



Spring核心编程



• KimmKing •

目录 | Contents

1. XXX
2. Java 8特性与常用工具
3. Spring框架与 AOP/loC
4. Spring Bean生命周期与核心源码
5. Spring ORM持久化与事务
6. Spring MVC实现REST
7. Spring Data/Messaging



二、Java 8特性与常用工具

面向对象5大编程原则:

S.O.L.I.D是面向对象设计和编程(OOD&OOP)中几个重要编码原则(Programming Principle)的首字母缩写。

SRP: The Single Responsibility Principle 单一责任原则

OCP: The Open Closed Principle 开放封闭原则

LSP: The Liskov Substitution Principle 里氏替换原则

ISP: The Interface Segregation Principle 接口分离原则

DIP: The Dependency Inversion Principle 依赖倒置原则

最小知识原则, KISS, 高内聚低耦合

设计模式与反模式

GoF 23 个经典设计模式（面向接口编程），

本质是一类特定场景下通用解决经验。

创建型

1. Factory Method (工厂方法)
2. Abstract Factory (抽象工厂)
3. Builder (建造者)
4. Prototype (原型)
5. Singleton (单例)

结构型

6. Adapter (适配器)
7. Bridge (桥接)
8. Composite (组合)
9. Decorator (装饰)
10. Facade (外观)
11. Flyweight (享元)
12. Proxy (代理)

行为型

13. Interpreter (解释器)
14. Template Method (模板方法)
15. Chain of Responsibility (责任链)
16. Command (命令)
17. Iterator (迭代器)
18. Mediator (中介者)
19. Memento (备忘录)
20. Observer (观察者)
21. State (状态)
22. Strategy (策略)
23. Visitor (访问者)



二、Java 8特性与常用工具

设计模式与反模式

模式的3个层次：解决方案层面（架构模式），组件层面（框架模式），代码层面（GoF设计模式）

其他模式：集成模式，事务模式，IO模式/Context模式，甚至状态机FSM，规则引擎RE，workflow都是模式。

反模式：死用模式，都是反模式。

二、Java 8特性与常用工具

集合相关操作:

1、JDK集合对象, List/Set/Vector, Queue/Stack, Map,

List的用法与注意事项;

Map的用法与注意事项;

集合的线程安全与否?

集合的可变与不可变?

Java的伪泛型与集合的关系,

泛型可以写 `class Student<A extends B>`, 如果A需要限定两个父接口B和C怎么办?

`enum AType`到底是什么?

二、Java 8特性与常用工具

集合相关操作：

2、集合工具类：

`java.util.Collections` 很有用：空集合，单对象集合，排序、反转、混淆、填充、最值，同步和不可修改。

二、Java 8特性与常用工具

Stream（流）是一个来自数据源的元素队列并支持聚合操作

- 元素：特定类型的对象，形成一个队列。Java中的Stream并不会存储元素，而是按需计算。
- 数据源：流的来源。可以是集合，数组，I/O channel，产生器generator 等。
- 聚合操作 类似SQL语句一样的操作，比如filter, map, reduce, find, match, sorted等。
- 和以前的Collection操作不同，Stream操作还有两个基础的特征：
- Pipelining：中间操作都会返回流对象本身。这样多个操作可以串联成一个管道，如同流式风格（fluent style）。这样做可以对操作进行优化，比如延迟执行(laziness)和短路(short-circuiting)。
- 内部迭代：以前对集合遍历都是通过Iterator或者For-Each的方式, 显式的在集合外部进行迭代，这叫做外部迭代。Stream提供了内部迭代的方式，通过访问者模式(Visitor)实现。

二、Java 8特性与常用工具

演示

java8 stream编程与lambda表达式

二、Java 8特性与常用工具

Lombok是什么？

Lombok是基于jsr269实现的一个非常神奇的 java 类库，会利用注解自动生成 java Bean 中烦人的 get、set方法及有参无参构造函数，还能自动生成 logger、ToString、HashCode、Builder 等 java特色的函数或是符合设计模式的方法，能够让你 java Bean 更简洁，更美观。

基于字节码增强，编译期处理。

可以配置开发工具IDE或Maven使用。

二、Java 8特性与常用工具

Guava 是什么?

Guava是一种基于开源的Java库，其中包含谷歌正在由他们很多项目使用的很多核心库。这个库是为了方便编码，并减少编码错误。这个库提供用于集合，缓存，支持原语，并发性，常见注解，字符串处理，I/O和验证的实用方法。

Guava的好处

- 标准化 - Guava库是由谷歌托管。
- 高效 - 可靠，快速和有效的扩展JAVA标准库。
- 优化 - Guava库经过高度的优化。

JDK8里的一些新特性源于Guava。

二、Java 8特性与常用工具

1. 基本工具 [Basic utilities]

让使用Java语言变得更舒适

1.1 使用和避免null: null是模棱两可的, 会引起令人困惑的错误, 有些时候它让人很不舒服。很多Guava工具类用快速失败拒绝null值, 而不是盲目地接受

1.2 前置条件: 让方法中的条件检查更简单

1.3 常见Object方法: 简化Object方法实现, 如hashCode()和toString()

1.4 排序: Guava强大的“流畅风格比较器”

1.5 Throwables: 简化了异常和错误的传播与检查

二、Java 8特性与常用工具

2. 集合[Collections]

Guava对JDK集合的扩展，这是Guava最成熟和为人所知的部分

2.1 不可变集合: 用不变的集合进行防御性编程和性能提升。

2.2 新集合类型: **multisets, multimaps, tables, bidirectional maps**等

2.3 强大的集合工具类: 提供java.util.Collections中没有的集合工具

2.4 扩展工具类: 让实现和扩展集合类变得更容易，比如创建Collection的装饰器，或实现迭代器

二、Java 8特性与常用工具

3. 缓存[Caches]

Guava Cache: 本地缓存实现, 支持多种缓存过期策略

4. 函数式风格[Functional idioms]

Guava的函数式支持可以显著简化代码, 但请谨慎使用它

5. 并发[Concurrency]

强大而简单的抽象, 让编写正确的并发代码更简单

5.1 ListenableFuture: 完成后触发回调的Future

5.2 Service框架: 抽象可开启和关闭的服务, 帮助你维护服务的状态逻辑

```
01 LoadingCache<Key, Graph> graphs = CacheBuilder.newBuilder()
02     .maximumSize(1000)
03     .expireAfterWrite(10, TimeUnit.MINUTES)
04     .removalListener(MY_LISTENER)
05     .build(
06         new CacheLoader<Key, Graph>() {
07             public Graph load(Key key) throws AnyException {
08                 return createExpensiveGraph(key);
09             }
10         });
```

```
01 ListeningExecutorService service =
02     MoreExecutors.listeningDecorator(Executors.newFixedThreadPool(10));
03 ListenableFuture explosion = service.submit(new Callable() {
04     public Explosion call() {
05         return pushBigRedButton();
06     }
07 });
08 Futures.addCallback(explosion, new FutureCallback() {
09     // we want this handler to run immediately after we push the big red button!
10     public void onSuccess(Explosion explosion) {
11         walkAwayFrom(explosion);
12     }
13     public void onFailure(Throwable thrown) {
14         battleArchNemesis(); // escaped the explosion!
15     }
16 });
```

二、Java 8特性与常用工具

6. 字符串处理[Strings]

非常有用的字符串工具，包括分割、连接、填充等操作

7. 原生类型[Primitives]

扩展 JDK 未提供的原生类型（如int、char）操作，包括某些类型的无符号形式

8. 区间[Ranges]

可比较类型的区间API，包括连续和离散类型

9. I/O

简化I/O尤其是I/O流和文件的操作，针对Java5和6版本

10. 散列[Hash]

提供比Object.hashCode()更复杂的散列实现，并提供Bloom过滤器的实现

二、Java 8特性与常用工具

11. 事件总线[EventBus]

发布-订阅模式的组件通信，进程内模块间解耦

12. 数学运算[Math]

优化的、充分测试的数学工具类

13. 反射[Reflection]

Guava 的 Java 反射机制工具类

JDK:

```
Foo foo = (Foo) Proxy.newProxyInstance(  
    Foo.class.getClassLoader(),  
    new Class<?>[] {Foo.class},  
    invocationHandler);
```

