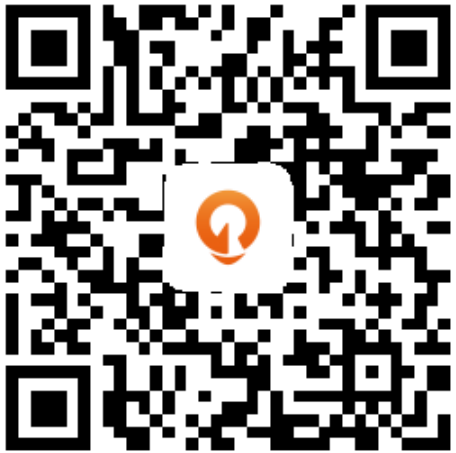
第二章：重新认识 IoC

小马哥 · mercyblitz

极客时间



扫码试看/订阅

《小马哥讲 Spring 核心编程思想》视频课程

重新认识 IoC

1. IoC 发展简介
2. IoC 主要实现策略
3. IoC 容器的职责
4. IoC 容器的实现
5. 传统 IoC 容器实现
6. 轻量级 IoC 容器
7. 依赖查找 VS. 依赖注入
8. 构造器注入 VS. Setter 注入
9. 面试题精选

IoC 发展简介

- 什么是 IoC ？

In software engineering, inversion of control (IoC) is a programming principle. IoC inverts the flow of control as compared to traditional control flow. In IoC, custom-written portions of a computer program receive the flow of control from a generic framework. A software architecture with this design inverts control as compared to traditional procedural programming: in traditional programming, the custom code that expresses the purpose of the program calls into reusable libraries to take care of generic tasks, but with inversion of control, it is the framework that calls into the custom, or task-specific, code.

来源：https://en.wikipedia.org/wiki/Inversion_of_control

IoC 发展简介

- IoC 的简史
  - 1983年，Richard E. Sweet 在《The Mesa Programming Environment》中提出“Hollywood Principle”（好莱坞原则）
  - 1988年，Ralph E. Johnson & Brian Foote 在《Designing Reusable Classes》中提出“Inversion of control”（控制反转）
  - 1996年，Michael Mattsson 在《Object-Oriented Frameworks, A survey of methodological issues》中将“Inversion of control”命名为 “Hollywood principle”
  - 2004年，Martin Fowler 在《Inversion of Control Containers and the Dependency Injection pattern》中提出了自己对 IoC 以及 DI 的理解
  - 2005年，Martin Fowler 在《InversionOfControl》对 IoC 做出进一步的说明

IoC 主要实现策略

- 维基百科（https://en.wikipedia.org/wiki/Inversion_of_control）

Implementation techniques 小节的定义：

"*In object-oriented programming, there are several basic techniques to implement inversion of control. These are:*

- *Using a service locator pattern*
- *Using dependency injection, for example* 使用依赖注入
    - *Constructor injection*
    - *Parameter injection*
    - *Setter injection*
    - *Interface injection*
- *Using a contextualized lookup* 使用上下文查找
- *Using template method design pattern* 使用模板方法(如 jdbc)
- *Using strategy design pattern*"

IoC 主要实现策略

- 《Expert One-on-One™ J2EE™ Development without EJB™》提到的主要实现策略：

“IoC is a broad concept that can be implemented in different ways. There are two main types:

依赖查找

- *Dependency Lookup: The container provides callbacks to components, and a lookup context. This is the EJB and Apache Avalon approach. It leaves the onus on each component to use container APIs to look up resources and collaborators. The Inversion of Control is limited to the container invoking callback methods that application code can use to obtain resources.*

依赖注入

- *Dependency Injection: Components do no look up; they provide plain Java methods enabling the container to resolve dependencies. The container is wholly responsible for wiring up components, passing resolved objects in to JavaBean properties or constructors. Use of JavaBean properties is called Setter Injection; use of constructor arguments is called Constructor Injection.”*

settter、参数、构造器注入

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

IoC 容器的职责

- 维基百科（https://en.wikipedia.org/wiki/Inversion_of_control）

在 Overview 小节中提到：

“Inversion of control serves the following design purposes:

- *To decouple the execution of a task from implementation.* 解耦任务实现的执行
- *To focus a module on the task it is designed for.* 关注于模块设计的任务
- *To free modules from assumptions about how other systems do what they do and instead rely on contracts.* 假定
- *To prevent side effects when replacing a module.* 消除替换模块带来的副作用

Inversion of control is sometimes facetiously referred to as the "Hollywood Principle: Don't call us, we'll call you".”

