从 Spring 到 Spring Boot

极客时间

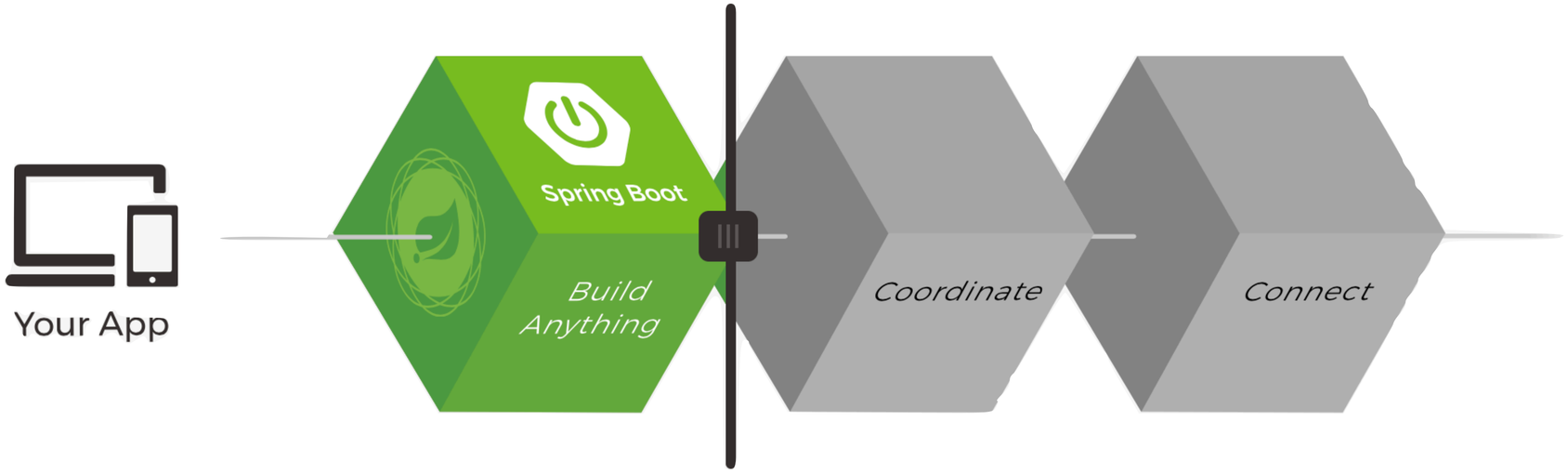
扫码试看/订阅

《玩转 Spring 全家桶》

重新认识 Spring Boot

认识 Spring Boot 的组成部分

Spring: the source for modern java



Spring Boot

BUILD ANYTHING

Spring Boot is designed to get you up and running as quickly as possible, with minimal upfront configuration of Spring. Spring Boot takes an opinionated view of building production-ready applications.

Spring Cloud

COORDINATE ANYTHING

Built directly on Spring Boot's innovative approach to enterprise Java, Spring Cloud simplifies distributed, microservice-style architecture by implementing proven patterns to bring resilience, reliability, and coordination to your microservices.

Spring Cloud Data Flow

CONNECT ANYTHING

Connect the Enterprise to the Internet of Anything—mobile devices, sensors, wearables, automobiles, and more. Spring Cloud Data Flow provides a unified service for creating composable data microservices that address streaming and ETL-based data processing patterns.

Spring Boot 不是什么

- 不是应用服务器
- 不是 Java EE 之类的规范
- 不是代码生成器
- 不是 Spring Framework 的升级版

Spring Boot 的特性

- 方便地创建可独立运行的 Spring 应用程序
- 直接内嵌 Tomcat、Jetty 或 Undertow
- 简化了项目的构建配置
- 为 Spring 及第三方库提供自动配置
- 提供生产级特性
- 无需生成代码或进行 XML 配置

Spring Boot 的四大核心

- 自动配置 - Auto Configuration
- 起步依赖 - Starter Dependency
- 命令行界面 - Spring Boot CLI
- Actuator

极客时间

了解自动配置的实现原理

了解自动配置

自动配置

- 基于添加的 JAR 依赖自动对 Spring Boot 应用程序进行配置
- spring-boot-autoconfiguration

开启自动配置

- `@EnableAutoConfiguration`
  - `exclude = Class<?>[]`
- `@SpringBootApplication`

自动配置的实现原理

@EnableAutoConfiguration

- AutoConfigurationImportSelector
- META-INF/spring.factories
  - org.springframework.boot.autoconfigure.EnableAutoConfiguration

自动配置的实现原理

条件注解

- @Conditional
- @ConditionalOnClass
- @ConditionalOnBean
- @ConditionalOnMissingBean
- @ConditionalOnProperty
- ……

了解自动配置的情况

观察自动配置的判断结果

- --debug

ConditionEvaluationReportLoggingListener

- Positive matches
- Negative matches
- Exclusions
- Unconditional classes

动手实现自己的自动配置

主要工作内容

编写 Java Config

- @Configuration

添加条件

- @Conditional

定位自动配置

- META-INF/spring.factories

条件注解大家庭

条件注解

- @Conditional

类条件

- @ConditionalOnClass
- @ConditionalOnMissingClass

属性条件

- @ConditionalOnProperty

条件注解大家庭

Bean 条件

- @ConditionalOnBean
- @ConditionalOnMissingBean
- @ConditionalOnSingleCandidate

资源条件

- @ConditionalOnResource

条件注解大家庭

Web 应用条件

- @ConditionalOnWebApplication
- @ConditionalOnNotWebApplication

其他条件

- @ConditionalOnExpression
- @ConditionalOnJava
- @ConditionalOnJndi

自动配置的执行顺序

执行顺序

- @AutoConfigureBefore
- @AutoConfigureAfter
- @AutoConfigureOrder

“Talk is cheap, show me the code.”

