个独立的服务中...



...并且通过在多个服务器上复制这个单体进行扩展



...并且通过跨服务器分发这些服务进行扩展，只在需要时才复制.



How Microservices

Josh Evans
@Ops_Engineering

Director of Operations Engineering at Netflix


Your App
Spring Boot
Build Anything
Spring Cloud
Coordinate Anything
Spring Cloud Data Flow
Connect Everything

- 你必须掌握以下内容：
  - Spring框架的使用经验
  - 熟练使用Maven进行项目构建和依赖管理
  - 熟练使用Eclipse或者IDEA

- 环境约束
  - jdk1.8
  - maven3.x
  - IntelliJ IDEA 2018
  - Spring Boot 2.0.3.RELEASE

二、HelloWorld

- 1、创建maven项目
- 2、引入starters
- 3、创建主程序
- 4、启动运行

❷

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>1.5.7.RELEASE</version>
</parent>
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
</dependencies>
```

❸

```
package hello;

import org.springframework.boot.*;
import org.springframework.boot.autoconfigure.*;
import org.springframework.stereotype.*;
import org.springframework.web.bind.annotation.*;

@Controller
@EnableAutoConfiguration
public class SampleController {

    @RequestMapping("/")
    @ResponseBody
    String home() {
        return "Hello World!";
    }

    public static void main(String[] args) throws Exception {
        SpringApplication.run(SampleController.class, args);
    }
}
```

三、HelloWorld探究

1、starters

- Spring Boot为我们提供了简化企业级开发绝大多数场景的starter pom（启动器），只要引入了相应场景的starter pom，相关技术的绝大部分配置将会消除（自动配置），从而简化我们开发。业务中我们就会使用到Spring Boot为我们自动配置的bean
- 参考 https://docs.spring.io/spring-boot/docs/1.5.9.RELEASE/reference/htmlsingle/#using-boot-starter
- 这些starters几乎涵盖了javaee所有常用场景，Spring Boot对这些场景依赖的jar也做了严格的测试与版本控制。我们不必担心jar版本合适度问题。
- spring-boot-dependencies里面定义了jar包的版本

2、入口类和@SpringBootApplication

1、程序从main方法开始运行

2、使用SpringApplication.run()加载主程序类

3、主程序类需要标注@SpringBootApplication

4、 @EnableAutoConfiguration是核心注解；

5、@Import导入所有的自动配置场景

6、@AutoConfigurationPackage定义默认的包扫描规则

7、程序启动扫描加载主程序类所在的包以及下面所有子包的组件；

```
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
        @Filter(type = FilterType.CUSTOM
        @Filter(type = FilterType.CUSTOM
public @interface SpringBootApplication
```

```
@AutoConfigurationPackage
@Import(EnableAutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
```

```
@Configuration
public @interface SpringBootConfiguration {

}
```

```
@Component
public @interface Configuration {
```

3、自动配置

- 1、xxxAutoConfiguration
  - Spring Boot中存现大量的这些类，这些类的作用就是帮我们进行自动配置
  - 他会将这个这个场景需要的所有组件都注册到容器中，并配置好
  - 他们在类路径下的 META-INF/spring.factories文件中
  - spring-boot-autoconfigure-1.5.9.RELEASE.jar中包含了所有场景的自动配置类代码
  - 这些自动配置类是Spring Boot进行自动配置的精髓

二、Spring Boot配置

配置文件、加载顺序、配置原理

一、配置文件

- Spring Boot使用一个全局的配置文件
  - application.properties
  - application.yml

- 配置文件放在**src/main/resources**目录或者**类路径/config**下

- .yml是YAML（YAML Ain't Markup Language）语言的文件，以数据为中心，比json、xml等更适合做配置文件
  - http://www.yaml.org/ 参考语法规范

- 全局配置文件的可以对一些默认配置值进行修改

二、YAML语法

1、YAML基本语法

- 使用缩进表示层级关系
- 缩进时不允许使用Tab键，只允许使用空格。
- 缩进的空格数目不重要，只要相同层级的元素左侧对齐即可
- 大小写敏感

2、YAML 支持的三种数据结构

- 对象：键值对的集合
- 数组：一组按次序排列的值
- 字面量：单个的、不可再分的值

https://nodeca.github.io/js-yaml/ yaml语法校验

- YAML常用写法
  - 对象（Map）
    - 对象的一组键值对，使用冒号分隔。如：username: admin
    - **冒号后面跟空格来分开键值；**
    - {k: v}是行内写法