IoC 容器的职责

- 通用职责
- 依赖处理
  - 依赖查找 更偏主动方式
  - 依赖注入
- 生命周期管理
  - 容器
  - 托管的资源（Java Beans 或其他资源）
    如事件的监听器
- 配置
  - 容器 如定时启动
  - 外部化配置 如属性配置
  - 托管的资源（Java Beans 或其他资源）
    如Tomcat、线程池

IoC 容器的实现

- 主要实现
- Java SE
  - Java Beans
  - Java ServiceLoader SPI　SPI
  - JNDI（Java Naming and Directory Interface）　java 命名(类比DNS)和目录(类比电话簿)接口
- Java EE
  - EJB（Enterprise Java Beans）
  - Servlet　如Model2设计模式
- 开源
  - Apache Avalon（http://avalon.apache.org/closed.html）
  - PicoContainer（http://picocontainer.com/）
  - Google Guice（https://github.com/google/guice）
  - Spring Framework（https://spring.io/projects/spring-framework）

传统 IoC 容器的实现

- Java Beans 作为 IoC 容器
- 特性
  - 依赖查找
  - 生命周期管理
  - 配置元信息
  - 事件　spring 事件是基于 java 事件做的
  - 自定义
  - 资源管理
  - 持久化
- 规范
  - JavaBeans：https://www.oracle.com/technetwork/java/javase/tech/index-jsp-138795.html
  - BeanContext：https://docs.oracle.com/javase/8/docs/technotes/guides/beans/spec/beancontext.html

轻量级 IoC 容器

- 《Expert One-on-One™ J2EE™ Development without EJB™》认为轻量级容器的特征：“
- A container that can manage application code. 可以管理应用代码(如代码的启停)
- A container that is quick to start up.
- A container that doesn't require any special deployment steps to deploy objects within it.
  不需要特别的部署步骤(运维)
- A container that has such a light footprint and minimal API dependencies that it can be run in a variety of environments.
  轻量级内存占用
- A container that sets the bar for adding a managed object so low in terms of deployment effort and performance overhead that it's possible to deploy and manage fine-grained objects, as well as coarse-grained components.”

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

轻量级 IoC 容器

- 《Expert One-on-One™ J2EE™ Development without EJB™》认为轻量级容器的好处：

- *Escaping the monolithic container*
  释放聚式/单体容器(微服务)
- *Maximizing code reusability*

- *Greater object orientation*
  更大化的面向对象
- *Greater productivity*

- *Better testability*

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

依赖查找 VS. 依赖注入

- Java Beans 作为 IoC 容器
- 特性
  - 依赖查找
  - 生命周期管理
  - 配置元信息
  - 事件
  - 自定义
  - 持久化
- 规范
  - JavaBeans：https://www.oracle.com/technetwork/java/javase/tech/index-jsp-138795.html
  - BeanContext：https://docs.oracle.com/javase/8/docs/technotes/guides/beans/spec/beancontext.html

构造器注入 VS. Setter 注入

- Spring Framework 对构造器注入与 Setter 的论点：

"*The Spring team generally advocates constructor injection, as it lets you implement application components as immutable objects and ensures that required dependencies are not null. Furthermore, constructor-injected components are always returned to the client (calling) code in a fully initialized state. As a side note, a large number of constructor arguments is a bad code smell, implying that the class likely has too many responsibilities and should be refactored to better address proper separation of concerns.*

Setter injection should primarily only be used for optional dependencies that can be assigned reasonable default values within the class. Otherwise, not-null checks must be performed everywhere the code uses the dependency. One benefit of setter injection is that setter methods make objects of that class amenable to reconfiguration or re-injection later. Management through JMX MBeans is therefore a compelling use case for setter injection."