Chapter 9 / autoconfigure-demo

如何在低版本 Spring 中快速实现类似自动配置的功能

需求与问题

核心的诉求

- 现存系统，不打算重构
- Spring 版本3.x，不打算升级版本和引入 Spring Boot
- 期望能够在少改代码的前提下实现一些功能增强

面临的问题

- 3.x 的 Spring 没有条件注解
- 无法自动定位需要加载的自动配置

核心解决思路

条件判断

- 通过 BeanFactoryPostProcessor 进行判断

配置加载

- 编写 Java Config 类
- 引入配置类
  - 通过 component-scan
  - 通过 XML 文件 import

Spring 的两个扩展点

BeanPostProcessor

- 针对 Bean 实例
- 在 Bean 创建后提供定制逻辑回调

BeanFactoryPostProcessor

- 针对 Bean 定义
- 在容器创建 Bean 前获取配置元数据
- Java Config 中需要定义为 `static` 方法

关于 Bean 的一些定制

Lifecycle Callback

- InitializingBean / @PostConstruct / init-method
- DisposableBean / @PreDestroy / destroy-method

XxxAware 接口

- ApplicationContextAware
- BeanFactoryAware
- BeanNameAware

一些常用操作

判断类是否存在

- ClassUtils.isPresent()

判断 Bean 是否已定义

- ListableBeanFactory.containsBeanDefinition()
- ListableBeanFactory.getBeanNamesForType()

注册 Bean 定义

- BeanDefinitionRegistry.registerBeanDefinition()
  - GenericBeanDefinition
- BeanFactory.registerSingleton()

“Talk is cheap, show me the code.”

Chapter 9 / geektime-autoconfigure-backport

了解起步依赖及其实现原理

很久以前……

- 你能记得多少 Maven 依赖
- 要实现一个功能，需要引入哪些依赖
- 多个依赖项目之间是否会有兼容问题
- ……

关于 Maven 依赖管理的一些小技巧

了解你的依赖

- `mvn dependency:tree`
- IDEA Maven Helper 插件

排除特定依赖

- `exclusion`

统一管理依赖

- `dependencyManagement`
- Bill of Materials - bom

Spring Boot 的起步依赖

Starter Dependencies

- 直接面向功能
- 一站获得所有相关依赖，不再复制粘贴

官方的 Starters

- spring-boot-starter-*

定制自己的起步依赖

你的 Starter

主要内容

- autoconfigure 模块，包含自动配置代码
- starter 模块，包含指向自动配置模块的依赖及其他相关依赖

命名方式

- xxx-spring-boot-autoconfigure
- xxx-spring-boot-starter

一些注意事项

- 不要使用 spring-boot 作为依赖的前缀
- 不要使用 spring-boot 的配置命名空间
- starter 中仅添加必要的依赖
- 声明对 spring-boot-starter 的依赖

“Talk is cheap, show me the code.”

Chapter 9 / geektime-spring-boot-starter

深挖 Spring Boot 的配置加载机制

外化配置加载顺序

- 开启 DevTools 时，~/.spring-boot-devtools.properties
- 测试类上的 @TestPropertySource 注解
- @SpringBootTest#properties 属性
- 命令行参数（ --server.port=9000 ）
- SPRING_APPLICATION_JSON 中的属性

外化配置加载顺序

- ServletConfig 初始化参数
- ServletContext 初始化参数
- java:comp/env 中的 JNDI 属性
- System.getProperties()
- 操作系统环境变量
- random.* 涉及到的 RandomValuePropertySource

外化配置加载顺序

- jar 包外部的 application-{profile}.properties 或 .yml
- jar 包内部的 application-{profile}.properties 或 .yml
- jar 包外部的 application.properties 或 .yml
- jar 包内部的 application.properties 或 .yml

外化配置加载顺序

- @Configuration 类上的 @PropertySource
- SpringApplication.setDefaultProperties() 设置的默认属性

application.properties

默认位置

- `./config`
- `./`
- CLASSPATH 中的 `/config`
- CLASSPATH 中的 `/`

application.properties

修改名字或路径

- spring.config.name
- spring.config.location
- spring.config.additional-location

Relaxed Binding

命名风格	使用范围	示例
短划线分隔	Properties 文件 YAML 文件 系统属性	geektime.spring-boot.first-demo
驼峰式		geektime.springBoot.firstDemo
下划线分隔		geektime.spring_boot.first_demo
全大写，下划线分隔	环境变量	GEEKTIME_SPRINGBOOT_FIRSTDEMO

理解配置背后的 PropertySource 抽象

PropertySource

添加 PropertySource

- <context:property-placeholder>
- PropertySourcesPlaceholderConfigurer
  - PropertyPlaceholderConfigurer
- @PropertySource
- @PropertySources

Spring Boot 中的 @ConfigurationProperties

- 可以将属性绑定到结构化对象上
- 支持 Relaxed Binding
- 支持安全的类型转换
- `@EnableConfigurationProperties`

定制 PropertySource

主要步骤

- 实现 PropertySource<T>
- 从 Environment 取得 PropertySources
- 将自己的 PropertySource 添加到合适的位置

切入位置

- EnvironmentPostProcessor
- BeanFactoryPostProcessor

“Talk is cheap, show me the code.”

Chapter 9 / property-source-demo

极客时间

扫码试看/订阅

《玩转 Spring 全家桶》