

极客大学 Java 进阶训练营

第 9 课

Java 相关框架（1）



KimmKing

Apache Dubbo/ShardingSphere PMC

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

- 1.Spring 技术发展
- 2.Spring 框架设计*
- 3.Spring AOP 详解*
- 4.Spring Bean 核心原理*
- 5.Spring XML 配置原理*
- 6.Spring Messaging 等技术
- 7.第9课总结回顾与作业实践

第9课 1. Spring 技术发展

Spring 技术发展

Spring 框架的产生与发展

2002 年 10 月，Rod Johnson 撰写了一本名为 Expert One-on-One J2EE 设计和开发的书。

Rod, Juergen 和 Yann 于 2003 年 2 月左右开始合作开发 Spring 项目。

自 2004 年 1.0 版本发布以来，Spring 框架迅速发展。

Spring 2.0 于 2006 年 10 月发布，到那时，Spring 的下载量超过了 100 万。

在 Rod 领导下管理 Interface21 项目于 2007 年 11 月更名为 SpringSource。同时发布了 Spring 2.5。Spring 2.5 中的主要新功能包括支持 Java 6 / Java EE 5，支持注释配置，classpath 中的组件自动检测和兼容 OSGi 的 bundle。

2007 年，SpringSource 从基准资本获得了 A 轮融资（1000 万美元）。

2009 年 8 月，SpringSource 以 4.2 亿美元被 VMWare 收购。

2009 年 12 月，Spring 3.0 发布。

2012 年 7 月，Rod Johnson 离开了团队。



Spring 技术发展

Spring 框架的产生与发展

2013 年 4 月，VMware 和 EMC 通过 GE 投资创建了一家名为 Pivotal 的合资企业。所有的 Spring 应用项目都转移到了 Pivotal。

2013 年 12 月，Pivotal 宣布发布 Spring 框架 4.0。Spring 4.0 是 Spring 框架的一大进步，它包含了对 Java 8 的全面支持，更高的第三方库依赖性（groovy 1.8+，ehcache 2.1+，hibernate 3.6+等），Java EE 7 支持，groovy DSL for bean 定义，对 websockets 的支持以及对泛型类型的支持作为注入 bean 的限定符。

2014 年至 2017 年期间发布了许多 Spring 框架 4.xx 系列版本。

Spring 5.0 GA 版本于 2017 年 9 月 28 日发布。

Spring 5.0 开始支持 JDK 8 和 Java EE 7，同时兼容 JDK9。

全面支持 Servlet 3.1，还引入了一个全新的模块 Spring WebFlux。

用于替代老话的 spring-webmvc；对 Kotlin 也有了更好的支持。



Spring 技术发展





The screenshot shows the Spring framework's official website. At the top is a navigation bar with the Spring logo and links for 'Why Spring', 'Learn', 'Projects', 'Training', 'Support', 'Community', and a settings icon. The main hero section features the headline 'Spring makes Java simple.' with two buttons: 'WHY SPRING' and 'QUICKSTART'. Below this is a grid of seven feature cards, each with an icon, a title, and a brief description.

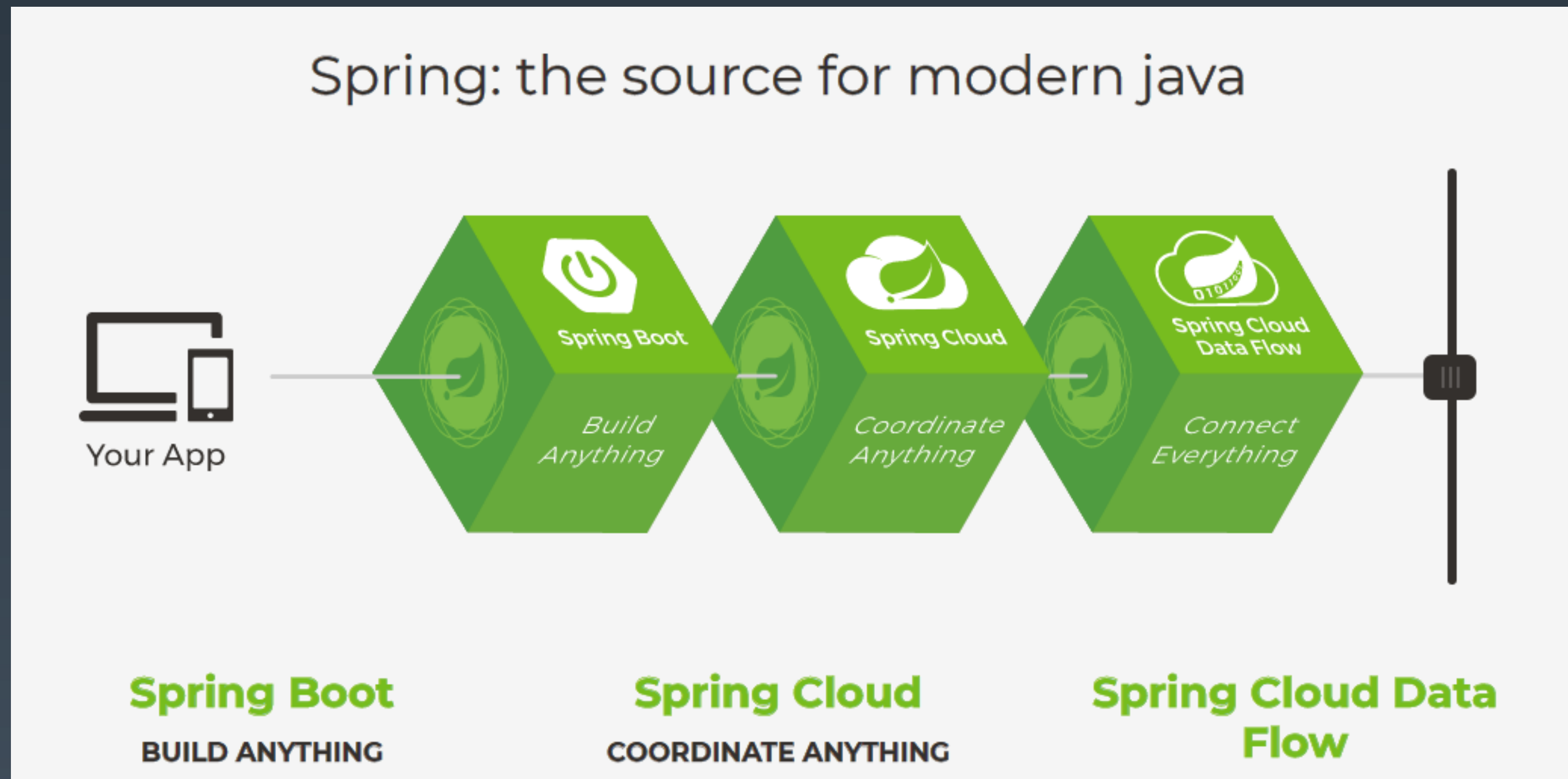
spring® Why Spring ▾ Learn ▾ Projects ▾ Training Support Community ▾ ⚙️

Spring makes Java simple.

[WHY SPRING](#) [QUICKSTART](#)

- **Microservices**
Quickly deliver production-grade features with independently evolvable microservices.
- **Reactive**
Spring's asynchronous, nonblocking architecture means you can get more from your computing resources.
- **Cloud**
Your code, any cloud—we've got you covered. Connect and scale your services, whatever your platform.
- **Web apps**
Frameworks for fast, secure, and responsive web applications connected to any data store.
- **Serverless**
The ultimate flexibility. Scale up on demand and scale to zero when there's no demand.
- **Event Driven**
Integrate with your enterprise. React to business events. Act on your streaming data in realtime.
- **Batch**
Automated tasks. Offline processing of data at a time to suit you.

Spring 技术发展





Pivotal 公司成立之后，于 2014 年发布了 Spring Boot，2015 年发布了 Spring Cloud，2018 年 Pivotal 公司在纽约上市。公司的开源产品有：Spring 以及 Spring 衍生产品、Web 服务器 Tomcat、缓存中间件 Redis、消息中间件 RabbitMQ、平台即服务的 Cloud Foundry、Greenplum 数据引擎、GemFire（12306 系统解决方案组件之一）。

Spring 技术发展

Spring 框架的产生与发展

Spring Framework

5.2.9.RELEASE

OVERVIEWLEARN

Documentation

Each **Spring project** has its own; it explains in great details how you can use **project features** and what you can achieve with them.

