# Java面试-框架篇Spring

## 一、什么是Spring IOC容器

Spring 框架的核心是 Spring 容器。容器创建对象，将它们装配在一起，配置它们并管理它们的完整生命周期。Spring 容器使用依赖注入来管理组成应用程序的组件。容器通过读取提供的配置元数据来接收对象进行实例化，配置和组装的指令。该元数据可以通过 XML，Java 注解或 Java 代码提供。

**依赖注入：**

在依赖注入中，您不必创建对象，但必须描述如何创建它们。您不是直接在代码 中将组件和服务连接在一起，而是描述配置文件中哪些组件需要哪些服务。由 IoC 容器将它们装配在一起。

**注入方式：**

构造函数注入 、setter 注入 、接口注入， 在 Spring Framework 中，仅使用构造函数和 setter 注入。

Ioc实现的主要机制就是工厂模式+反射。

## 二、BeanFactory和 ApplicationContext

beanFactory：

1. 是Spring最底层的接口，提供了简单的容器功能，实例化对象和拿对象的功能。
2. 懒加载bean，启动不加载，调用时再去加载
3. 不支持国家化
4. 不支持依赖的注解

ApplicationContext：

1. 应用上下文继承自BeanFactory接口，他是Spring更高级的一种容器。
2. 启动时初始化所有的Bean，即时加载。
3. 支持国际化
4. 支持依赖的注解

## 聊一聊Spring Bean

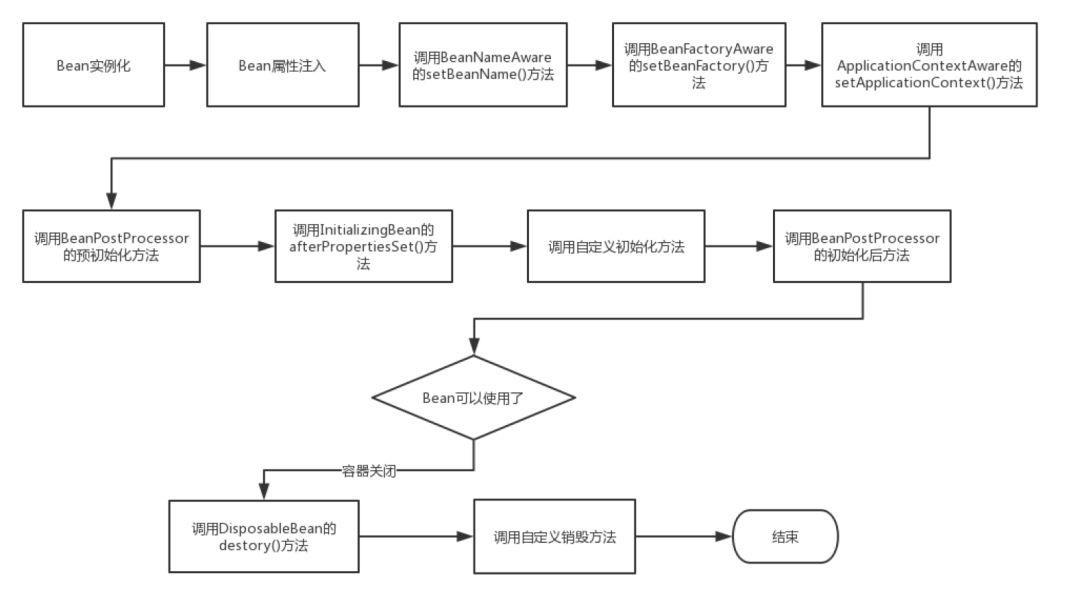
它们是构成用户应用程序主干的对象。 · Bean 由 Spring IoC 容器管理。 · 它们由 Spring IoC 容器实例化，配置，装配和管理。 · Bean 是基于用户提供给容器的配置元数据创建。

**Bean的5中Scope：**

1. Singleton 每个Spring Ioc容器仅用一个bean
2. Propotype：每次请求都会产生新的Bean
3. Request：每次HTTP请求产生一个新的实例
4. Session：类似于标准的HTTP Session作用域