Guava:

```
Foo foo = Reflection.newProxy(Foo.class, invocationHandler);
```


三、Spring框架与 AOP/IoC

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。

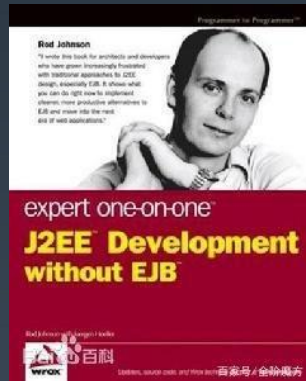
在 Rod 领导下管理 Interface21 项目于 2007 年 11 月更名为 SpringSource。同时发布了 Spring 2.5。Spring 2.5 中的主要新功能包括支持 Java 6 / Java EE 5，支持注释配置，classpath 中的组件自动检测和兼容 OSGi 的 bundle。

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。



三、Spring框架与 AOP/IoC



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。

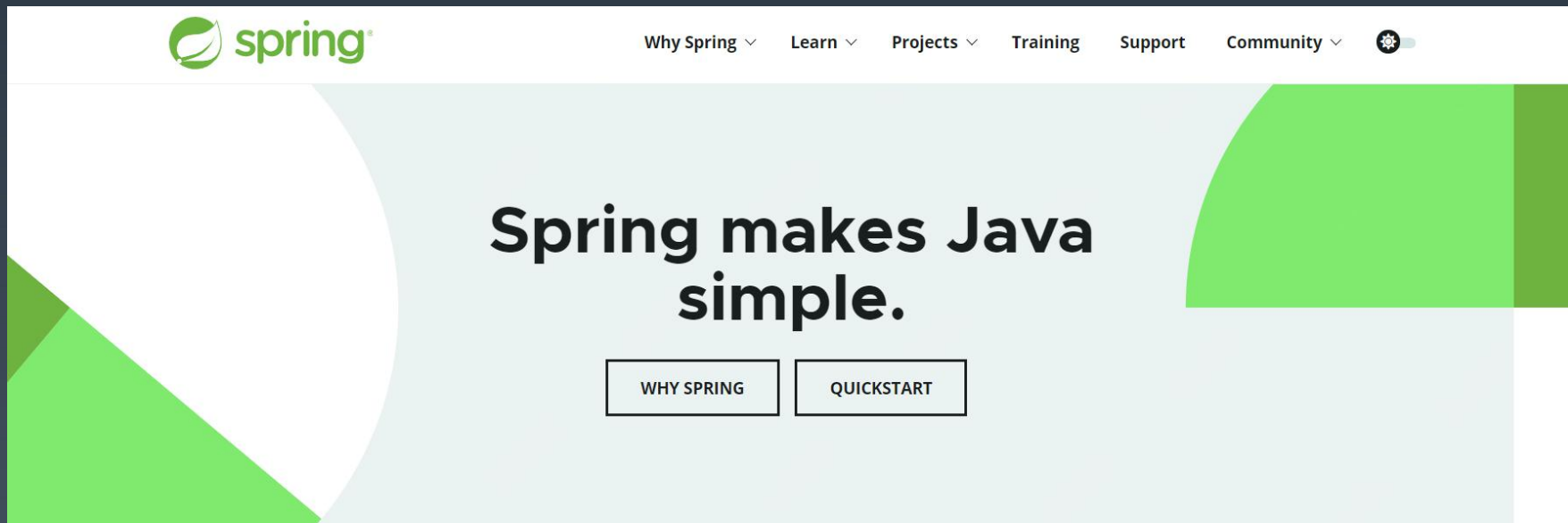
Spring 5.0 开始支持JDK 8和Java EE 7，同时兼容JDK9。

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。



三、Spring框架与 AOP/IOC



三、Spring框架与 AOP/IOC



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.

三、Spring框架与 AOP/IOC

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.

三、Spring框架与 AOP/IoC



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）。

三、Spring框架与 AOP/IOC

Spring framework 6大模块:

1. **Core technologies:** dependency injection, events, resources, i18n, validation, data binding, type conversion, SpEL, AOP.
2. **Testing:** mock objects, TestContext framework, Spring MVC Test, WebTestClient.
3. **Data Access:** transactions, DAO support, JDBC, ORM, Marshalling XML.
4. **Spring MVC/WebFlux:** web frameworks.
5. **Integration:** remoting, JMS, JCA, JMX, email, tasks, scheduling, cache.
6. **Languages:** Kotlin, Groovy, dynamic languages.