5.2.9.RELEASE CURRENT GA	Reference Doc.	API Doc.
5.3.0-SNAPSHOT SNAPSHOT	Reference Doc.	API Doc.
5.3.0-RC2 PRE	Reference Doc.	API Doc.
5.2.10.BUILD-SNAPSHOT SNAPSHOT	Reference Doc.	API Doc.
5.1.19.BUILD-SNAPSHOT SNAPSHOT	Reference Doc.	API Doc.
5.1.18.RELEASE GA	Reference Doc.	API Doc.
5.0.19.RELEASE GA	Reference Doc.	API Doc.
4.3.29.RELEASE GA	Reference Doc.	API Doc.

2.5.6

3.3.1

4.x

2.Spring 框架设计*

Spring 框架设计

思考一下：什么是框架？

你理解的 Spring 框架是什么呢？

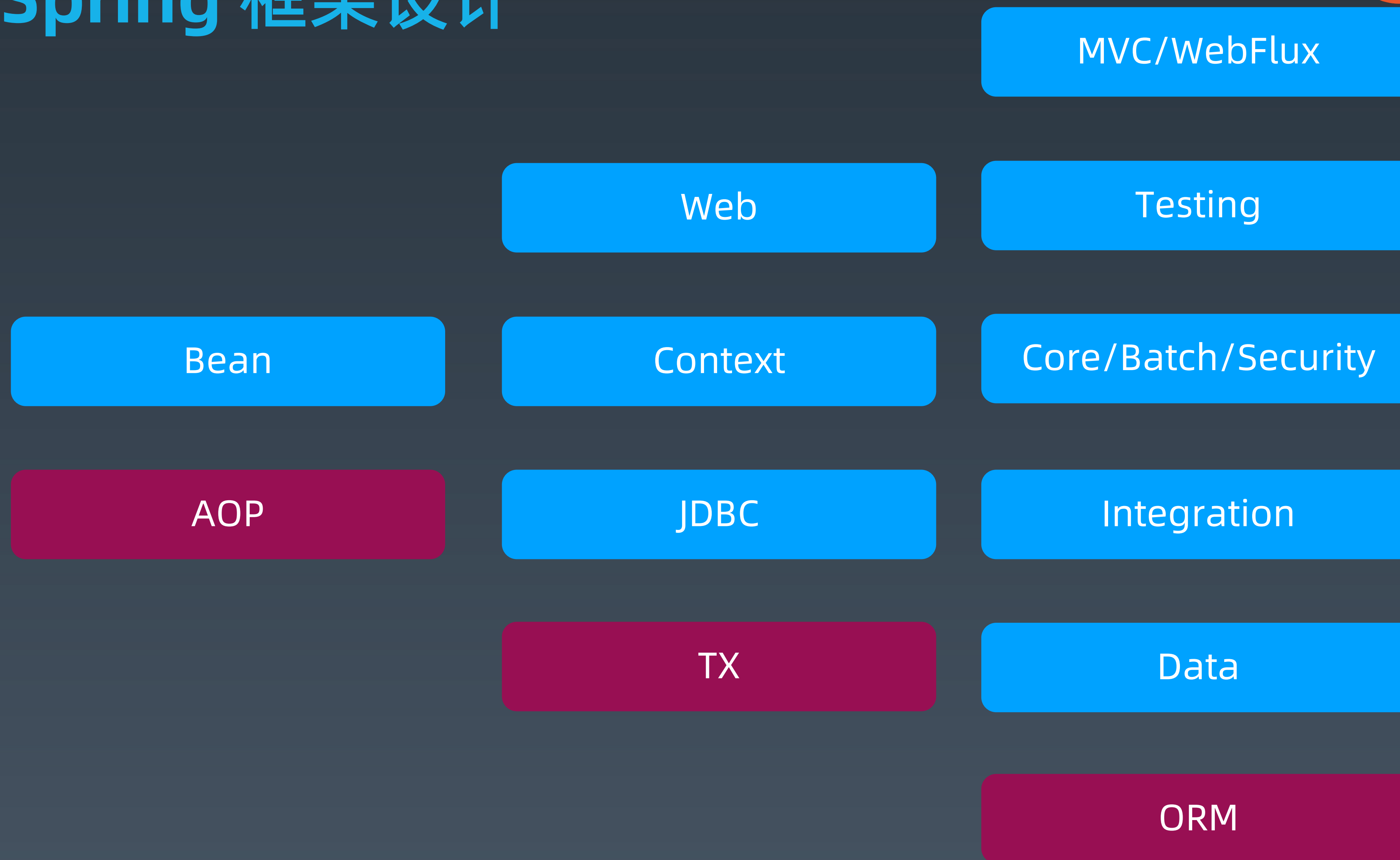
Spring framework 6大模块

1. Core: Bean/Context/AOP
2. Testing: Mock/TestContext
3. DataAccess: Tx/JDBC/ORM
4. Spring MVC/WebFlux: web

-
5. Integration: remoting/JMS/WS
 6. Languages: Kotlin/Groovy

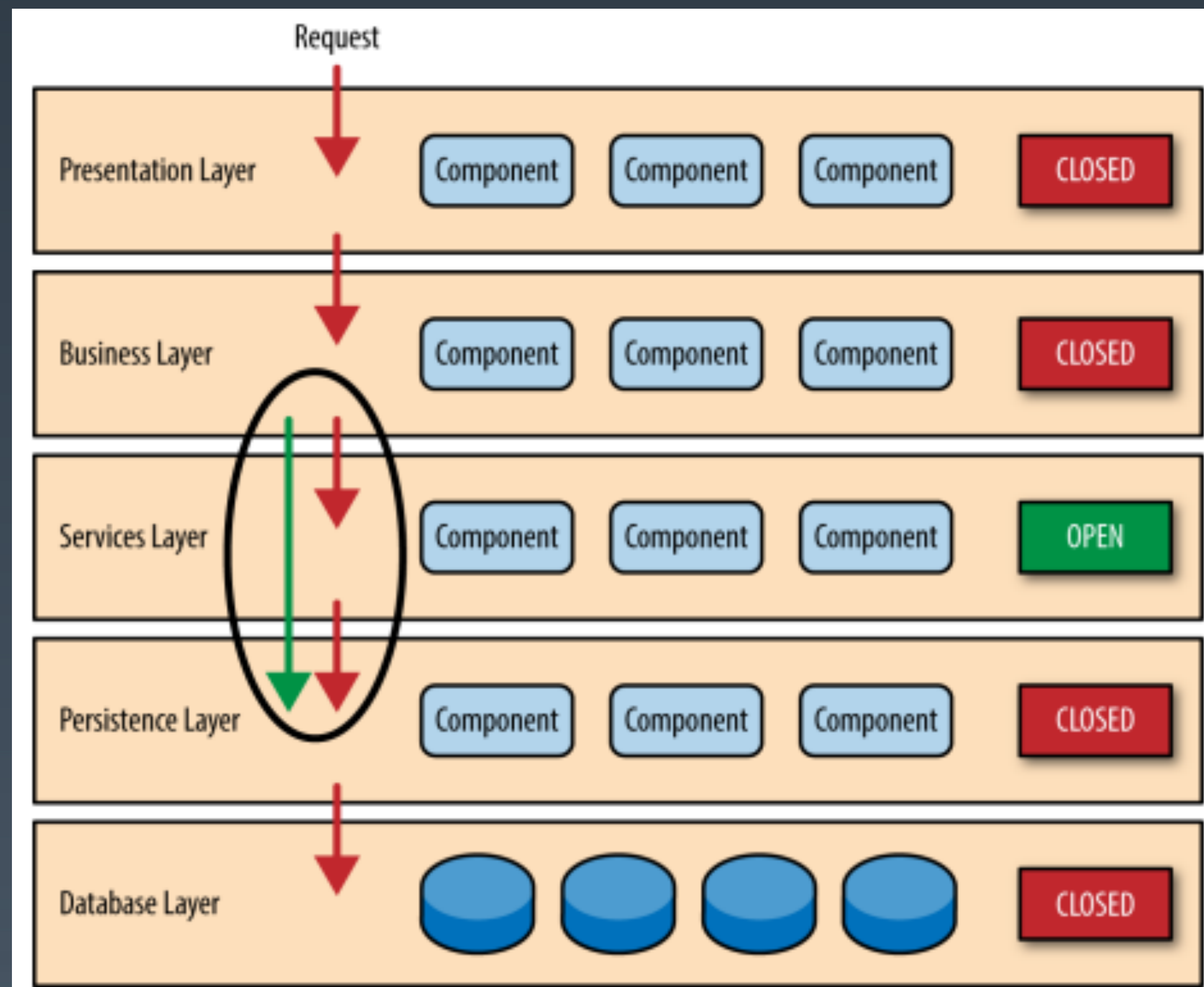
4 个常用模块

Spring 框架设计