```yaml
# YAML
hero:
  hp: 34
  sp: 8
  level: 4
orc:
  hp: 12
  sp: 0
  level: 2
```

```
# Java
{'hero': {'hp': 34, 'sp': 8, 'level': 4}, 'orc': {'hp': 12, 'sp': 0, 'level': 2}}
```

- 数组
  - 一组连词线（-）开头的行，构成一个数组，[]为行内写法
  - 数组，对象可以组合使用

```yaml
# YAML
- name: PyYAML
  status: 4
  license: MIT
  language: Python
- name: PySyck
  status: 5
  license: BSD
  language: Python
```

```
# Java
[{'status': 4, 'language': 'Python', 'name': 'PyYAML', 'license': 'MIT'},
{'status': 5, 'license': 'BSD', 'name': 'PySyck', 'language': 'Python'}]
```

```yaml
- Cat
- Dog
- Goldfish
```

```yaml
-
  - Cat
  - Dog
  - Goldfish
```

```yaml
animal: [Cat, Dog]
```

```
var obj = [ 'Cat', 'Dog', 'Goldfish' ];
var obj = [[ 'Cat', 'Dog', 'Goldfish' ]];
var obj = { animal: [ 'Cat', 'Dog' ] };
```

– 复合结构。以上写法的任意组合都是可以

– 字面量
  - 数字、字符串、布尔、日期
  - 字符串
    – 默认不使用引号
    – 可以使用单引号或者双引号，单引号会转义特殊字符
    – 字符串可以写成多行，从第二行开始，必须有一个单空格缩进。换行符会被转为空格。

– 文档
  - 多个文档用 - - - 隔开

注意：

Spring Boot使用 snakeyaml 解析yml文件；

https://bitbucket.org/asomov/snakeyaml/wiki/Documentation#markdown-header-yaml-syntax 参考语法

```yaml
person:
  name: 'zhangsan \n'
  username: 张三
  age: 18
  pet:
    name:  小狗
    gender: male
  animal:
    - dog
    - cat
    - fish
  interests: [足球,篮球]
  friends:
    -
      - zhangsan is my
       best friend
      - "lisi\n"
  childs:
    - name: xiaozhang
      age: 18
    - name: xiaoli
      pets:
        -  a
        -  b
    - {name: lisi, age: 18}
```

```java
@Component
@ConfigurationProperties(prefix = "person")
public class Person {
    private String name;
    private String username;
    private Integer age;
    private Map<String,Object> pet;
    private List<String> animal;
    private List<String> interests;
    private List<Object> friends;
    private List<Map<String,Object>> childs;
```

Pets{name='zhangsan \n', username='张三', age=18, pet={name=小狗, gender=male}, animal=[dog, cat, fish], interests=[足球, 篮球], friends=[[zhangsan is my best friend, lisi
]], childs=[{age=18, name=xiaozhang}, {pets={1=b, 0=a}, name=xiaoli}, {age=18, name=lisi}]}

三、配置文件值注入

- **@Value和@ConfigurationProperties为属性注值对比**

Feature	`@ConfigurationProperties`	`@Value`
Relaxed binding	Yes	No
Meta-data support	Yes	No
`SpEL` evaluation	No	Yes

- **属性名匹配规则（Relaxed binding）**
  - person.firstName：使用标准方式
  - person.first-name：大写用-
  - person.first_name：大写用_
  - PERSON_FIRST_NAME：
    - 推荐系统属性使用这种写法

- **@ConfigurationProperties**
  - 与@Bean结合为属性赋值
  - 与@PropertySource（只能用于properties文件）结合读取指定文件

- **@ConfigurationProperties Validation**
  - 支持JSR303进行配置文件值校验；