三、Spring框架与 AOP/IOC

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

也成为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。

三、Spring框架与 AOP/IoC

AOP的设计实现

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。

三、Spring框架与 AOP/IOC

Spring简单用法回顾

配置方式: XML、Annotation

组装方式: ByName, ByType, Lazy

自动装配: Autowired、Resource

定义Bean的方式

手工引用Bean、查看哪些Beans被加载

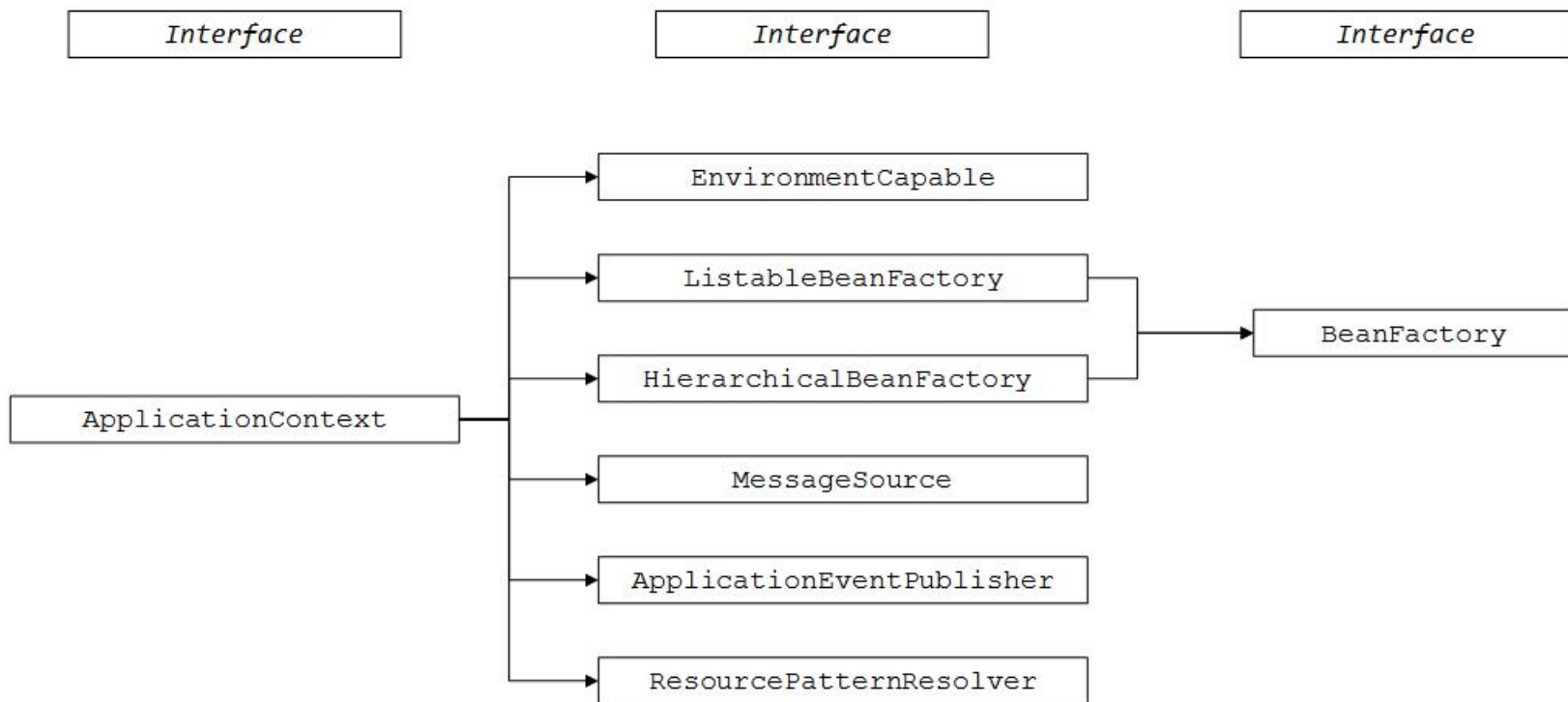
三、Spring框架与 AOP/IoC

示例：做一个简单的AOP

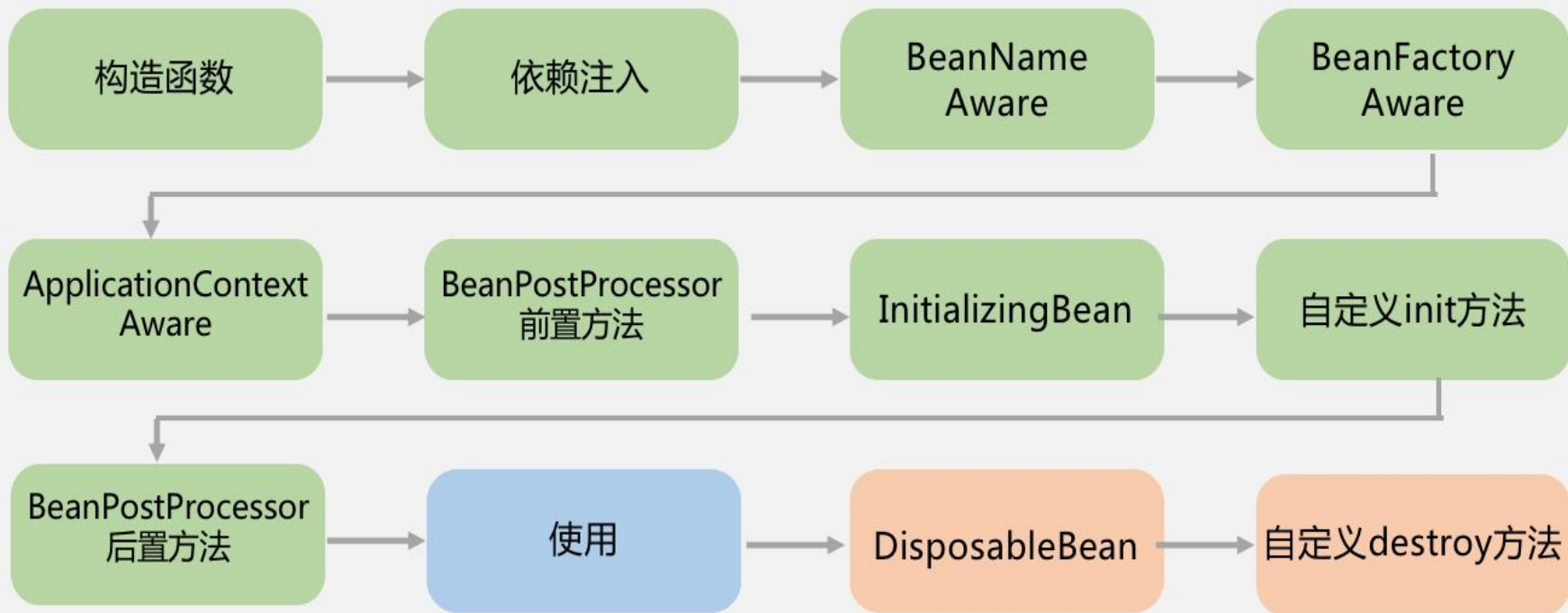
Spring早期版本的核心功能，管理对象生命周期与对象装配。

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。

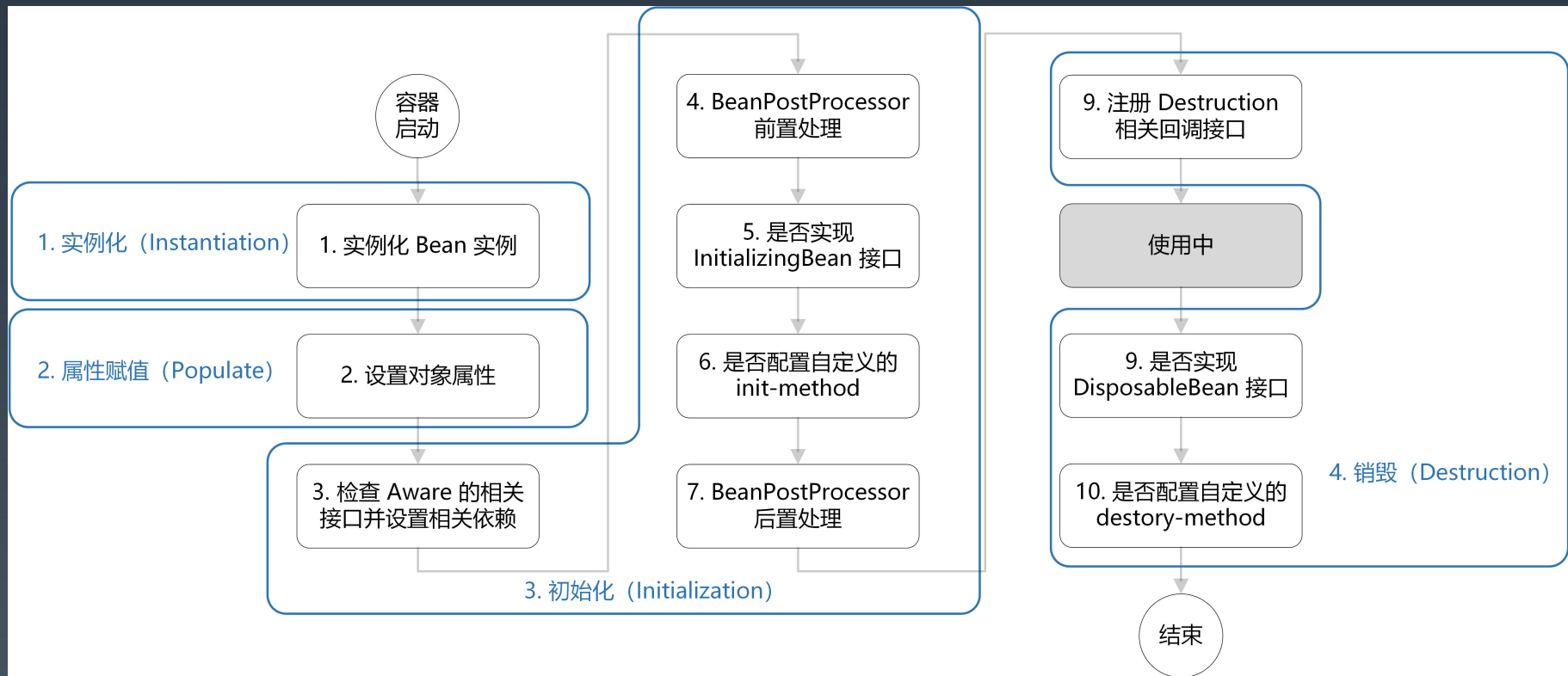
四、Spring Bean生命周期



四、Spring Bean生命周期



四、Spring Bean生命周期



四、Spring Bean生命周期

- 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 @Nullable
    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;
}
```

四、Spring Bean生命周期

- 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 RootBeanDefin
// 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;
}
```


四、Spring Bean生命周期

Aware 接口有：

BeanNameAware：注入当前 bean 对应 beanName；

BeanClassLoaderAware：注入加载当前 bean 的 ClassLoader；

BeanFactoryAware：注入 当前BeanFactory容器 的引用。

对于 ApplicationContext 类型的容器（通过 BeanPostProcessor）：

EnvironmentAware：注入 Enviroment，一般用于获取配置属性；

EmbeddedValueResolverAware：注入 EmbeddedValueResolver（Spring EL解析器），一般用于参数解析；

ApplicationContextAware（ResourceLoader、ApplicationEventPublisherAware、MessageSourceAware）：注入 ApplicationContext 容器本身。

四、Spring Bean生命周期

BeanPostProcessor 是 Spring 为修改 bean提供的强大扩展点，其可作用于容器中所有 bean。

```
java 复制代码

public interface BeanPostProcessor {

    // 初始化前置处理
    default Object postProcessBeforeInitialization(Object bean, String beanName) throws BeansException {
        return bean;
    }

    // 初始化后置处理
    default Object postProcessAfterInitialization(Object bean, String beanName) throws BeansException {
        return bean;
    }

}
```