Spring 框架设计

引入 Spring 意味着引入了一种研发协作模式



3.Spring AOP 详解*

Spring AOP

AOP-面向切面编程

Spring 早期版本的核心功能，管理对象生命周期与对象装配。

为了实现管理和装配，一个自然而然的想法就是，加一个中间层代理（字节码增强）来实现所有对象的托管。

IoC-控制反转

也称为 DI（Dependency Injection）依赖注入。

对象装配思路的改进。

从对象 A 直接引用和操作对象 B，变成对象 A 里指需要依赖一个接口 IB，系统启动和装配阶段，把 IB 接口的实例对象注入到对象 A，这样 A 就不需要依赖一个 IB 接口的具体实现，也就是类 B。

从而可以实现在不修改代码的情况，修改配置文件，即可以运行时替换成注入 IB 接口另一实现类 C 的一个对象实例。

什么类型的循环依赖 Spring 无法处理？
除了 Spring，循环依赖还有哪些类似场景？

Spring AOP

接口类型

默认使用 JdkProxy

`com.sun.proxy.$Proxy`

`proxyTargetClass`

`EnhancerBySpringCGLIB`

非接口类型

默认使用 CGLib

`EnhancerBySpringCGLIB`

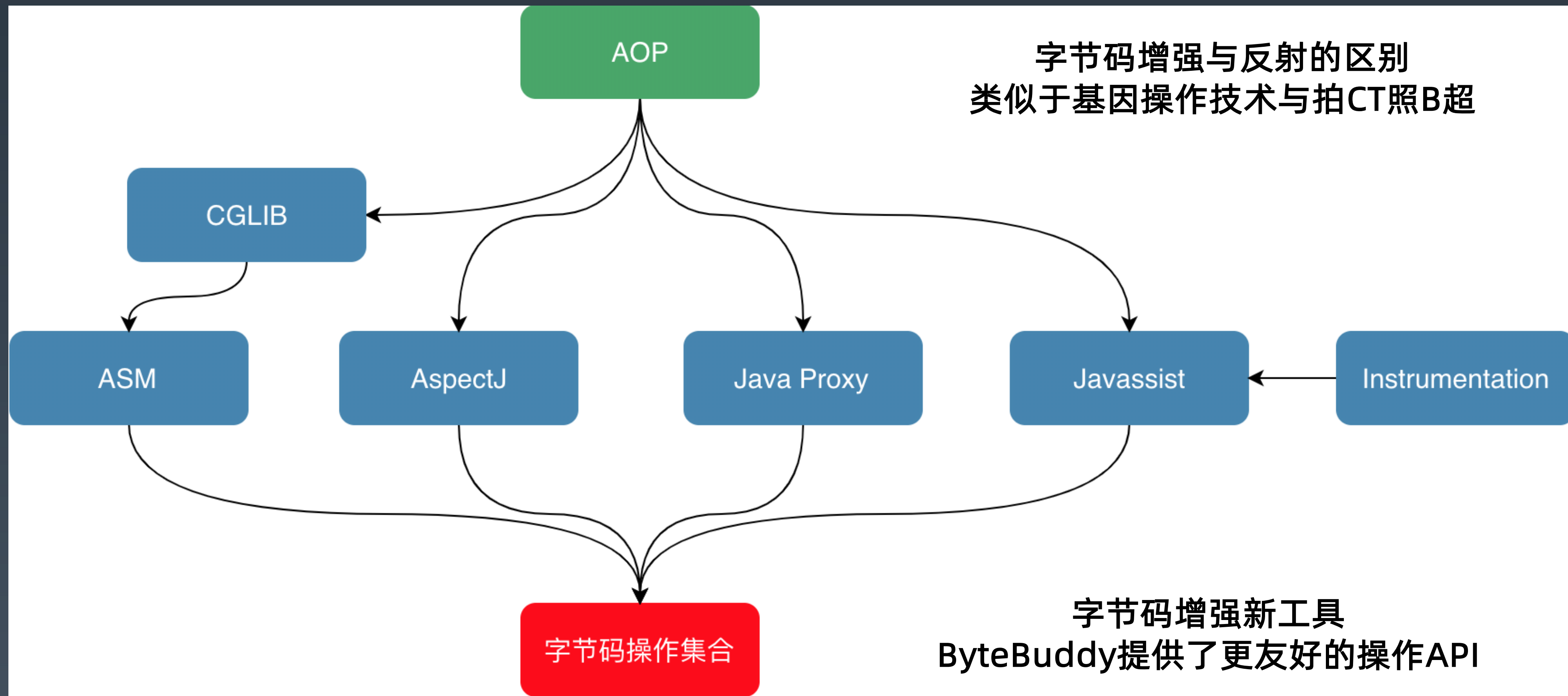
一个对象的代理有哪些种类？用在什么场景？

Spring AOP

AOP-面向切面编程

演示 AOP 的例子

Spring AOP

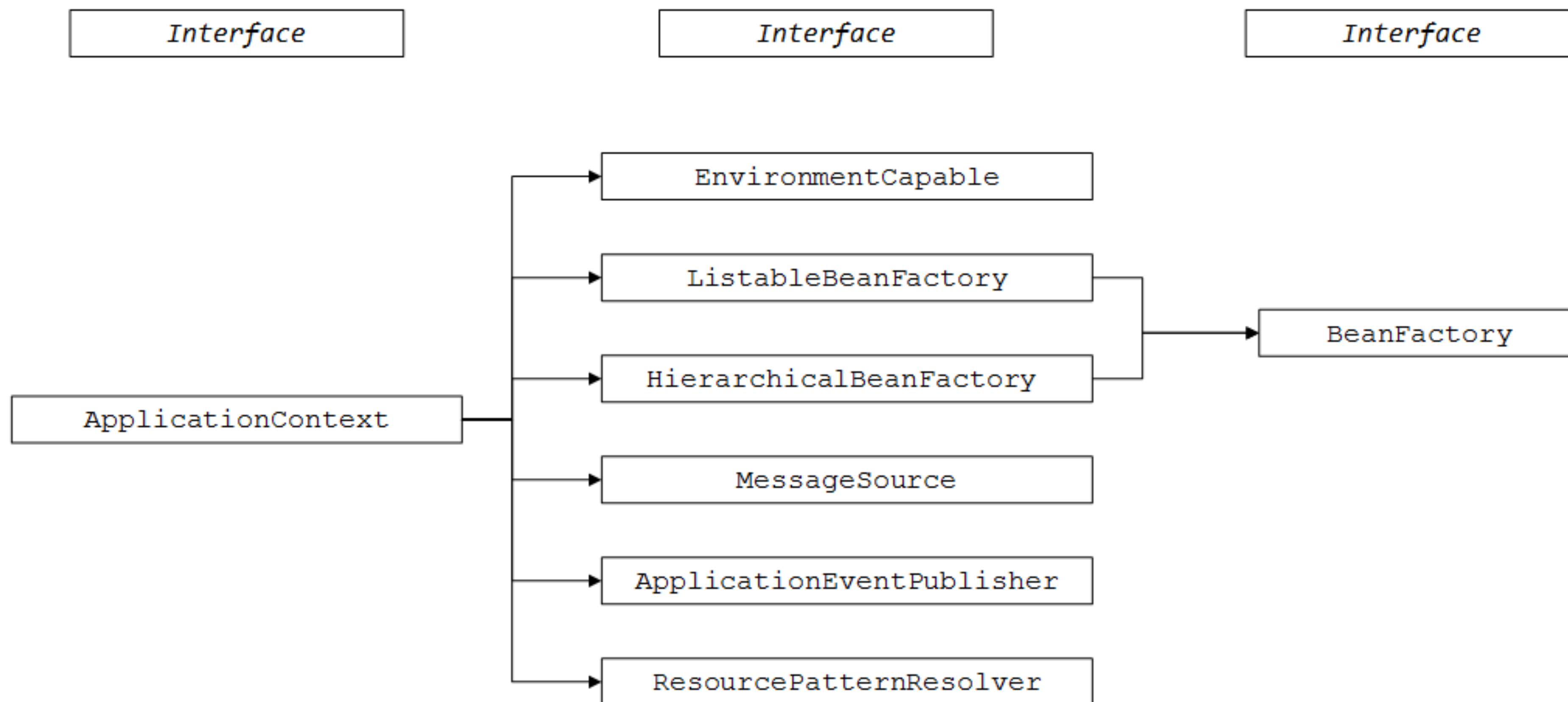


字节码增强有哪些类似 Cglib 工具?