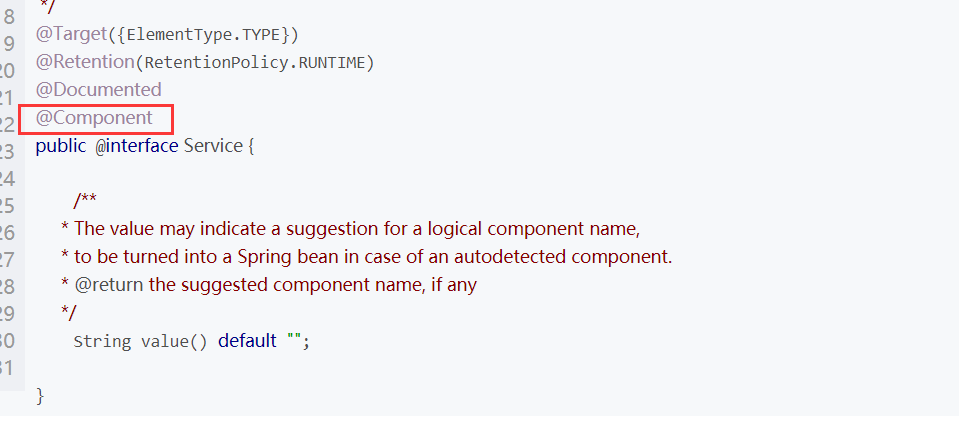
**Spring Bean生命周期：**

1. Spring容器根据配置中的bean定义实例化的bean。
2. Spring使用依赖注入填充所有属性，如bean中所定义的属性。
3. 如果 bean 实现BeanNameAware 接口，则工厂通过传递 bean 的 ID 来调用setBeanName()。
4. 如果bean实现了BeanFactoryAware接口，则工厂通过传递自身的实例对象来调用setBeanFactory()。
5. 如果存在bean关联的任何BeanPostProcessor，则调用preProcessorBeforeInitialization()方法。
6. 如果bean指定了init方法（bean标签下的init-method属性），那么将调用他。
7. 最后如果存在bean关联的任何BeanPostProcessor，则调用PostProcessorAfterInitialization方法。
8. 如果bean实现了DisposableBean接口，当spring容器关闭时，会调用destroy方法。
9. 如果bean指定了destroy方法（bean标签下的destroy-method属性），那么将调用它。



## 四、@Component, @Controller, @Repository,@Service 有何区别？

@Repository：持久层，用于标注数据访问组件，即DAO组件。  
@Service：业务层，用于标注业务逻辑层主键。  
@Controller：控制层，用于标注控制层组件。  
@Component：当你不确定是属于哪一层的时候使用



从@service的源码看，service仍然是使用了@Component注解（@Controller与@Repository与service一样，这里就不贴源码了）。

component即组件，相当于xml配置文件中的bean申明，通过spring的自动扫描机制，在规定的类路径中寻找标注了@Component,@Service,@Repository,@Controller注解的类，并把这些类纳入进容器中进行管理。getBean的默认名称是类名（头字母小写），并且是单例的，如果想自定义，可以使用@Service(“beanName”)@Scope(“prototype”)来改变。

## 五、常用注解释义

@Required 应用于 bean 属性 setter 方法。此注解仅指示必须在配置时使用bean 定义中的显式属性值或使用自动装配填充受影响的 bean 属性。如果尚未填充受影响的 bean 属性，则容器将抛出 BeanInitializationException。

@Autowired 可以更准确地控制应该在何处以及如何进行自动装配。此注解用于在 setter 方法，构造函数，具有任意名称或多个参数的属性或方法上自动装配bean。类型驱动注入。

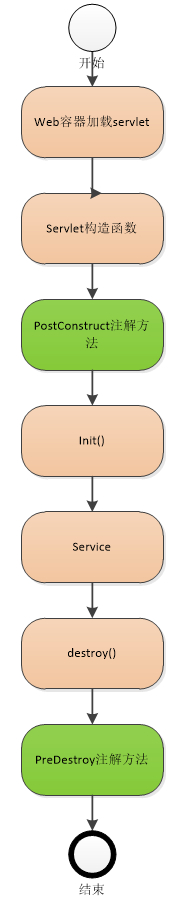
@Qualifier：指定bean的name，当您创建多个相同类型的 bean 并希望仅使用属性装配其中一个 bean 时，您可以使用@Qualifier 注解和 @Autowired 通过指定应该装配哪个确切的 bean来消除歧义。