四、Spring Bean生命周期

InitializingBean 和 init-method 是 Spring 为 bean 初始化提供的扩展点。

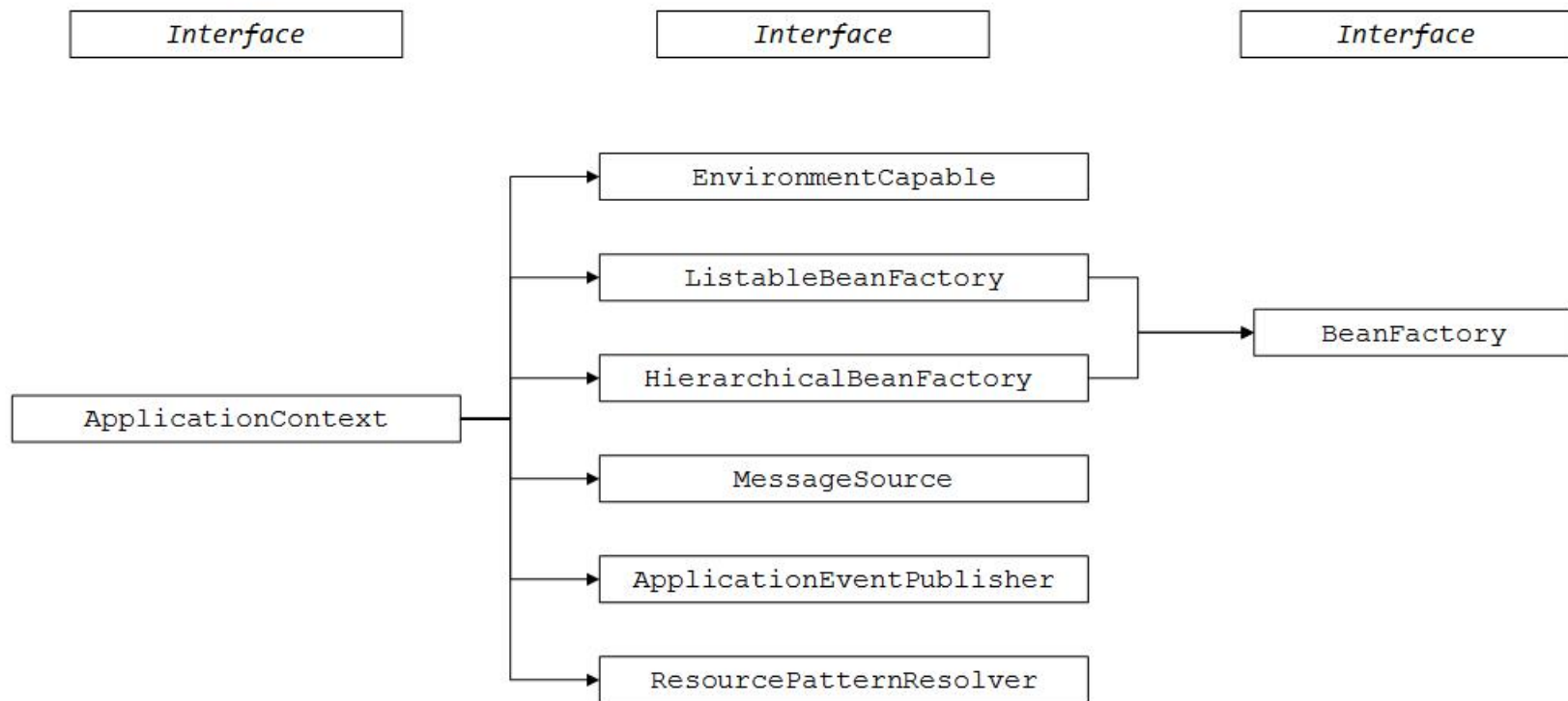
```
public interface InitializingBean {  
    void afterPropertiesSet() throws Exception;  
}
```

java 复制代码

```
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans"  
  
    <bean id="demo" class="com.chaycao.Demo" init-method="init()"/>  
  
</beans>
```