4.Spring Bean 核心原理*

Spring Bean 生命周期

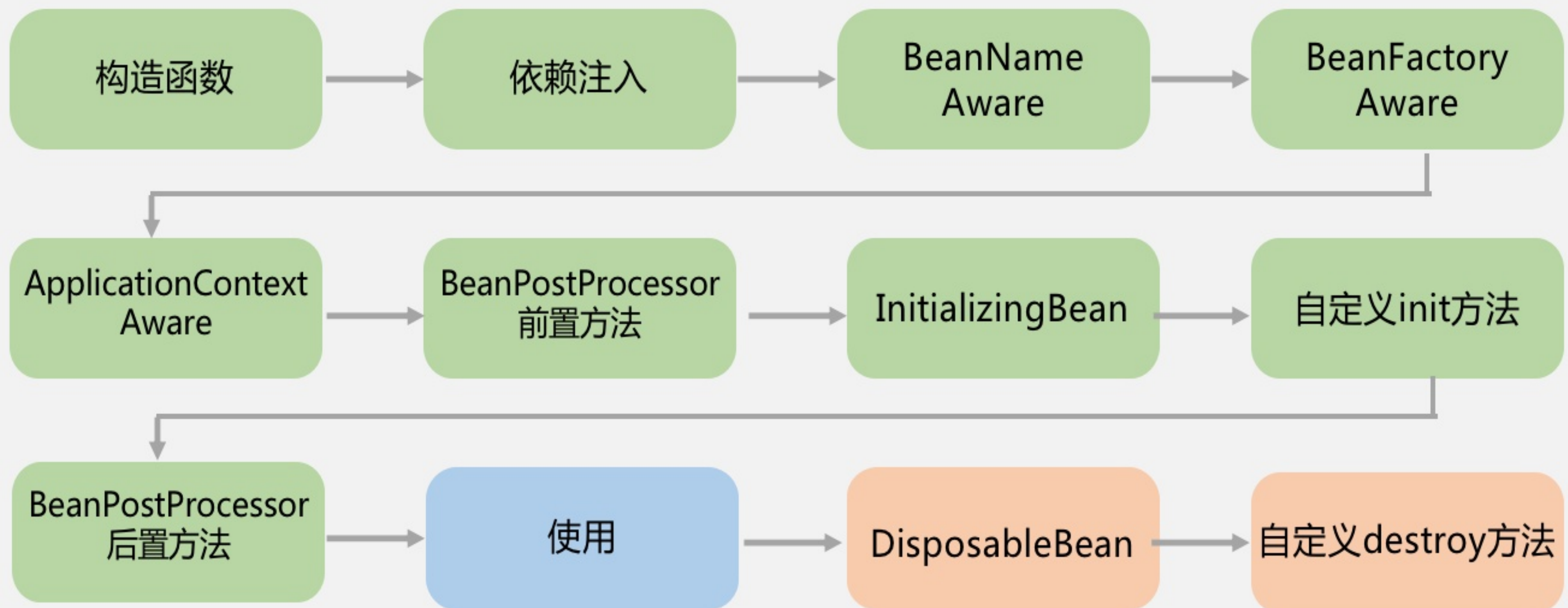
Bean 的加载过程



从 Bean 工厂到应用上下文

Spring Bean 生命周期

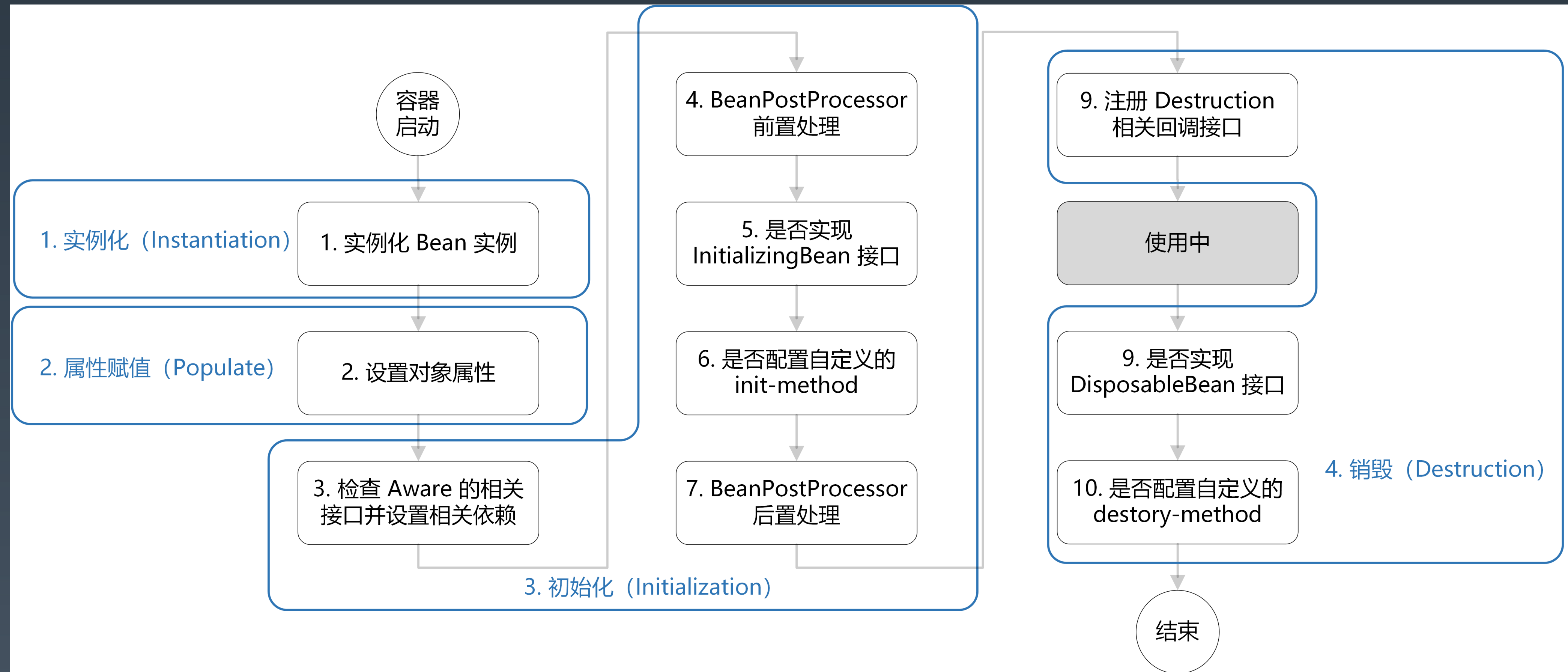
Bean 的加载过程



为什么设计得这么复杂？

Spring Bean 生命周期

Bean 的加载过程



是否可以对照 Classloader 加载?