@RequestMapping 注解用于将特定 HTTP 请求方法映射到将处理相应请求的控制器中的特定类/方法。

@Resource的装配顺序：  
(1)、@Resource后面没有任何内容，默认通过name属性去匹配bean，找不到再按type去匹配  
(2)、指定了name或者type则根据指定的类型去匹配bean  
(3)、指定了name和type则根据指定的name和type去匹配bean，任何一个不匹配都将报错

然后，区分一下@Autowired和@Resource两个注解的区别：  
(1)、@Autowired默认按照byType方式进行bean匹配，@Resource默认按照byName方式进行bean匹配  
(2)、@Autowired是Spring的注解，@Resource是J2EE的注解，这个看一下导入注解的时候这两个注解的包名就一清二楚了  
Spring属于第三方的，J2EE是Java自己的东西，因此，建议使用@Resource注解，以减少代码和Spring之间的耦合。

@Configuration把一个类作为一个IoC容器，它的某个方法头上如果注册了@Bean，就会作为这个Spring容器中的Bean。  
@Scope注解 作用域  
@Lazy(true) 表示延迟初始化  
@Scope用于指定scope作用域的（用在类上）  
@PostConstruct用于指定初始化方法（用在方法上）  
@PreDestory用于指定销毁方法（用在方法上）  
@DependsOn：定义Bean初始化及销毁时的顺序  
@Primary：自动装配时当出现多个Bean候选者时，被注解为@Primary的Bean将作为首选者，否则将抛出异常  
@Async异步方法调用



## 六、什么是AOP

AOP(Aspect-Oriented Programming), 即 面向切面编程, 它与OOP( Object-Oriented Programming, 面向对象编程) 相辅相成, 提供了与OOP 不同的抽象软件结构的视角. 在 OOP 中, 我们以类(class)作为我们的基本单元, 而 AOP 中的基本单元是 Aspect(切面)

Aop应用举例：日志、事务、权限、缓存。

AOP相关概念：

(1)横切关注点：对哪些方法进行拦截，拦截后怎么处理，这些关注点称之为横切关注点  
(2)Aspect(切面):通常是一个类，里面可以定义切入点和通知  
(3)JointPoint(连接点):程序执行过程中明确的点，一般是方法的调用。被拦截到的点，因为Spring只支持方法类型的连接点，所以在Spring中连接点指的就是被拦截到的方法，实际上连接点还可以是字段或者构造器  
(4)Advice(通知):AOP在特定的切入点上执行的增强处理，有before(前置),after(后置),afterReturning(最终),afterThrowing(异常),around(环绕)  
(5)Pointcut(切入点):就是带有通知的连接点，在程序中主要体现为书写切入点表达式  
(6)weave(织入)：将切面应用到目标对象并导致代理对象创建的过程  
(7)introduction(引入)：在不修改代码的前提下，引入可以在运行期为类动态地添加一些方法或字段  
(8)AOP代理(AOP Proxy)：AOP框架创建的对象，代理就是目标对象的加强。Spring中的AOP代理可以使JDK动态代理，也可以是CGLIB代理，前者基于接口，后者基于子类  
(9)目标对象（Target Object）: 包含连接点的对象。也被称作被通知或被代理对象。POJO

通知类型介绍：

1. before：在目标方法被调用之前做增强处理。
2. afterReturning：在目标方法正常完成后做增强，除了指定切入点表达式后，还可以指定returning的返回值值形参名，代表目标方法的返回值。
3. AfterThrowing:主要用来处理程序中未处理的异常,@AfterThrowing除了指定切入点表达式后，还可以指定一个throwing的返回值形参名,可以通过该形参名来访问目标方法中所抛出的异常对象
4. After:在目标方法完成之后做增强，无论目标方法时候成功完成。@After可以指定一个切入点表达式
5. Around:环绕通知,在目标方法完成前后做增强处理,环绕通知是最重要的通知类型,像事务,日志等都是环绕通知,注意编程中核心是一个ProceedingJoinPoint

什么是静态代理：

由程序员创建或工具生成代理类的源码，再编译代理类。所谓静态也就是在**程序运行前就已经存在代理类**的字节码文件，代理类和委托类的关系在运行前就确定了。

什么是动态代理

1.代理对象,不需要实现接口

2.代理对象的生成,是利用JDK的API,动态的在内存中构建代理对象(需要我们指定创建代理对象/目标对象实现的接口的类型)