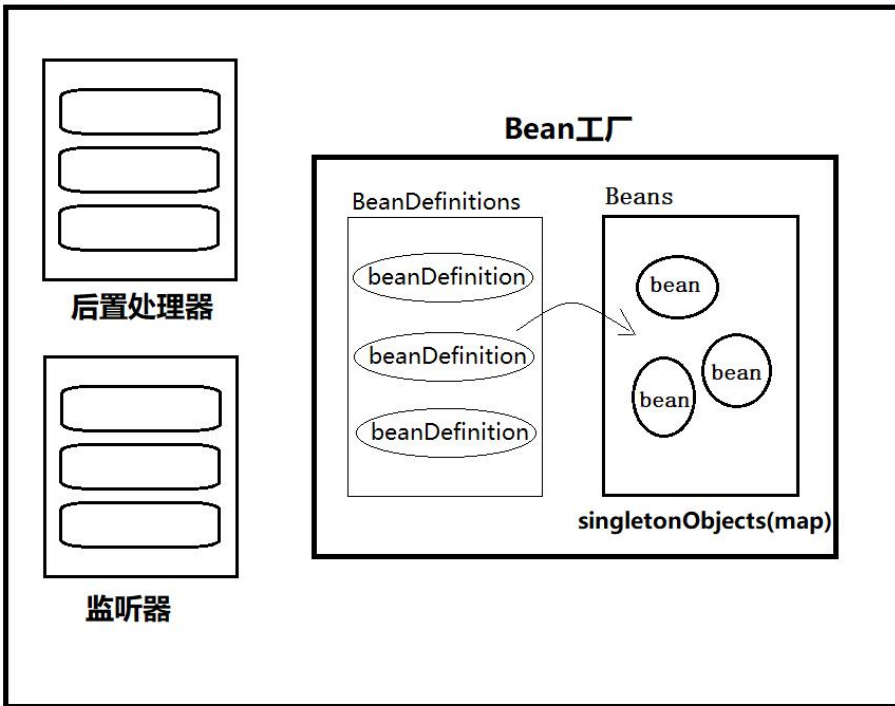
xml 复制代码

四、Spring Bean生命周期

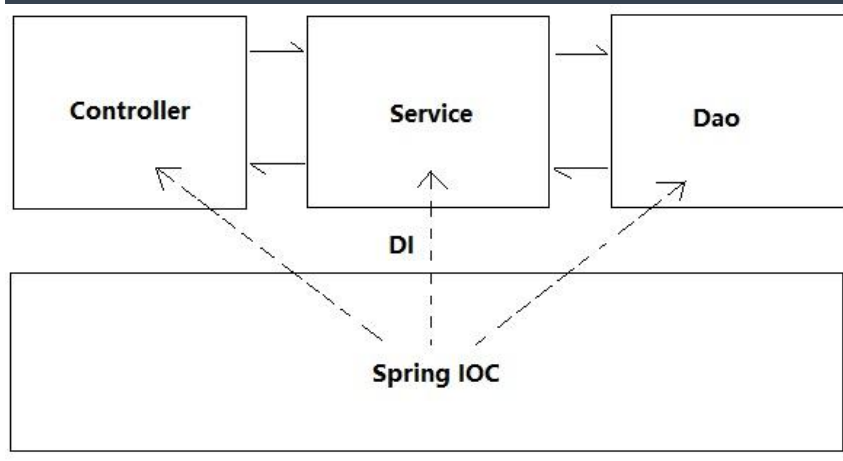


四、Spring Bean生命周期

ApplicationContext



Spring管理对象生命周期以后，也就改变了编程和协作模式。



四、Spring Bean生命周期

实例：

通过自定义namespace和xsd定制bean xml加载。

深入理解XML配置方式。

四、Spring Bean生命周期

XML配置的问题：太复杂了

怎么办？

从XML装配，走向注解Annotation。

注解的本质是什么？就是一个标记。

下面介绍12种常见注解。

四、Spring Bean生命周期

1.声明组件的注解

@Component 组件，没有明确的角色

@Service 在业务逻辑层使用（service层）

@Repository 在数据访问层使用（dao层）

@Controller 在展现层使用，控制器的声明（C）

四、Spring Bean生命周期

2.注入bean的注解

@Autowired: 由Spring提供

@Inject: 由JSR-330提供

@Resource: 由JSR-250提供

四、Spring Bean生命周期

3.java配置类相关注解

@Configuration 声明当前类为配置类，相当于xml形式的Spring配置（类上）

@Bean 注解在方法上，声明当前方法的返回值为一个bean，替代xml中的方式（方法上）

@Component注解，表明这个类是一个bean（类上）

@ComponentScan 用于对Component进行扫描，相当于xml中的（类上）

@WishlyConfiguration 为@Configuration与@ComponentScan的组合注解，可以替代这两个注解

四、Spring Bean生命周期

4.切面（AOP）相关注解

@Aspect 声明一个切面（类上）

使用@After、@Before、@Around定义建言（advice），可直接将拦截规则（切点）作为参数。

@After 在方法执行之后执行（方法上）

@Before 在方法执行之前执行（方法上）

@Around 在方法执行之前与之后执行（方法上）

@PointCut 声明切点

在java配置类中使用@EnableAspectJAutoProxy注解开启Spring对AspectJ代理的支持（类上）

四、Spring Bean生命周期

5.@Bean的属性支持

@Scope 设置Spring容器如何新建Bean实例（方法上，得有@Bean）

其设置类型包括：

Singleton （单例,一个Spring容器中只有一个bean实例，默认模式），

Protetype （每次调用新建一个bean），

Request （web项目中，给每个http request新建一个bean），

Session （web项目中，给每个http session新建一个bean），

GlobalSession （给每一个 global http session新建一个Bean实例）

@StepScope 在Spring Batch中还有涉及

@PostConstruct 由JSR-250提供，在构造函数执行完之后执行，等价于xml配置文件中bean的initMethod

@PreDestory 由JSR-250提供，在Bean销毁之前执行，等价于xml配置文件中bean的destroyMethod

四、Spring Bean生命周期

6.@Value注解: @Value 为属性注入值 (属性上)

支持如下方式的注入:

》注入普通字符 @Value("Michael Jackson")String name;

》注入操作系统属性 @Value("#{systemProperties['os.name']}")String osName;

》注入表达式结果 @Value("#{ T(java.lang.Math).random()* 100 }") String randomNumber;

》注入其它bean属性 @Value("#{domeClass.name}")String name;

》注入文件资源 @Value("classpath:com/hgs/hello/test.txt")String Resource file;

》注入URL资源 @Value("http://www.javastack.cn")Resource url;

》注入配置文件 Value("\${book.name}")String bookName;

注入配置使用方法:

① 编写配置文件 (test.properties) book.name= 《三体》

② @PropertySource 加载配置文件(类上) @PropertySource("classpath:com/hgs/hello/test/test.propertie")

③ 还需配置一个PropertySourcesPlaceholderConfigurer的bean。

四、Spring Bean生命周期

7.环境切换

@Profile 通过设定Environment的ActiveProfiles来设定当前context需要使用的配置环境。（类或方法上）

@Conditional Spring4中可以使用此注解定义条件话的bean，通过实现Condition接口，并重写matches方法，从而决定该bean是否被实例化。（方法上）

四、Spring Bean生命周期

8.异步相关

@EnableAsync 配置类中，通过此注解开启对异步任务的支持，叙事性AsyncConfigurer接口（类上）

@Async 在实际执行的bean方法使用该注解来申明其是一个异步任务（方法上或类上所有的方法都将异步，需要@EnableAsync开启异步任务）

四、Spring Bean生命周期

9.定时任务相关

`@EnableScheduling` 在配置类上使用，开启计划任务的支持（类上）

`@Scheduled` 来申明这是一个任务，包括cron,fixDelay,fixRate等类型（方法上，需先开启计划任务的支持）

四、Spring Bean生命周期

10.@Enable*注解，这些注解主要用来开启对xxx的支持。

@EnableAspectJAutoProxy 开启对AspectJ自动代理的支持

@EnableAsync 开启异步方法的支持

@EnableScheduling 开启计划任务的支持

@EnableWebMvc 开启Web MVC的配置支持

@EnableConfigurationProperties 开启对@ConfigurationProperties注解配置Bean的支持

@EnableJpaRepositories 开启对SpringData JPA Repository的支持

@EnableTransactionManagement 开启注解式事务的支持

@EnableTransactionManagement 开启注解式事务的支持

@EnableCaching 开启注解式的缓存支持

四、Spring Bean生命周期

11.测试相关注解

@RunWith 运行器，Spring中通常用于对JUnit的支持

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration 用来加载配置ApplicationContext，其中classes属性用来加载配置类

@ContextConfiguration(classes={TestConfig.class})

四、Spring Bean生命周期

12.SpringMVC相关注解

`@EnableWebMvc` 在配置类中开启Web MVC的配置支持，如一些ViewResolver或者MessageConverter等，若无此句，重写WebMvcConfigurerAdapter方法（用于对SpringMVC的配置）。

`@Controller` 声明该类为SpringMVC中的Controller

`@RequestMapping` 用于映射Web请求，包括访问路径和参数（类或方法上）

`@ResponseBody` 支持将返回值放在response内，而不是一个页面，通常用户返回json数据（返回值旁或方法上）

`@RequestBody` 允许request的参数在request体中，而不是在直接连接在地址后面。（放在参数前）

`@PathVariable` 用于接收路径参数，比如`@RequestMapping("/hello/{name}")`申明的路径。

`@RestController` 该注解为一个组合注解，相当于`@Controller`和`@ResponseBody`的组合。

`@ControllerAdvice` 通过该注解，我们可以将对于控制器的全局配置放置在同一个位置。

`@ExceptionHandler` 用于全局处理控制器里的异常

`@InitBinder` 用来设置WebDataBinder，WebDataBinder用来自动绑定前台请求参数到Model中。

`@ModelAttribute` 本来的作用是绑定键值对到Model里，在`@ControllerAdvice`中是让全局的`@RequestMapping`都能获得在此处设置的键值对。