```
@ConfigurationProperties(prefix="connection")
@Validated
public class FooProperties {

    @NotNull
    private InetAddress remoteAddress;

    @Valid
    private final Security security = new Security();
```

- **@ImportResource读取外部配置文件**

四、配置文件占位符

- **RandomValuePropertySource：配置文件中可以使用随机数**
  ${random.value}、${random.int}、${random.long}
  ${random.int(10)}、${random.int[1024,65536]}

- **属性配置占位符**

```
app.name=MyApp
app.description=${app.name} is a Spring Boot application
```

  - 可以在配置文件中引用前面配置过的属性（优先级前面配置过的这里都能用）。
  - ${app.name:默认值}来指定找不到属性时的默认值

五、 Profile

Profile是Spring对不同环境提供不同配置功能的支持，可以通过激活、指定参数等方式快速切换环境

1、多profile文件形式：

- 格式：application-{profile}.properties/yml：
  - application-dev.properties、application-prod.properties

2、多profile文档块模式：

3、激活方式：

- 命令行 --spring.profiles.active=dev
- 配置文件 spring.profiles.active=dev
- jvm参数 –Dspring.profiles.active=dev

```yaml
spring:
  profiles:
    active: prod # profiles.active: 激活指定配置
---
spring:
  profiles: prod
server:
  port: 80
--- #三个短横线分割多个profile区（文档块）
spring:
  profiles: default # profiles: default表示未指定默认配置
server:
  port: 8080
```

六、配置文件加载位置

- spring boot 启动会扫描以下位置的application.properties或者application.yml文件作为Spring boot的默认配置文件
  - file:./config/
  - file:./
  - classpath:/config/
  - classpath:/
  - 以上是按照**优先级从高到低**的顺序，所有位置的文件都会被加载，**高优先级配置**内容会**覆盖低优先级配置**内容。
  - 我们也可以通过配置spring.config.location来改变默认配置

七、外部配置加载顺序

Spring Boot 支持多种外部配置方式

这些方式优先级如下：

https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/htmlsingle/#boot-features-external-config

1. 命令行参数
2. 来自java:comp/env的JNDI属性
3. Java系统属性（System.getProperties()）
4. 操作系统环境变量
5. RandomValuePropertySource配置的random.*属性值
6. jar包外部的application-{profile}.properties或application.yml(带spring.profile)配置文件
7. jar包内部的application-{profile}.properties或application.yml(带spring.profile)配置文件
8. jar包外部的application.properties或application.yml(不带spring.profile)配置文件
9. jar包内部的application.properties或application.yml(不带spring.profile)配置文件