Spring Bean 生命周期

Bean 的加载过程：

- 1) 创建对象
- 2) 属性赋值
- 3) 初始化
- 4) 注销接口注册

```
java 复制代码
// AbstractAutowireCapableBeanFactory.java
protected Object doCreateBean(final String beanName, final RootBeanDefinition mbd, final @Nullabl
    throws BeanCreationException {

    // 1. 实例化
    BeanWrapper instanceWrapper = null;
    if (instanceWrapper == null) {
        instanceWrapper = createBeanInstance(beanName, mbd, args);
    }

    Object exposedObject = bean;
    try {
        // 2. 属性赋值
        populateBean(beanName, mbd, instanceWrapper);
        // 3. 初始化
        exposedObject = initializeBean(beanName, exposedObject, mbd);
    }

    // 4. 销毁-注册回调接口
    try {
        registerDisposableBeanIfNecessary(beanName, bean, mbd);
    }

    return exposedObject;
}
```

Spring Bean 生命周期

Bean 的加载过程：

- 1) 检查 Aware 装配
- 2) 前置处理、After 处理
- 3) 调用 init method
- 4) 后置处理

返回包装类

```
// AbstractAutowireCapableBeanFactory.java
protected Object initializeBean(final String beanName, final Object bean, @Nullable RootBeanDefir
// 3. 检查 Aware 相关接口并设置相关依赖
if (System.getSecurityManager() != null) {
    AccessController.doPrivileged((PrivilegedAction<Object>) () -> {
        invokeAwareMethods(beanName, bean);
        return null;
    }, getAccessControlContext());
}
else {
    invokeAwareMethods(beanName, bean);
}

// 4. BeanPostProcessor 前置处理
Object wrappedBean = bean;
if (mbd == null || !mbd.isSynthetic()) {
    wrappedBean = applyBeanPostProcessorsBeforeInitialization(wrappedBean, beanName);
}

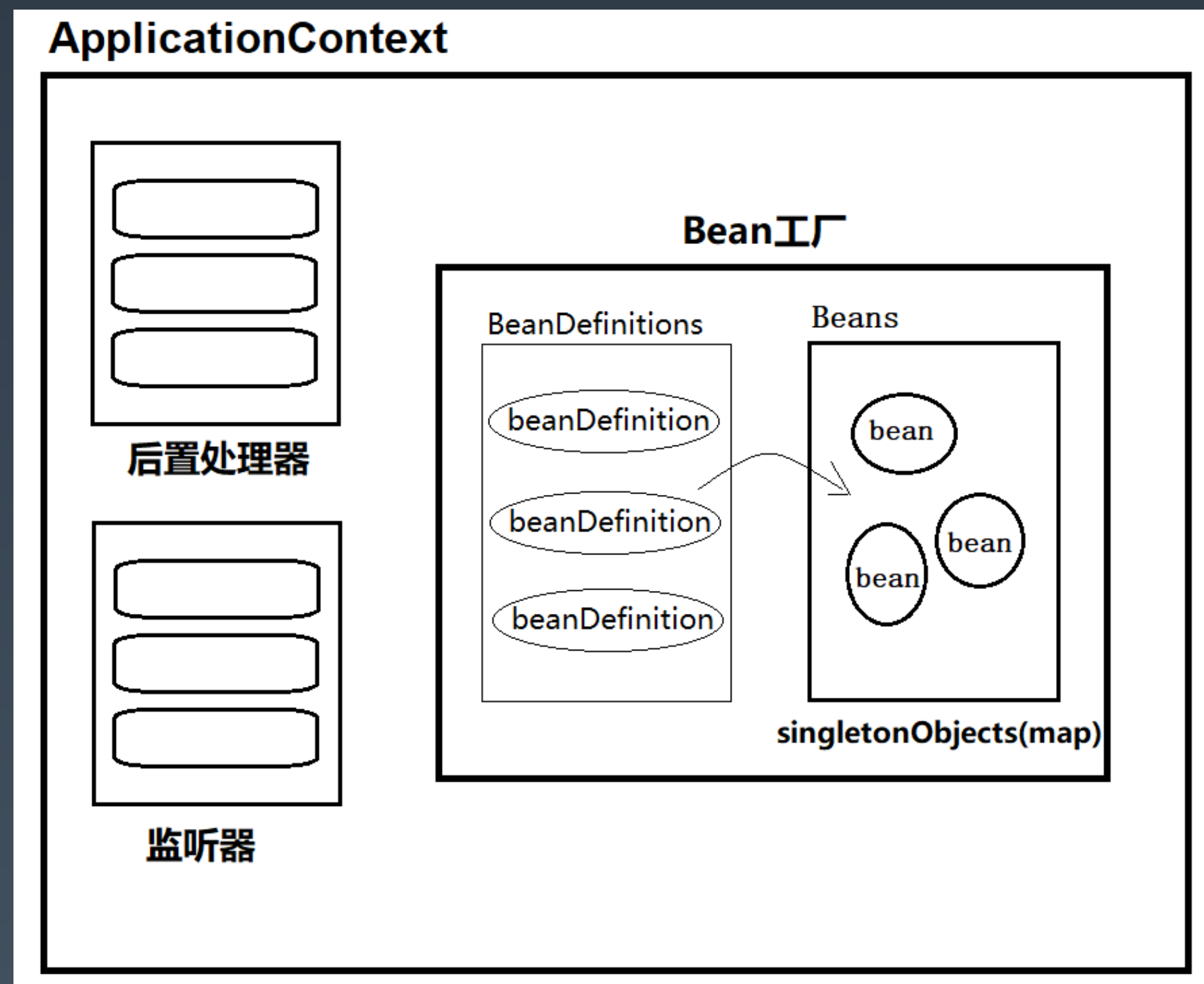
// 5. 若实现 InitializingBean 接口, 调用 afterPropertiesSet() 方法
// 6. 若配置自定义的 init-method方法, 则执行
try {
    invokeInitMethods(beanName, wrappedBean, mbd);
}
catch (Throwable ex) {
    throw new BeanCreationException(
        (mbd != null ? mbd.getResourceDescription() : null),
        beanName, "Invocation of init method failed", ex);
}

// 7. BeanPostProcessor 后置处理
if (mbd == null || !mbd.isSynthetic()) {
    wrappedBean = applyBeanPostProcessorsAfterInitialization(wrappedBean, beanName);
}

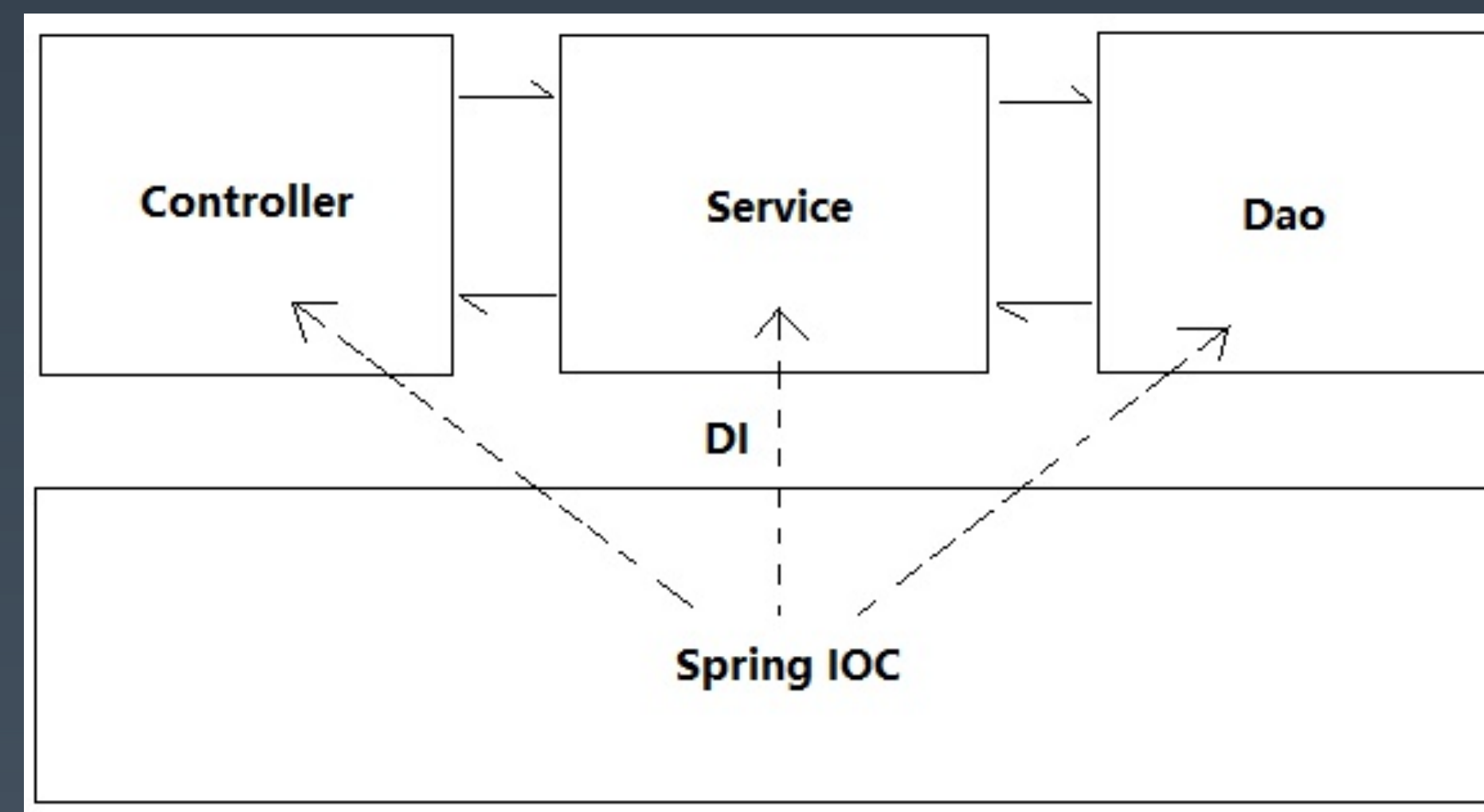
return wrappedBean;
}
```