四、Spring Bean生命周期

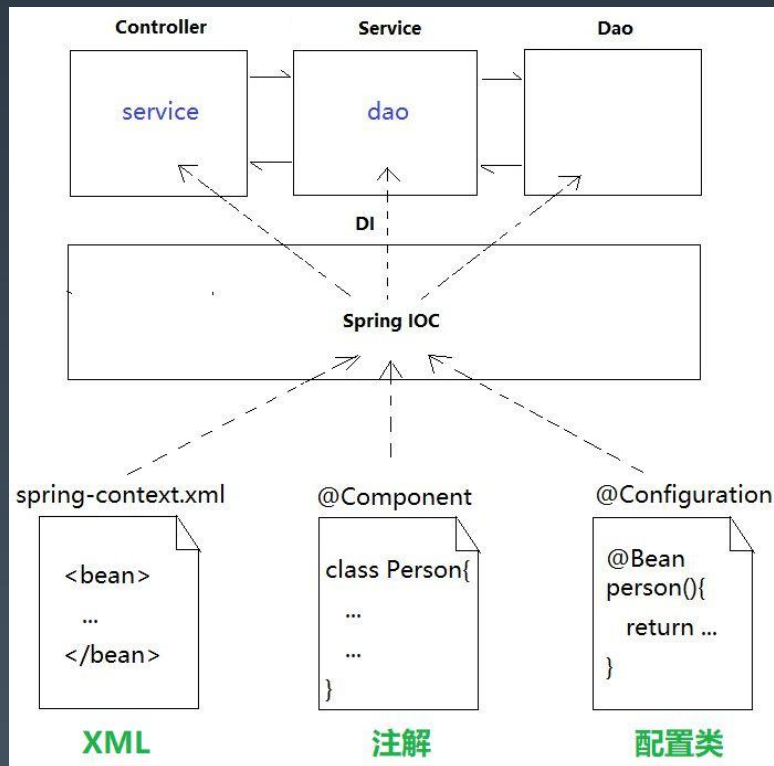
配置的发展方向：

XML--全局

注解--类

配置类--方法

Spring 4以上的新特性，走向Spring Boot



四、Spring Bean生命周期

纯XML配置开发：没有注解，全部<bean>标签，但也可以配置自动装配

注解开发不能单独存在，需要开启扫描。自动装配一般用@Autowired

XML+注解：XML+<context:component-scan>+@Component

JavaConfig+注解：@Configuration+@ComponentScan+@Component

JavaConfig方式：@Configuration+@Bean

3种编程风格：XML、注解、JavaConfig

2种注入方式：setter方法、构造方法

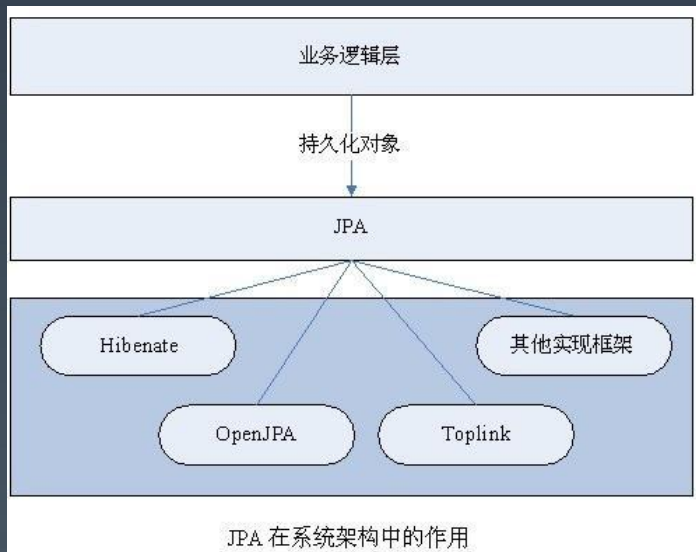
4种装配模式：byType、byName、constructor、no(ref)

五、Spring ORM持久化与事务

ORM (Object-Relational Mapping) 表示对象关系映射。

Hibernate是一个开源的对象关系映射框架，它对JDBC进行了非常轻量级的对象封装，它将POJO与数据库表建立映射关系，是一个全自动的orm框架，hibernate可以自动生成SQL语句，自动执行，使得Java程序员可以使用面向对象的思维来操纵数据库。

JPA的全称是Java Persistence API，
即Java 持久化API，是一套基于ORM的规范，
内部是由一系列的接口和抽象类构成。
JPA通过JDK 5.0注解描述对象-关系表映射关系，
并将运行期的实体对象持久化到数据库中。



五、Spring ORM持久化与事务

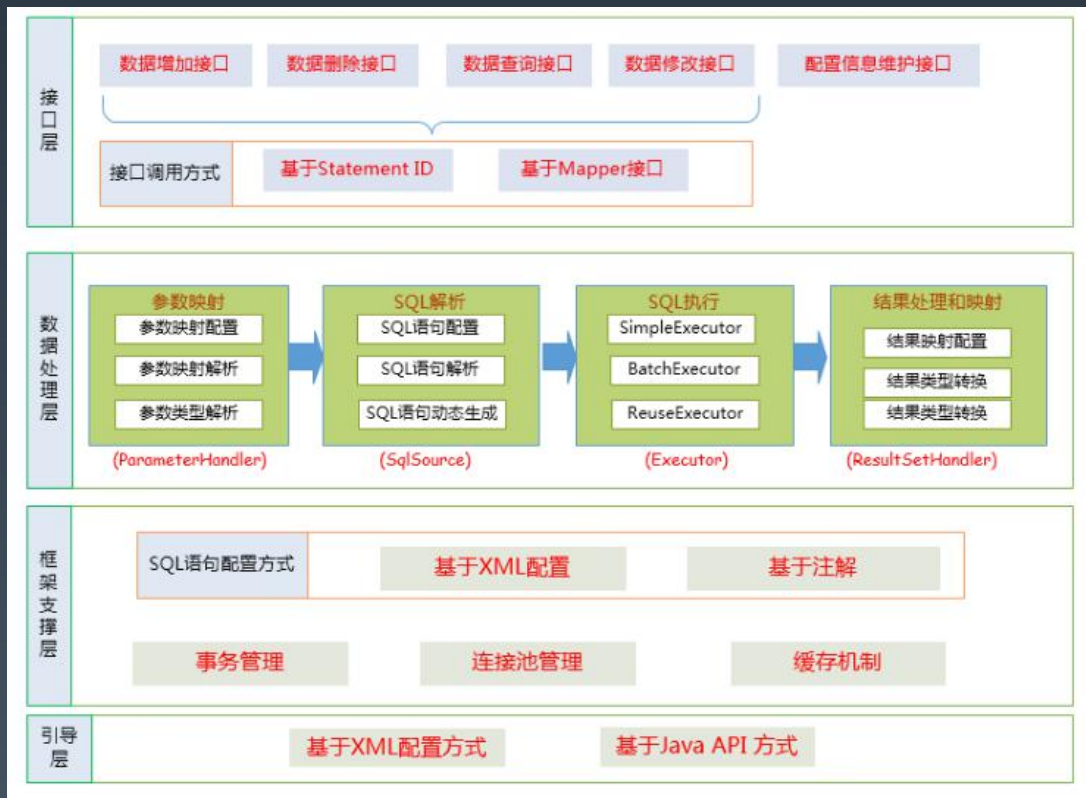
Hibernate/JPA 用法展示

User实体类

注解

五、Spring ORM持久化与事务

MyBatis 是一款优秀的持久层框架，它支持定制化 SQL、存储过程以及高级映射。MyBatis 避免了几乎所有的 JDBC 代码和手动设置参数以及获取结果集。MyBatis 可以使用简单的 XML 或注解来配置和映射原生信息，将接口和 Java 的 POJOs(Plain Old Java Objects,普通的 Java对象)映射成数据库中的记录。



五、Spring ORM持久化与事务

Mybatis 用法展示

User表

XML/Mapper

五、Spring ORM持久化与事务

MyBatis 与 Hibernate 的区别与联系？

Mybatis优点：原生SQL（XML语法），直观，对DBA友好

Hibernate优点：简单场景不用写SQL（HQL、Criteria、SQL）

Mybatis缺点：繁琐，可以用MyBatis-generator、MyBatis-Plus之类的插件

Hibernate缺点：对DBA不友好

考虑为什么大公司都用MyBatis？

五、Spring ORM持久化与事务

怎么在Spring中解决事务问题？

事务的本质是什么？

事务的特性

- 原子性 (Atomicity)：事务是一个原子操作，由一系列动作组成。事务的原子性确保动作要么全部完成，要么完全不起作用。
- 一致性 (Consistency)：一旦事务完成（不管成功还是失败），系统必须确保它所建模的业务处于一致的状态，而不会是部分完成部分失败。在现实中的数据不应该被破坏。
- 隔离性 (Isolation)：可能有许多事务会同时处理相同的数据，因此每个事务都应该与其他事务隔离开来，防止数据损坏。
- 持久性 (Durability)：一旦事务完成，无论发生什么系统错误，它的结果都不应该受到影响，这样就能从任何系统崩溃中恢复过来。通常情况下，事务的结果被写到持久化存储器中。

JDBC层，数据库访问层，怎么操作事务？ 程式化事务管理

Spring怎么做到无侵入实现事务？ 声明式事务管理：事务管理器+AOP

五、Spring ORM持久化与事务

事务的传播性一般用在事务嵌套的场景，比如一个事务方法里面调用了另外一个事务方法，那么两个方法是各自作为独立的方法提交还是内层的事务合并到外层的事务一起提交，这就是需要事务传播机制的配置来确定怎么样执行。