3.动态代理也叫做:JDK代理,接口代理

JDK动态代理

1)原理：是根据类加载器和接口创建代理类（此代理类是接口的实现类，所以必须使用接口 面向接口生成代理，位于java.lang.reflect包下）

2)实现方式：

A、通过实现InvocationHandler接口创建自己的调用处理器 IvocationHandler handler = new InvocationHandlerImpl(…);

B、通过为Proxy类指定ClassLoader对象和一组interface创建动态代理类Class clazz = Proxy.getProxyClass(classLoader,new Class[]{…});

C、通过反射机制获取动态代理类的构造函数，其参数类型是调用处理器接口类型Constructor constructor = clazz.getConstructor(new Class[]{InvocationHandler.class});

D、通过构造函数创建代理类实例，此时需将调用处理器对象作为参数被传入Interface Proxy = (Interface)constructor.newInstance(new Object[] (handler));

缺点：jdk动态代理，必须是面向接口，目标业务类必须实现接口

CGLIB动态代理

**原理：**利用asm开源包，对代理对象类的class文件加载进来，通过修改其字节码生成子类来处理。

**什么是CGLIB动态代理**

使用cglib[Code Generation Library]实现动态代理，并不要求委托类必须实现接口，底层采用asm字节码生成框架生成代理类的字节码

CGLIB动态代理与JDK动态区别

java动态代理是利用反射机制生成一个实现代理接口的匿名类，在调用具体方法前调用InvokeHandler来处理。而cglib动态代理是利用asm开源包，对代理对象类的class文件加载进来，通过修改其字节码生成子类来处理。

Spring中：

1、如果目标对象实现了接口，默认情况下会采用JDK的动态代理实现AOP

2、如果目标对象实现了接口，可以强制使用CGLIB实现AOP

3、如果目标对象没有实现了接口，必须采用CGLIB库，spring会自动在JDK动态代理和CGLIB之间转换

## 介绍一下Spring事务

事务基本要素ACID：原子性、一致性、隔离性、持久性。

JDBC事务一般步骤：

1、获取连接 Connection con = DriverManager.getConnection()

2、开启事务con.setAutoCommit(true/false);

3、执行CRUD

4、提交事务/回滚事务 con.commit() / con.rollback();

5、关闭连接 conn.close()；

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 隔离级别 | 隔离级别的值 | 脏读 | 不可重复读 | 幻读 |
| 读未提交Read-Uncommitted | 0 | 是 | 是 | 是 |
| 不可重复读（read-committed） | 1 | 否 | 是 | 否 |
| 可重复读（repeatable-read） | 2 | 否 | 否 | 是 |
| 串行化（serializable） | 3 | 否 | 否 | 否 |

**脏读**：一事务对数据进行了增删改，但未提交，另一事务可以读取到未提交的数据。如果第一个事务这时候回滚了，那么第二个事务就读到了脏数据。

**不可重复读**：一个事务中发生了两次读操作，第一次读操作和第二次操作之间，另外一个事务对数据进行了修改，这时候两次读取的数据是不一致的。

**幻读：**第一个事务对一定范围的数据进行批量修改，第二个事务在这个范围增加一条数据，这时候第一个事务就会丢失对新增数据的修改。

**小结：**不可重复读的和幻读很容易混淆，不可重复读侧重于修改，幻读侧重于新增或删除。解决不可重复读的问题只需锁住满足条件的行，解决幻读需要锁表

**事务的7种传播级别：**

1） PROPAGATION\_REQUIRED ，默认的spring事务传播级别，如果上下文中已经存在事务，那么就加入到事务中执行，如果当前上下文中不存在事务，则新建事务执行。

2）PROPAGATION\_SUPPORTS ，如果上下文存在事务，则支持事务加入事务，如果没有事务，则使用非事务的方式执行。

3）PROPAGATION\_MANDATORY ， 该级别的事务要求上下文中必须要存在事务，否则就会抛出异常！配置该方式的传播级别是有效的控制上下文调用代码遗漏添加事务控制的保证手段。比如一段代码不能单独被调用执行，但是一旦被调用，就必须有事务包含的情况，就可以使用这个传播级别。

4）PROPAGATION\_REQUIRES\_NEW ，从字面即可知道，new，每次都要一个新事务，该传播级别的特点是，每次都会新建一个事务，并且同时将上下文中的事务挂起，执行当前新建事务完成以后，上下文事务恢复再执行。