10. @Configuration注解类上的@PropertySource
11. 通过SpringApplication.setDefaultProperties指定的默认属性

八、自动配置原理

1、可以查看HttpEncodingAutoConfiguration

2、通用模式

- xxxAutoConfiguration：自动配置类
- xxxProperties：属性配置类
- yml/properties文件中能配置的值就来源于[属性配置类]

3、几个重要注解

- @Bean
- @Conditional

4、--debug=true查看详细的自动配置报告

@Conditional扩展

@Conditional扩展注解	作用（判断是否满足当前指定条件）
@ConditionalOnJava	系统的java版本是否符合要求
@ConditionalOnBean	容器中存在指定Bean；
@ConditionalOnMissingBean	容器中不存在指定Bean；
@ConditionalOnExpression	满足SpEL表达式指定
@ConditionalOnClass	系统中有指定的类
@ConditionalOnMissingClass	系统中没有指定的类
@ConditionalOnSingleCandidate	容器中只有一个指定的Bean，或者这个Bean是首选Bean
@ConditionalOnProperty	系统中指定的属性是否有指定的值
@ConditionalOnResource	类路径下是否存在指定资源文件
@ConditionalOnWebApplication	当前是web环境
@ConditionalOnNotWebApplication	当前不是web环境
@ConditionalOnJndi	JNDI存在指定项

三、Spring Boot与日志

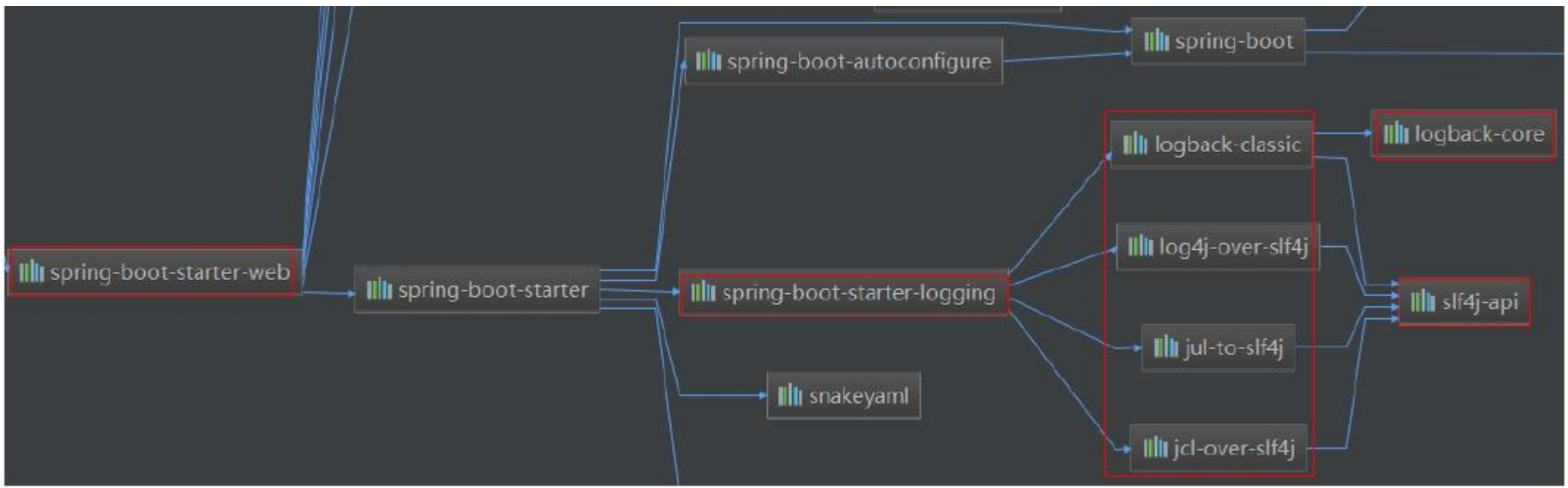
日志框架、日志配置

一、日志框架

市场上存在非常多的日志框架。JUL（java.util.logging），JCL（Apache Commons Logging），Log4j，Log4j2，Logback、SLF4j、jboss-logging等。Spring Boot在框架内容部使用JCL，spring-boot-starter-logging采用了slf4j+logback的形式，Spring Boot也能自动适配（jul、log4j2、logback） 并简化配置

日志门面	日志实现
JCL（JakartaCommonsLogging） SLF4j（SimpleLoggingFacadeforJava） jboss-logging	Log4j JUL（java.util.logging） Log4j2 Logback

Logging System	Customization
Logback	`logback-spring.xml`, `logback-spring.groovy`, `logback.xml` or `logback.groovy`
Log4j2	`log4j2-spring.xml` or `log4j2.xml`
JDK (Java Util Logging)	`logging.properties`



二、默认配置

1、全局常规设置（格式、路径、级别）

2、指定日志配置文件位置

3、切换日志框架

二选一

`spring-boot-starter-log4j2`	Starter for using Log4j2 for logging. An alternative to `spring-boot-starter-logging`
`spring-boot-starter-logging`	Starter for logging using Logback. Default logging starter

logging.file	logging.path	Example	Description
(none)	*(none)*		只在控制台输出
指定文件名	*(none)*	my.log	输出日志到my.log文件
(none)	指定目录	/var/log	输出到指定目录的 spring.log 文件中

四、Spring Boot与Web开发

Thymeleaf、web定制、容器定制

一、web自动配置规则