常用的事务传播机制如下：

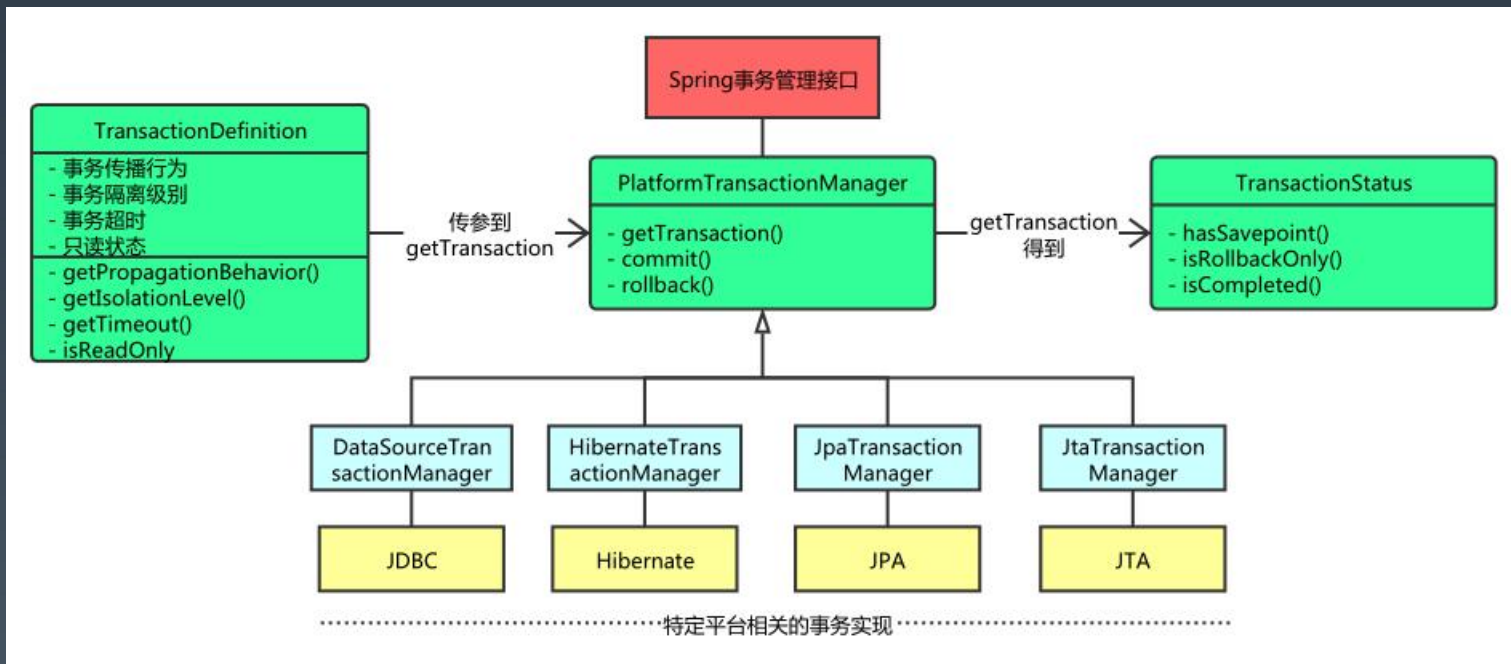
- PROPAGATION_REQUIRED：Spring默认的传播机制，能满足绝大部分业务需求，如果外层有事务，则当前事务加入到外层事务，一块提交，一块回滚。如果外层没有事务，新建一个事务执行
- PROPAGATION_REQUIRES_NEW：该事务传播机制是每次都会新开启一个事务，同时把外层事务挂起，当当前事务执行完毕，恢复上层事务的执行。如果外层没有事务，执行当前新开启的事务即可
- PROPAGATION_SUPPORT：如果外层有事务，则加入外层事务，如果外层没有事务，则直接使用非事务方式执行。完全依赖外层的事务
- PROPAGATION_NOT_SUPPORT：该传播机制不支持事务，如果外层存在事务则挂起，执行完当前代码，则恢复外层事务，无论是否异常都不会回滚当前的代码
- PROPAGATION_NEVER：该传播机制不支持外层事务，即如果外层有事务就抛出异常
- PROPAGATION_MANDATORY：与NEVER相反，如果外层没有事务，则抛出异常
- PROPAGATION_NESTED：该传播机制的特点是可以保存状态保存点，当前事务回滚到某一个点，从而避免所有的嵌套事务都回滚，即各自回滚各自的，如果子事务没有把异常吃掉，基本还是会引起全部回滚的。

五、Spring ORM持久化与事务

隔离级别	含义
ISOLATION_DEFAULT	使用后端数据库默认的隔离级别
ISOLATION_READ_UNCOMMITTED	允许读取尚未提交的更改。可能导致脏读、幻读或不可重复读。
ISOLATION_READ_COMMITTED	(Oracle 默认级别) 允许从已经提交的并发事务读取。可防止脏读，但幻读和不可重复读仍可能会发生。
ISOLATION_REPEATABLE_READ	(MYSQL 默认级别) 对相同字段的多次读取的结果是一致的，除非数据被当前事务本身改变。可防止脏读和不可重复读，但幻读仍可能发生。
ISOLATION_SERIALIZABLE	完全服从ACID的隔离级别，确保不发生脏读、不可重复读和幻影读。这在所有隔离级别中也是最慢的，因为它通常是通过完全锁定当前事务所涉及的数据表来完成的。

五、Spring ORM持久化与事务

事务相关原理实现



五、Spring ORM持久化与事务

Spring声明式事务配置参考

事务的传播性:

@Transactional(propagation=Propagation.REQUIRED)

事务的隔离级别:

@Transactional(isolation = Isolation.READ_UNCOMMITTED)

读取未提交数据(会出现脏读, 不可重复读) 基本不使用

只读:

@Transactional(readOnly=true)

该属性用于设置当前事务是否为只读事务, 设置为true表示只读, false则表示可读写, 默认值为false。

事务的超时性:

@Transactional(timeout=30)

回滚:

指定单一异常类: @Transactional(rollbackFor=RuntimeException.class)

指定多个异常类: @Transactional(rollbackFor={RuntimeException.class, Exception.class})