这是一个很有用的传播级别，举一个应用场景：现在有一个发送100个红包的操作，在发送之前，要做一些系统的初始化、验证、数据记录操作，然后发送100封红包，然后再记录发送日志，发送日志要求100%的准确，如果日志不准确，那么整个父事务逻辑需要回滚。  
怎么处理整个业务需求呢？就是通过这个PROPAGATION\_REQUIRES\_NEW 级别的事务传播控制就可以完成。发送红包的子事务不会直接影响到父事务的提交和回滚。

5）PROPAGATION\_NOT\_SUPPORTED ，这个也可以从字面得知，not supported ，不支持，当前级别的特点就是上下文中存在事务，则挂起事务，执行当前逻辑，结束后恢复上下文的事务。

这个级别有什么好处？可以帮助你将事务极可能的缩小。我们知道一个事务越大，它存在的风险也就越多。所以在处理事务的过程中，要保证尽可能的缩小范围。比如一段代码，是每次逻辑操作都必须调用的，比如循环1000次的某个非核心业务逻辑操作。这样的代码如果包在事务中，势必造成事务太大，导致出现一些难以考虑周全的异常情况。所以这个事务这个级别的传播级别就派上用场了。用当前级别的事务模板抱起来就可以了。

6）PROPAGATION\_NEVER ，该事务更严格，上面一个事务传播级别只是不支持而已，有事务就挂起，而PROPAGATION\_NEVER传播级别要求上下文中不能存在事务，一旦有事务，就抛出runtime异常，强制停止执行！这个级别上辈子跟事务有仇。

7）PROPAGATION\_NESTED ，字面也可知道，nested，嵌套级别事务。该传播级别特征是，如果上下文中存在事务，则嵌套事务执行，如果不存在事务，则新建事务。

## 八、介绍一下beanDefinition

1、beanDefinition存储了类名、scope、属性、构造参数列表、依赖的bean、是否是单例等信息。

2、BeanDefinition是一个接口，是一个抽象的定义，实际使用的是其实现类，如 ChildBeanDefinition、RootBeanDefinition、GenericBeanDefinition等。

3、BeanDefinition继承了AttributeAccessor，说明它具有处理属性的能力；  
4、BeanDefinition继承了BeanMetadataElement，说明它可以持有Bean元数据元素，作用是可以持有XML文件的一个bean标签对应的Object。

## 九、Spring标签解析

参考文档：<https://blog.csdn.net/xkshihaoren/article/details/87898017>

## 十、循环依赖如何解决的

第一个步骤的getSingleton()方法的作用是尝试从缓存中获取目标对象，如果没有获取到，则尝试获取半成品的目标对象；如果第一个步骤没有获取到目标对象的实例，那么就进入第二个步骤

第二个步骤的getSingleton()方法的作用是尝试创建目标对象，并且为该对象注入其所依赖的属性。

这里其实就是主干逻辑，我们前面图中已经标明，在整个过程中会调用三次doGetBean()方法

第一次调用的时候会尝试获取A对象实例，此时走的是第一个getSingleton()方法，由于没有已经创建的A对象的成品或半成品，因而这里得到的是null

然后就会调用第二个getSingleton()方法，创建A对象的实例，然后递归的调用doGetBean()方法，尝试获取B对象的实例以注入到A对象中

此时由于Spring容器中也没有B对象的成品或半成品，因而还是会走到第二个getSingleton()方法，在该方法中创建B对象的实例

创建完成之后，尝试获取其所依赖的A的实例作为其属性，因而还是会递归的调用doGetBean()方法

此时需要注意的是，在前面由于已经有了一个半成品的A对象的实例，因而这个时候，再尝试获取A对象的实例的时候，会走第一个getSingleton()方法

在该方法中会得到一个半成品的A对象的实例，然后将该实例返回，并且将其注入到B对象的属性a中，此时B对象实例化完成。

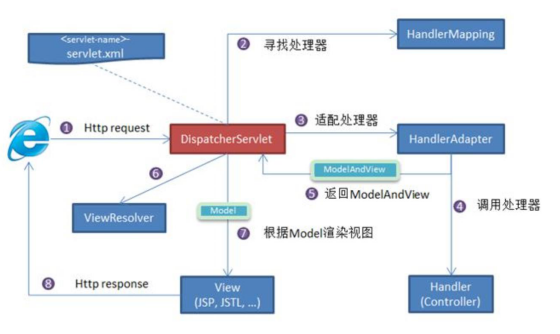
然后，将实例化完成的B对象递归的返回，此时就会将该实例注入到A对象中，这样就得到了一个成品的A对象。