1、WebMvcAutoConfiguration
2、WebMvcProperties
3、ViewResolver自动配置
4、静态资源自动映射
5、Formatter与Converter自动配置
6、HttpMessageConverter自动配置
7、静态首页
8、favicon
9、错误处理

二、Thymeleaf模板引擎

Thymeleaf是一款用于渲染XML/XHTML/HTML5内容的模板引擎。类似JSP，Velocity，FreeMaker等，它也可以轻易的与Spring MVC等Web框架进行集成作为Web应用的模板引擎。与其它模板引擎相比，Thymeleaf最大的特点是能够直接在浏览器中打开并正确显示模板页面，而不需要启动整个Web应用

Spring Boot推荐使用Thymeleaf、Freemarker等后现代的模板引擎技术；一但导入相关依赖，会自动配置ThymeleafAutoConfiguration、FreeMarkerAutoConfiguration。

1、整合Thymeleaf

- 1、导入starter-thymeleaf
- 2、template文件夹下创建模板文件
- 3、测试页面&取值
- 4、基本配置

2、基本语法

- 表达式：
  - #{...}：国际化消息
  - ${...}：变量取值
  - *{...}：当前对象/变量取值
  - @{...}：url表达式
  - ~{...}：片段引用
  - 内置对象/共用对象：

- 判断/遍历：
  - th:if
  - th:unless
  - th:each
  - th:switch、th:case

- th:属性

三、定制web扩展配置

1、WebMvcConfigurerAdapter

Spring Boot提供了很多xxxConfigurerAdapter来定制配置

2、定制SpringMVC配置

3、@EnableWebMvc全面接管SpringMVC

4、注册view-controller、interceptor等

5、注册Interceptor

四、配置嵌入式Servlet容器

1、ConfigurableEmbeddedServletContainer

2、EmbeddedServletContainerCustomizer

3、注册Servlet、Filter、Listener

ServletRegistrationBean

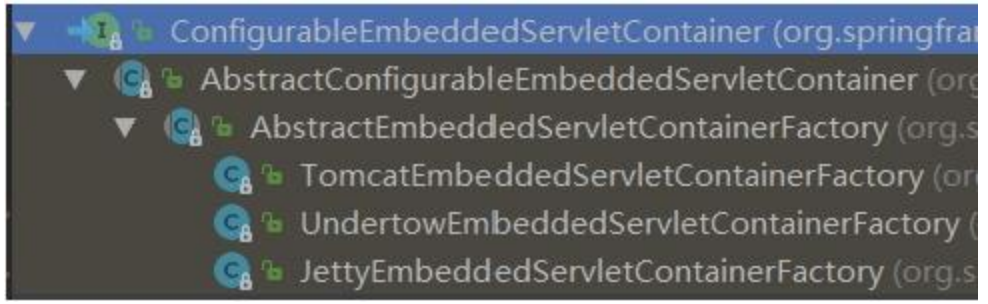
FilterRegistrationBean

ServletListenerRegistrationBean

4、使用其他Servlet容器

Jetty（长连接）

Undertow（不支持JSP）




```
public interface EmbeddedServletContainerCustomizer {

    /**
     * Customize the specified {@Link ConfigurableEmbeddedServletContainer}.
     * @param container the container to customize
     */
    void customize(ConfigurableEmbeddedServletContainer container);

}
```

五、使用外部Servlet容器

1、 SpringBootServletInitializer
   - 重写configure
2、SpringApplicationBuilder
   - builder.source(@SpringBootApplication类)
3、启动原理
   - Servlet3.0标准ServletContainerInitializer扫描所有jar包中META-INF/services/javax.servlet.ServletContainerInitializer文件指定的类并加载
   - 加载spring web包下的SpringServletContainerInitializer
   - 扫描@HandleType(WebApplicationInitializer)
   - 加载SpringBootServletInitializer并运行onStartup方法
   - 加载@SpringBootApplication主类，启动容器等