Spring Bean 生命周期

Bean 的加载过程



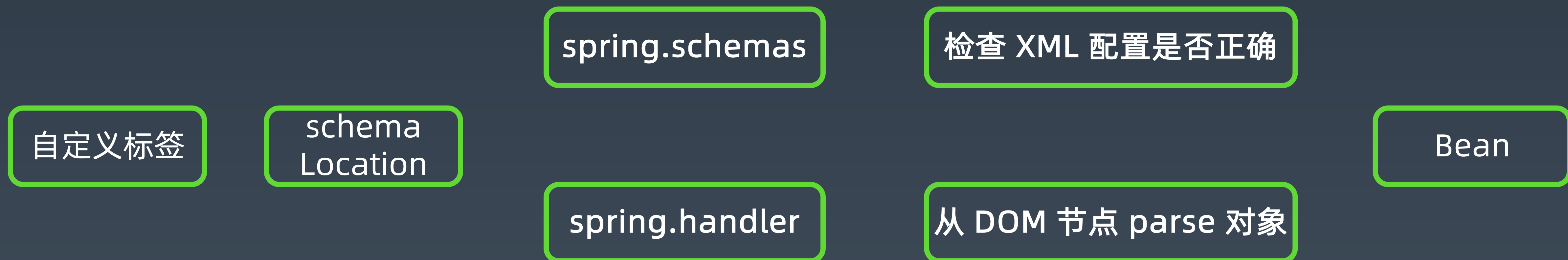
Spring 管理对象生命周期以后，也就改变了编程和协作模式。



5.Spring XML 配置原理*

Spring XML 配置原理

XML 配置原理：



Spring XML 配置原理

自动化 XML 配置工具：

XmlBeans -> Spring-xbean

2个原理：

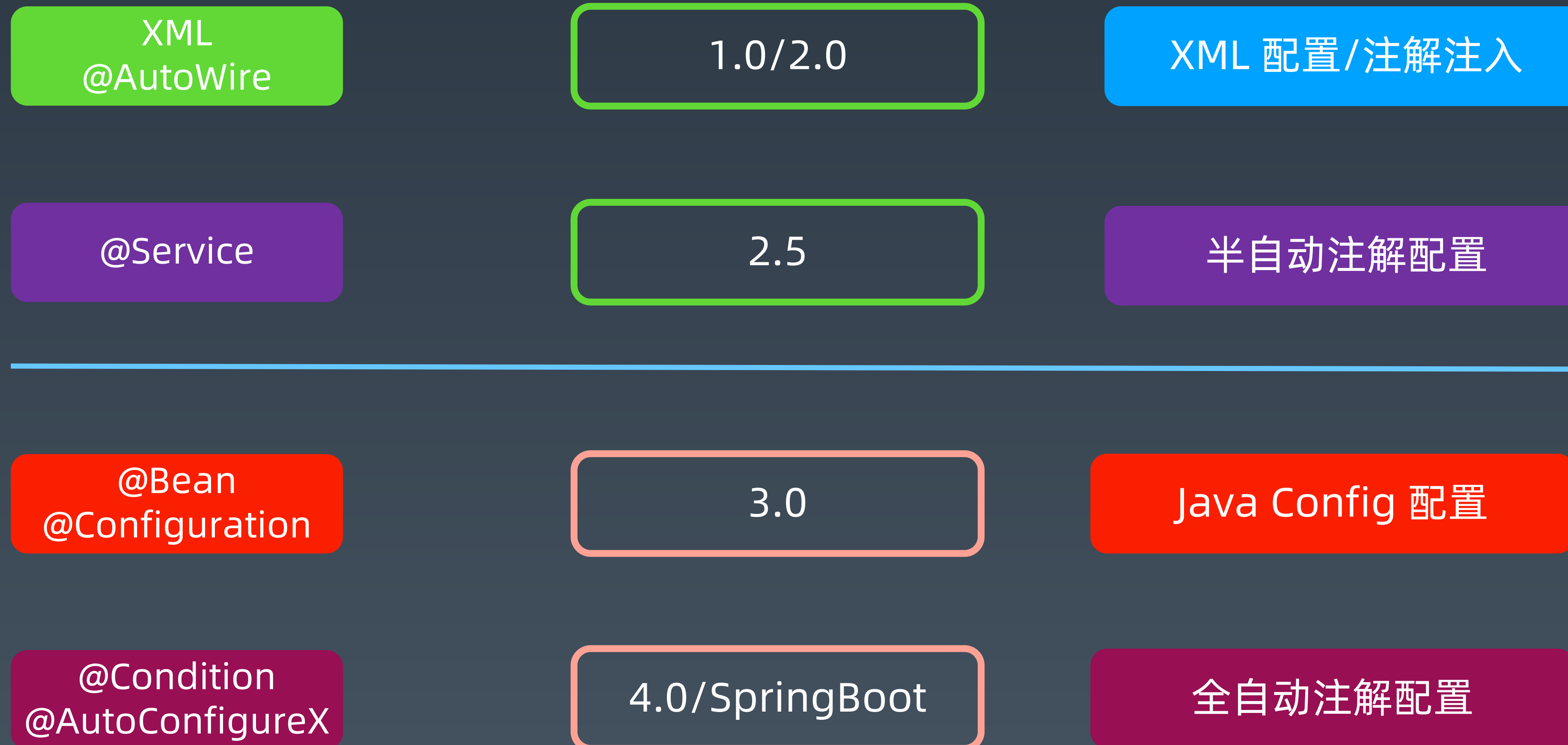
- 1、根据 Bean 的字段结构，自动生成 XSD
- 2、根据 Bean 的字段结构，配置 XML 文件