五、Spring ORM持久化与事务

Spring的JdbcTemplate介绍

一个非常简单的仿ORM工具类

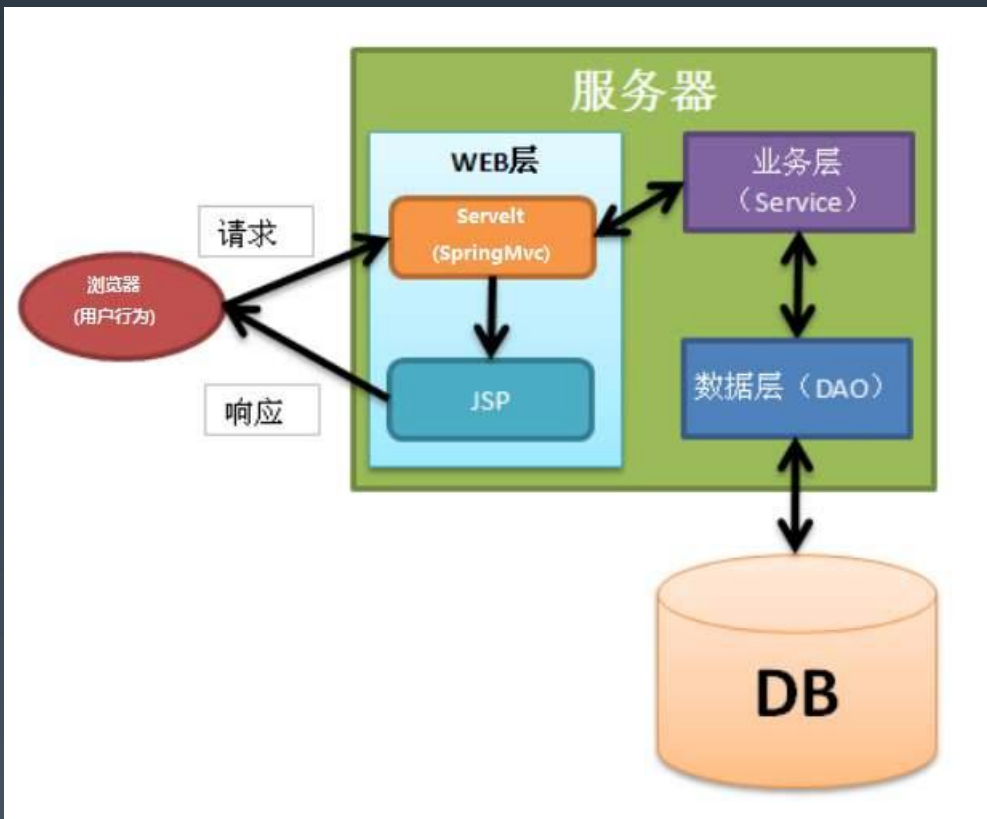
Spring里有各种不同的XXXTemplate

举例一个事务不起作用的坑

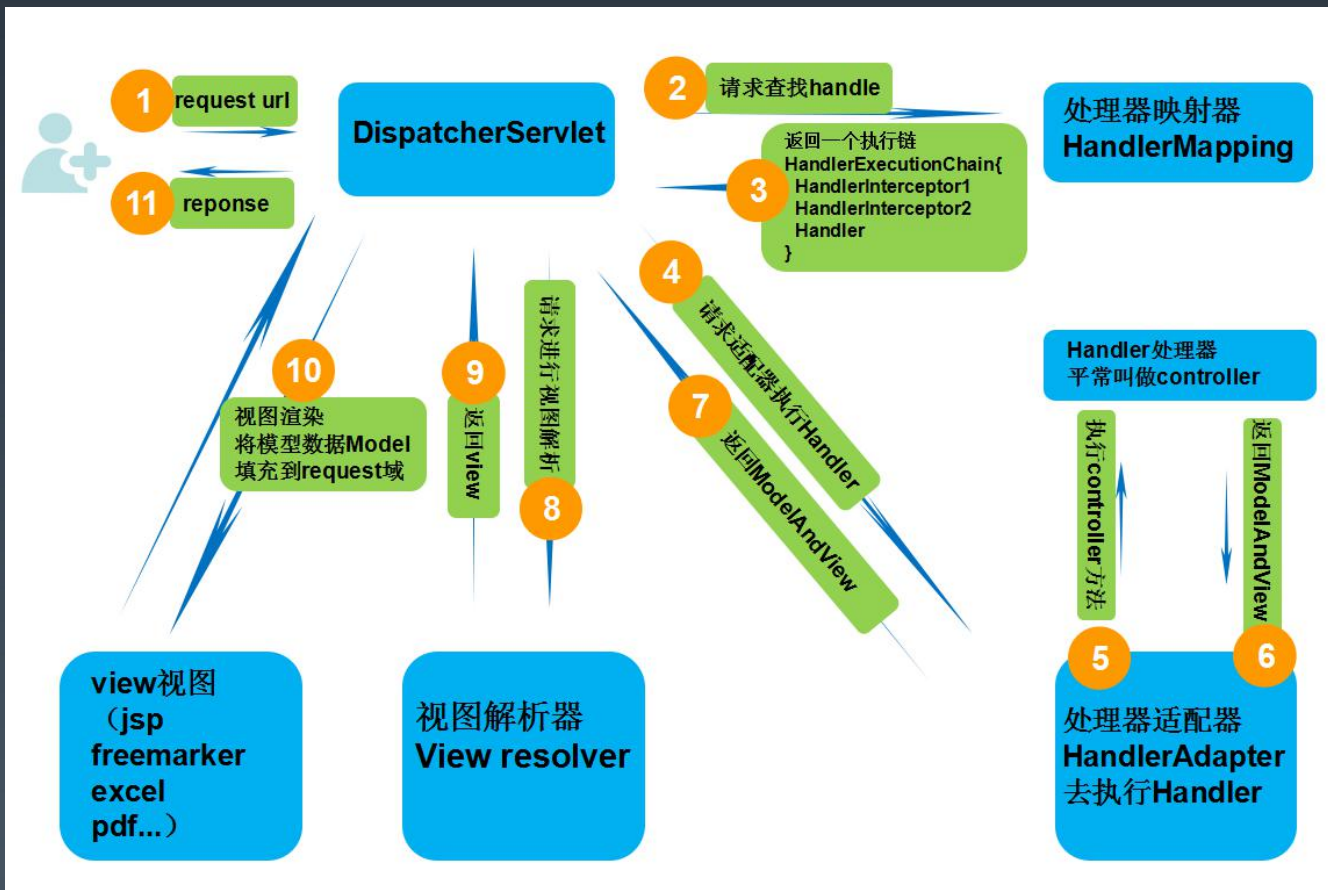
六、Spring MVC实现REST

什么是MVC

为什么有Spring MVC (via Spring Web)



六、Spring MVC实现REST



六、Spring MVC实现REST

第一步：用户发起请求到前端控制器 (DispatcherServlet)

第二步：前端控制器请求处理器映射器 (HandlerMapping) 去查找处理器 (Handle) ,通过xml配置或者注解查找

第三步：找到以后处理器映射器 (HandlerMapping) 像前端控制器返回执行链 (HandlerExecutionChain)

第四步：前端控制器 (DispatcherServlet) 调用处理器适配器 (HandlerAdapter) 去执行处理器 (Handler)

第五步：处理器适配器去执行Handler

第六步：Handler执行完给处理器适配器返回ModelAndView

第七步：处理器适配器向前端控制器返回ModelAndView

第八步：前端控制器请求视图解析器 (ViewResolver) 去进行视图解析

第九步：视图解析器像前端控制器返回View

第十步：前端控制器对视图进行渲染

第十一步：前端控制器向用户响应结果

六、Spring MVC实现REST

从Spring MVC走向前后端分离REST

前端：三大框架

后端：SpringMVC->RestController, SwaggerUI

六、Spring MVC实现REST



六、Spring MVC实现REST

Spring MVC 与 REST 示例演示

<https://gitee.com/nicefish/nicefish-backend>

七、Spring Data/Messaging

Spring Data: Spring 的一个子项目。用于简化数据库访问，支持NoSQL和关系数据库存储。其主要目标是使数据库的访问变得方便快捷。

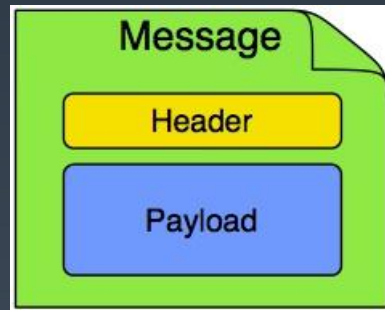
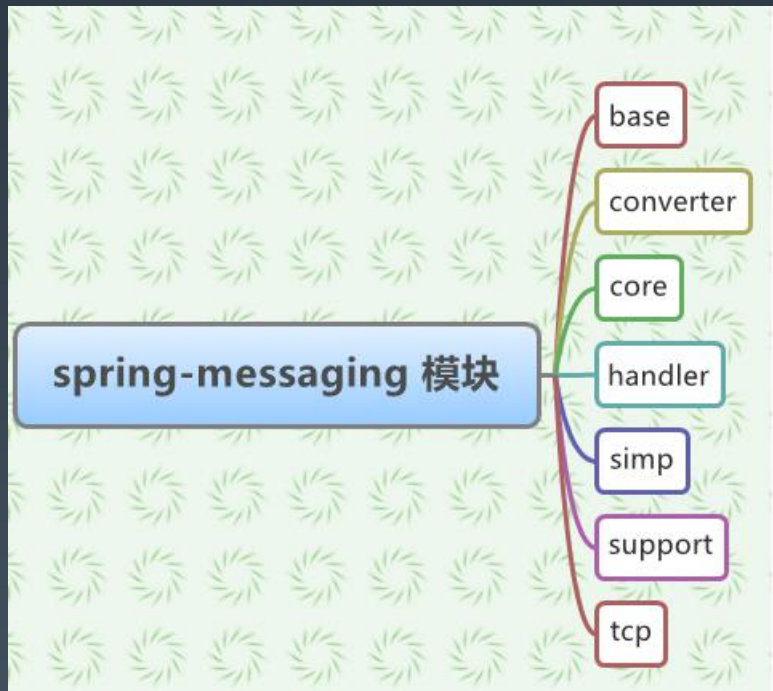
Spring Data 项目所支持NoSQL存储（通过统一抽象，像操作数据库一样）：

- - MongoDB（文档数据库）
- - Neo4j（图形数据库）
- - Redis（键/值存储）
- - Hbase（列族数据库）

Spring Data 项目所支持的关系数据存储（前面ORM部分）：

- - JDBC
- - JPA

七、Spring Data/Messaging/JMS



七、Spring Data/Messaging

简化消息操作，实现面向对象一样的方式去发送和接收消息

XXXTemplate: 如JMSTemplate, KafkaTemplate, AmqpTemplate

Converter: 转换Pojo与Message对象

七、Spring Data/Messaging

实例演示:

ActiveMQ/JMS消息处理

Spring基础知识总结

1. Java 8特性与常用工具
2. Spring框架与 AOP/IOC
3. Spring Bean生命周期与核心源码
4. Spring ORM持久化与事务
5. Spring MVC实现REST
6. Spring Data/Messaging



还有更多组件, Spring Batch, Spring Task, Spring Integration, Spring WS, Spring Security, , ,
越来越复杂, 慢慢变成了Spring自己曾经讨厌的样子, 怎么能更好的适应软件开发的实际需要?

我们进入下一部分, 王朝复兴--走向Spring Boot之路

THANKS! |  极客大学