**总结：**

Spring在实例化一个bean的时候，是首先递归的实例化其所依赖的所有bean，直到某个bean没有依赖其他bean，此时就会将该实例返回，然后反递归的将获取到的bean设置为各个上层bean的属性的。

参考文档：<https://zhuanlan.zhihu.com/p/84267654>

## 十一、描述一下 DispatcherServlet 的工作流程（SpringMVC流程）



（1）向服务器发送 HTTP 请求，请求被前端控制器 **DispatcherServlet 捕获**。

（2） DispatcherServlet 根据 -servlet.xml 中的配置对请求的 URL 进行解析，得到请求资源标识符（URI）。然后根据该 URI，**调用 HandlerMapping**获得该 Handler 配置的所有相关的对象（包括 Handler 对象以及 Handler 对象对应的拦截器），最后以**HandlerExecutionChain 对象**的形式返回。  
（3） DispatcherServlet 根据获得的 Handler，**选择一个合适的HandlerAdapter。**（附注：如果成功获得 HandlerAdapter 后，此时将开始执行拦截器的 preHandler(...)方法）。  
（4）提取 Request 中的模型数据，填充 Handler 入参，开始执行 Handler（ Controller)。在填充 Handler 的入参过程中，根据你的配置，Spring 将帮你做一些额外的工作： · HttpMessageConveter：将请求消息（如 Json、xml 等数据）转换成一个对象，将对象转换为指定的响应信息。 · 数据转换：对请求消息进行数据转换。如 String 转换成 Integer、Double 等。 · 数据根式化：对请求消息进行数据格式化。如将字符串转换成格式化数字或格式化日期等。 · 数据验证：验证数据的有效性（长度、格式等），验证结果存储到BindingResult 或 Error 中。   
（5）Handler(Controller)执行完成后，向 DispatcherServlet **返回一个ModelAndView** 对象；   
（6）根据返回的 ModelAndView，选择一个适合的 **ViewResolver**（必须是已经注册到 Spring 容器中的 ViewResolver)返回给 DispatcherServlet。   
（7） ViewResolver 结合 Model 和 View，来渲染视图。  
（8）视图负责将渲染结果返回给客户端。

## Spring框架有哪些好处？

· 轻量：Spring 是轻量的，基本的版本大约 2MB。   
· 控制反转：Spring 通过控制反转实现了松散耦合，对象们给出它们的依赖，而不是创建或查找依赖的对象们。   
· 面向切面的编程(AOP)：Spring 支持面向切面的编程，并且把应用业务逻辑和系统服务分开。 · 容器：Spring 包含并管理应用中对象的生命周期和配置。   
· MVC 框架：Spring 的 WEB 框架是个精心设计的框架，是 Web 框架的一个很好的替代品。   
· 事务管理：Spring 提供一个持续的事务管理接口，可以扩展到上至本地事务下至全局事务（JTA）。   
· 异常处理：Spring 提供方便的 API 把具体技术相关的异常（比如由 JDBC，Hibernate or JDO 抛出的）转化为一致的 unchecked 异常。

# Java面试-框架篇Mybatis

## mybatis简介

（1）Mybatis 是一个半 ORM（对象关系映射）框架，它内部封装了 JDBC，开发时只需要关注 SQL 语句本身，不需要花费精力去处理加载驱动、创建连接、创建statement 等繁杂的过程。程序员直接编写原生态 sql，可以严格控制 sql 执行性能，灵活度高。

（2）MyBatis 可以使用 XML 或注解来配置和映射原生信息，将 POJO 映射成数据库中的记录，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。

（3）通过 xml 文件或注解的方式将要执行的各种 statement 配置起来，并通过java 对象和 statement 中 sql 的动态参数进行映射生成最终执行的 sql 语句，最后由 mybatis 框架执行 sql 并将结果映射为 java 对象并返回。（从执行 sql 到返回 result 的过程）。

## 二、mybatis 优缺点对比

**优点：**

1. 基于sql语句编程，相当灵活。
2. 消除了JDBC的大量冗余代码，不需要手动开关连接。
3. 兼容各种数据库。
4. 能够很好的与Spring结合
5. 提供映射标签，支持对象与数据库的ORM字段关系映射。