思考：1、解析 XML 的工具有哪些，都有什么特点？

2、XML <-> Bean 相互转换的工具，除了 xbean，还有什么？

演示自定义 XML 配置的例子

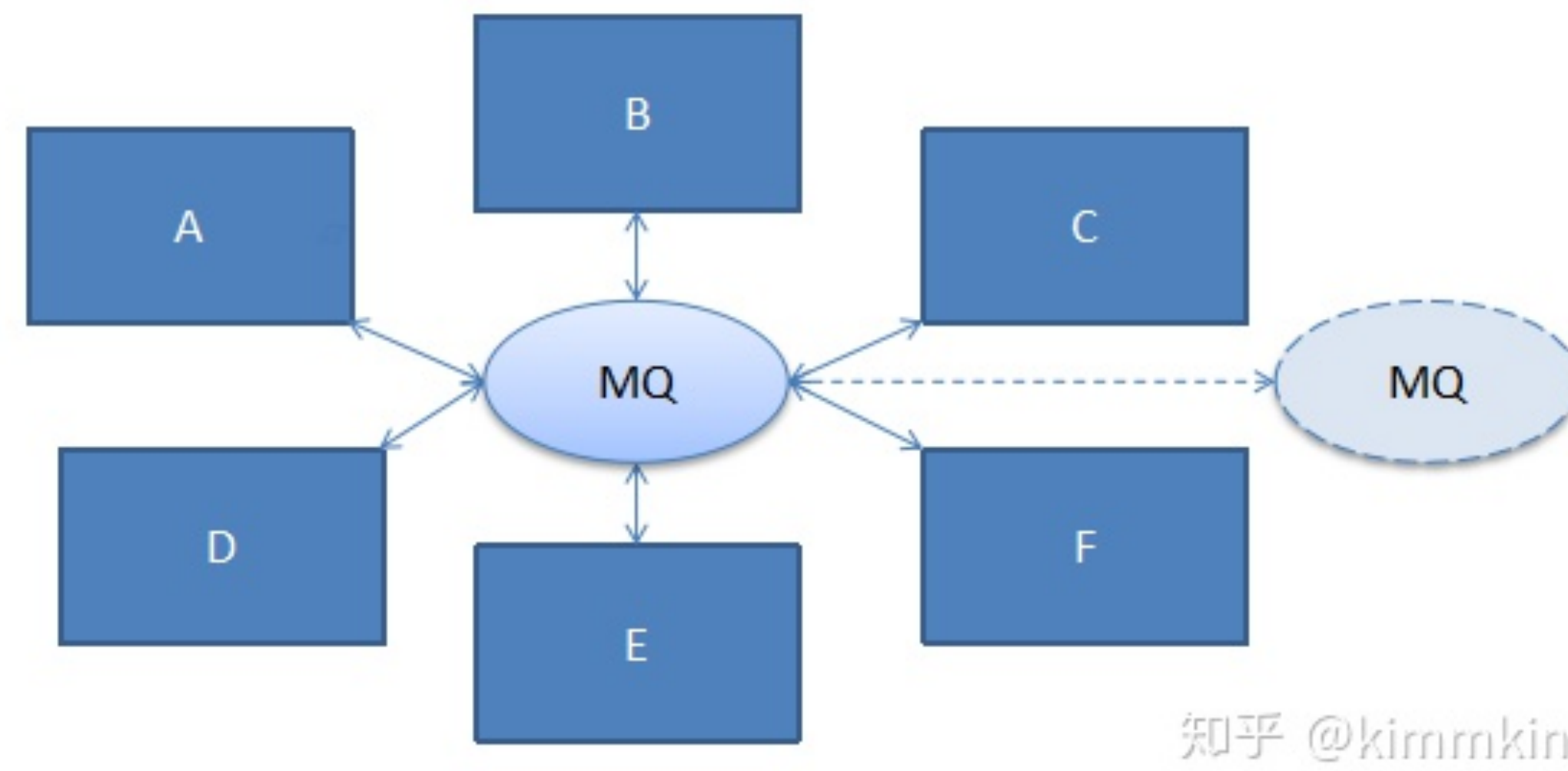
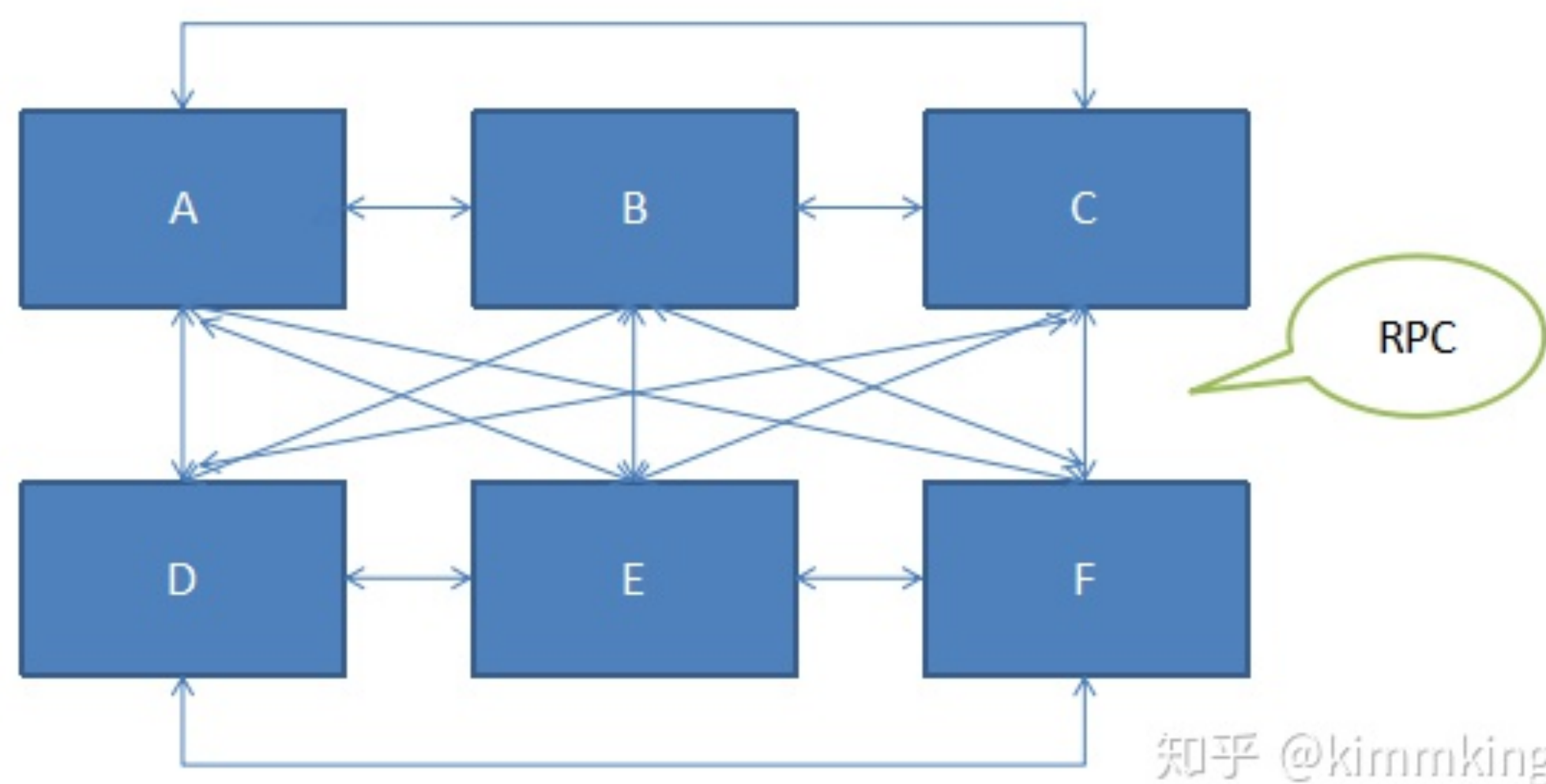
Spring Bean 配置方式演化