鼓励构造器注入

不可变对象不为空

settet 注入应该主要仅用于可选依赖

使得对象更可重新配置和之后重新注入

来自“Constructor-based or setter-based DI” 节选

构造器注入 VS. Setter 注入

- 《Expert One-on-One™ J2EE™ Development without EJB™》认为 Setter 注入的优点：

"*Advantages of Setter Injection include:*

- *JavaBean properties are well supported in IDEs.* ide支持良好
- *JavaBean properties are self-documenting.*
- *JavaBean properties are inherited by subclasses without the need for any code.* 得益于继承
- *It's possible to use the standard JavaBeans property-editor machinery for type conversions if necessary.*
- *Many existing JavaBeans can be used within a JavaBean-oriented IoC container without modification.*
- *If there is a corresponding getter for each setter (making the property readable, as well as writable), it is possible to ask the component for its current configuration state. This is particularly useful if we want to persist that state: for example, in an XML form or in a database. With Constructor Injection, there's no way to find the current state.*
- *Setter Injection works well for objects that have default values, meaning that not all properties need to be supplied at runtime."*

类型转换

覆盖默认值

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

构造器注入 VS. Setter 注入

- 《Expert One-on-One™ J2EE™ Development without EJB™》认为 Setter 注入的缺点：

"*Disadvantages include:*

The order in which setters are called is not expressed in any contract. Thus, we sometimes need to invoke a method after the last setter has been called to initialize the component. Spring provides the org.springframework.beans.factory.InitializingBean interface for this; it also provides the ability to invoke an arbitrary init method. However, this contract must be documented to ensure correct use outside a container.

setter 不能约束顺序

Not all the necessary setters may have been called before use. The object can thus be left partially configured."

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

构造器注入 VS. Setter 注入

- 《Expert One-on-One™ J2EE™ Development without EJB™》认为构造器注入的优点：

"*Advantages of Constructor Injection include:*

Each managed object is guaranteed to be in a consistent state—fully configured—before it can be invoked in any business methods. This is the primary motivation of Constructor Injection. (However, it is possible to achieve the same result with JavaBeans via dependency checking, as Spring can optionally perform.) There's no need for initialization methods.

There may be slightly less code than results from the use of multiple JavaBean methods, although will be no difference in complexity."

一致的状态

减少代码

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

构造器注入 VS. Setter 注入

- 《Expert One-on-One™ J2EE™ Development without EJB™》认为构造器注入的缺点：

"*Disadvantages include:*

- *Although also a Java-language feature, multi-argument constructors are probably less common in existing code than use of JavaBean properties.*
- *Java constructor arguments don't have names visible by introspection.*
- *Constructor argument lists are less well supported by IDEs than JavaBean setter methods.*
- *Long constructor argument lists and large constructor bodies can become unwieldy.*
- *Concrete inheritance can become problematic.*
- *Poor support for optional properties, compared to JavaBeans*
- *Unit testing can be slightly more difficult*
- *When collaborators are passed in on object construction, it becomes impossible to change the reference held in the object."*

内省

来自“Chapter 6. Lightweight Containers and Inversion of Control” 节选

面试题

沙雕面试题 - 什么是 IoC ？

答：简单地说，IoC 是反转控制，类似于好莱坞原则，主要有依赖查找和依赖注入实现

面试题

996 面试题 - 依赖查找和依赖注入的区别？

答：依赖查找是主动或手动的依赖查找方式，通常需要依赖容器或标准 API 实现。而依赖注入则是手动或自动依赖绑定的方式，无需依赖特定的容器和 API

如servlet api

DI更便利

面试题

劝退面试题 - Spring 作为 IoC 容器有什么优势？

答：
典型的 IoC 管理，依赖查找和依赖注入
AOP 抽象
事务抽象
事件机制 EventObject、EventListener
SPI 扩展
强大的第三方整合
易测试性
更好的面向对象

面试题

02

996 面试题

Spring Framework 重要模块有哪些？

我真的没笑

02

面试题

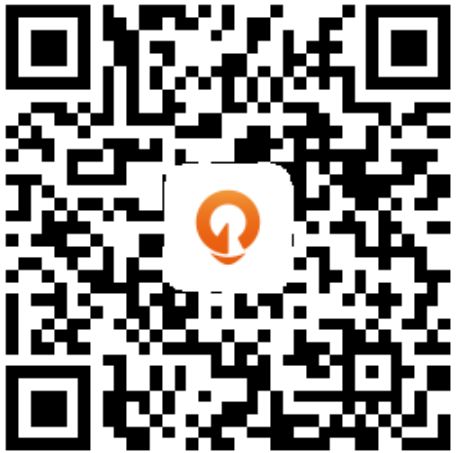
Spring Framework 重要模块有哪些？

03

劝退面试题

Spring Framework 的优势和不足是什么?

极客时间



扫码试看/订阅

《小马哥讲 Spring 核心编程思想》视频课程