理解 Spring 技术栈或许没有那么难

Spring Framework 核心特性

IoC container（IoC 容器）

- Java Bean
  - Setter/Getter method
- Java Beans（规范）
  - 内省
    - 基础核心特性：反射（Reflection）
    - 附加特性：引用（Reference）
      - `Reference`
        - `SoftReference`
        - `WeakReference`
        - `PhantomReference`
        - `FinalReference`
    - `BeanInfo`
      - `BeanDescriptor`
      - `PropertyDescriptor`
        - 事件:`PropertyChangeEvent`
          - `java.util.EventObject`
        - 监听器：`PropertyChangeListener`
          - `java.util.EventListener`
      - `MethodDescriptor`

- IoC（反转控制）
  - 应用不关心依赖组件的来源，通过一定 DI 方式查找
  - 抄袭了 Java EE 概念, Java EE 容器（EJB 容器、JSF 容器）
    - EJB 容器
      - Bean 模式
        - Local
          - 容器 Bean
        - Remote
          - RMI

        - Bean 类型
            - 会话（Session Bean）
                - 无状态 Bean
                - 有状态 Bean
            - 持久化（Persistence Bean）
                - Hibernate
            - 消息驱动 Bean（Java Messaging Bean）
                - JMS
                    - ActiveMQ、Kafka
- DI（依赖注入）
    - Java EE
        - JNDI（Java Naming and Directory Interface）
            - `javax.naming.Context`
                - 名称
                    - `javax.naming.Context#lookup(java.lang.String)`
                    - `javax.naming.Context#lookup(javax.naming.Name)`
        - EJB
            - `javax.ejb.@Ejb`
            -
    - 依赖查找（Dependency Lookup）
        - ID、别名、名称查找
            - `BeanFactory#getBean(String)`：Object
        - 类型查找
            - `BeanFactory#getBean(Class)`：T
        - 注解查找
            - `ListableBeanFactory#getBeansWithAnnotation(Class)`
        - `FactoryBean` 查找
            - `FactoryBean#getObject()`
        - `ObjectFactory` 查找
            - `ObjectFactory#getObject()`
    - 依赖注入（Dependency Injection）
        - 方法
            - Spring `@Autowired`
            - Java `@Resource`
            - Java EE `@Inject`
        - 途径
            - 字段（Field）注入
            - 属性（Setter）注入
            - 方法（Method）注入
            - 构造器（Constructor）注入

Events

Essentially, this is the standard *Observer* design pattern.

核心模式：观察模式

Spring 事件：`ApplicationEvent`

- `java.util.EventObject`

Spring 事件监听器：`ApplicationListener`

- `java.util.EventListener`
-

Spring 事件广播器：`ApplicationEventMulticaster`

- `SimpleApplicationEventMulticaster`

```
    @Override
    public void multicastEvent(final ApplicationEvent event, @Nullable ResolvableType eventType)
{
        ResolvableType type = (eventType != null ? eventType : resolveDefaultEventType(event));
        for (final ApplicationListener<?> listener : getApplicationListeners(event, type)) {
            Executor executor = getTaskExecutor();
            if (executor != null) {
                executor.execute(() -> invokeListener(listener, event));
            }
            else {
                invokeListener(listener, event);
            }
        }
    }
```

Bean 生命周期管理之一

- 事件（包装）数据变化

Resources

Java 资源管理

URL 资源管理

- 协议

- 获取协议：`java.net.URL#getProtocol`
- 协议处理
  - HTTP：`sun.net.www.protocol.http.Handler`
  - FTP：`sun.net.www.protocol.ftp.Handler`
  - HTTPS：`sun.net.www.protocol.https.Handler`
  - File：`sun.net.www.protocol.file.Handler`
  - Email：`sun.net.www.protocol.emailto.Handler`

ClassLoader（Class Path）资源管理

- 资源
  - 获取当前 ClassLoader 资源 URL：`java.lang.ClassLoader#getResource(String)`
  - 获取当前 ClassLoader 所有资源 URL：`java.lang.ClassLoader#getResources(String)`
  - 获取当前 ClassLaoder 资源 `InputStream`：`java.lang.ClassLoader#getResourceAsStream(String)`

Spring 资源管理

资源定位：`classpath:/META-INF/abc.properties`

多资源定位：`classpath*:/META-INF/abc.properties`

`Resource` 接口

`URL`、`File` 和 `ClassLoader` 封装实现

语义：

- 资源定位（`URL`、`File`）
- 资源流读取（`InputStream`）

- 实现类
  - `ClassPathResource`
    - `getURL()` -> `ClassLoader#getResource(String)`
    - `getInputStream()` -> `ClassLoader#getResourceAsStream(String)`
- 加载器
  - `ResourceLoader`
    - 获取 `Resource`：`ResourceLoader#getResource(String)`
    - 默认实现：`DefaultResourceLoader`
      - 前缀 = "classpath:" -> `ClassPathResource`

        - 否则 -> `FileUrlResource` 或 `UrlResource`
- 协议扩展
    - `ProtocolResolver`
        - 通过路径解析出 `Resource`

i18n

Java i18n

Java 标准接口 `ResourceBundle`

- < Java 1.6 : 乱码，解决方案 native2ascci
- Java 1.6 `ResourceBundle.Control`
- Java 1.8 `ResourceBundleControlProvider`
    - Java 1.6 `ServiceLoader` SPI

Spring `MessageSource`

- `MesasgeFormat`
    - Hello,{0} -> 0="World" -> Hello,World
- 实现类
    - `ResourceBundleMessageSource`

Validation

Java Bean Validation （ JSR-308 ）

Spring Validator

Spring Validator + Bean Validation : `LocalValidatorFactoryBean`

使用场景

- 标准 Java Bean 校验
- Spring MVC 校验

    - Form 校验、`@RequestBody` 校验，可以自定义
- Spring Boot 外部化配置
    - `@ConfigurationProperties`

Data Binding

使用场景

- Spring 自定义绑定
- Spring MVC 参数绑定
- Spring Boot 外部化配置
    - `@ConfigurationProperties`

Type Conversion

常见类型装换

自定义装换

`ConversionService`

使用场景

- Spring Boot 外部化配置
    - `@ConfigurationProperties`
-

SpEL

AOP

Spring Boot 实际的场景

不适合场景

多 DataSource

多 事务

Reactive -> RxJava 或者 Reactor

Java 8 CompletableFuture、Java 9 Flow API

Vert.x ->

Spring Cloud 应该怎么学