6.Spring Messaging 等技术

Spring Messaging 等技术

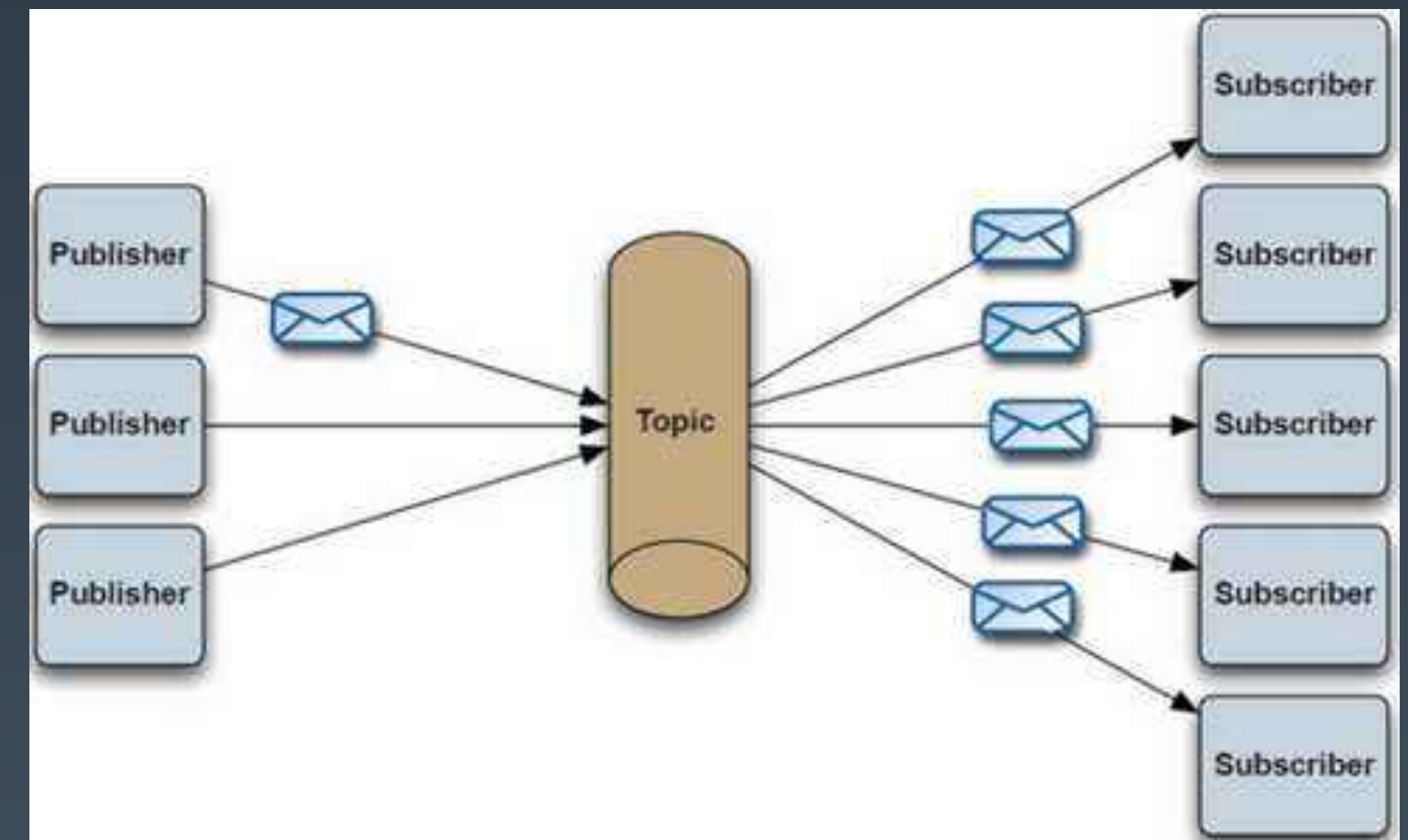
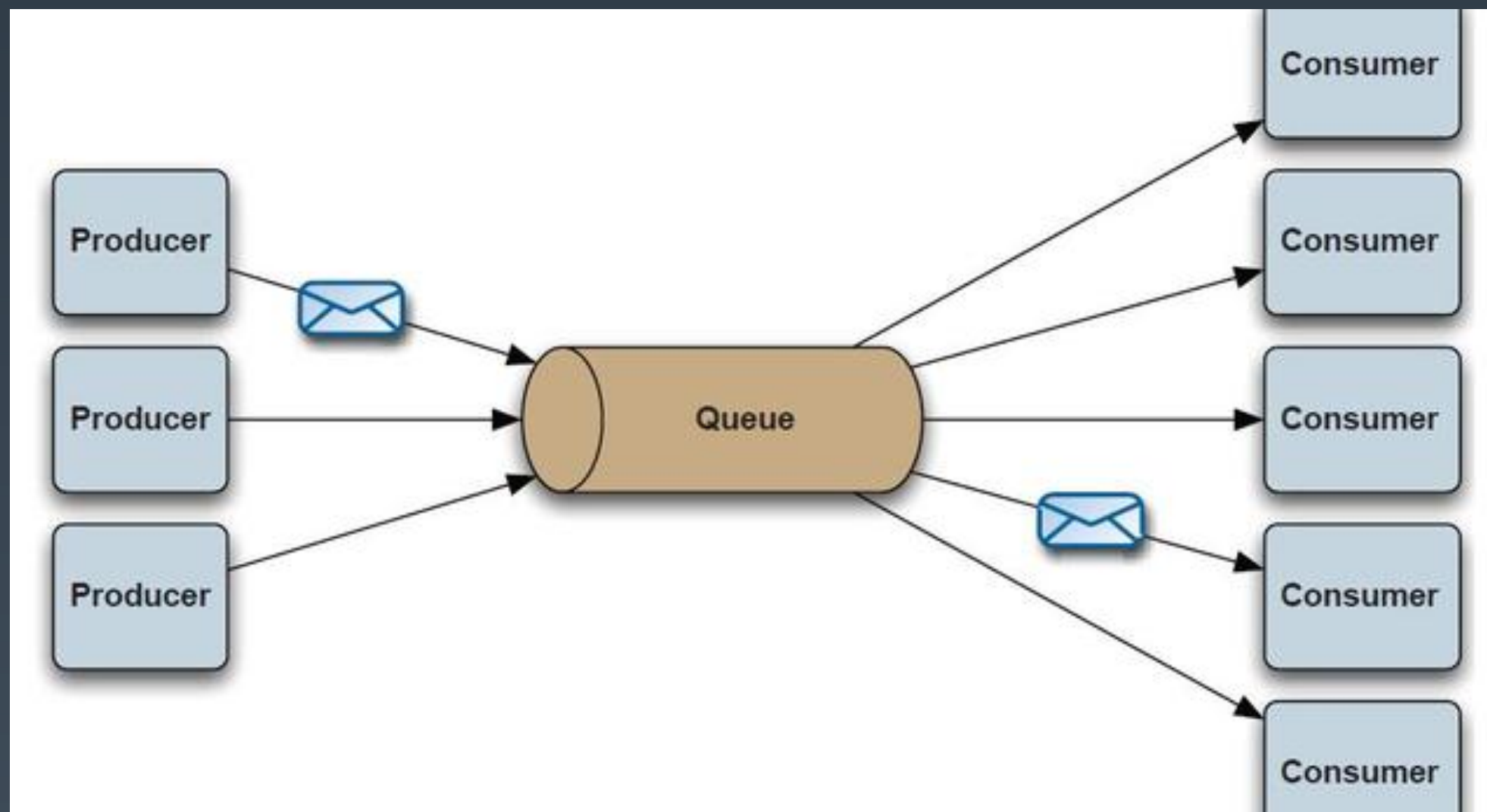
介绍 Messaging 与 JMS



同步转异步

Spring Messaging 等技术

介绍 Messaging 与 JMS



JMS 规范类似于 JDBC

Spring Messaging 等技术

介绍 Messaging 与 JMS

不要着急，《分布式消息》模块系统化讲解 MQ

演示 JMS 的例子

7.第9课总结回顾与作业实践

第 9 节课总结回顾

Spring 发展与框架

Spring AOP

Spring Bean

Spring XML 配置

Spring JMS 示例

第 9 节课作业实践

- 1、（选做）使 Java 里的动态代理，实现一个简单的 AOP。
- 2、（**必做**）写代码实现 Spring Bean 的装配，方式越多越好（XML、Annotation 都可以），提交到 Github。
- 3、（选做）实现一个 Spring XML 自定义配置，配置一组 Bean，例如：Student/Klass/School。
- 4、（*选做，会添加到高手附加题*）
 - 4.1 （挑战）讲网关的 frontend/backend/filter/router 线程池都改造成 Spring 配置方式；
 - 4.2 （挑战）基于 AOP 改造 Netty 网关，filter 和 router 使用 AOP 方式实现；
 - 4.3 （中级挑战）基于前述改造，将网关请求前后端分离，中级使用 JMS 传递消息；
 - 4.4 （中级挑战）尝试使用 ByteBuddy 实现一个简单的基于类的 AOP；
 - 4.5 （超级挑战）尝试使用 ByteBuddy 与 Instrument 实现一个简单 JavaAgent 实现无侵入下的 AOP。

THANKS! |  极客大学