**缺点：**

1. SQL语句编写工作量比较大
2. Sql语句依赖数据库，导致移植性比较差。

## Mybatis与hibernate有哪些不同

1. Mybatis需要编写SQL语句。
2. Mybatis可以直接写原生sql，可以控制sql执行性能，灵活度高。
3. Hibernate对象\关系映射能力强，数据无关性好。

## #{} 与 ${} 区别

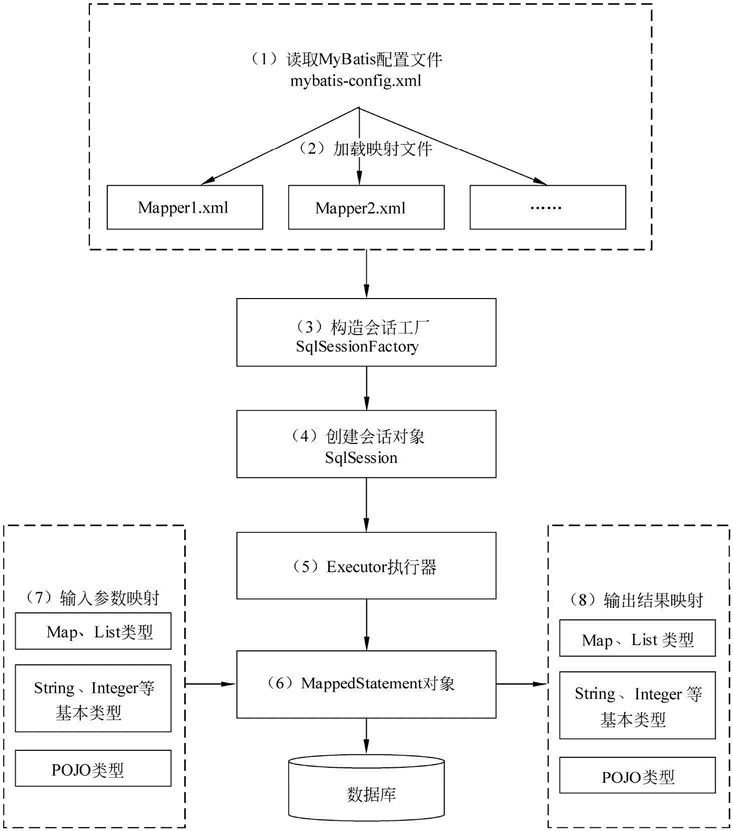
#{}是预编译处理，${}是字符串替换。

Mybatis 在处理#{}时，会将 sql 中的#{}替换为?号，调用 PreparedStatement 的set 方法来赋值；

Mybatis 在处理${}时，就是把${}替换成变量的值。

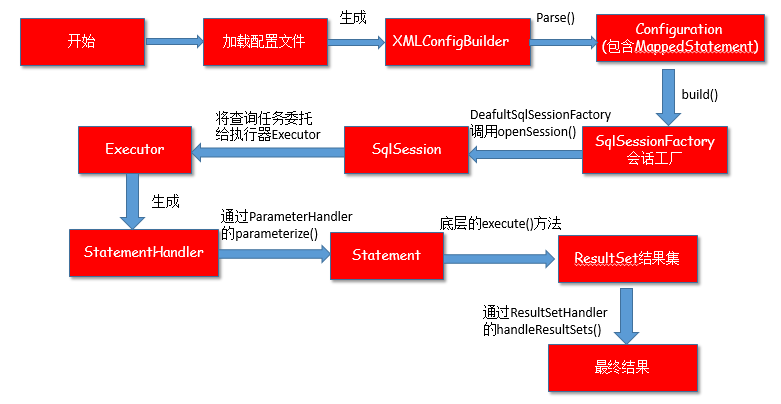
使用#{}可以有效的防止 SQL 注入，提高系统安全性。

## Mybatis执行原理



1）读取 MyBatis 配置文件：mybatis-config.xml 为 MyBatis 的全局配置文件，配置了 MyBatis 的运行环境等信息，例如数据库连接信息。  
2）加载映射文件。映射文件即 SQL 映射文件，该文件中配置了操作数据库的 SQL 语句，需要在 MyBatis 配置文件 mybatis-config.xml 中加载。mybatis-config.xml 文件可以加载多个映射文件，每个文件对应数据库中的一张表。  
3）构造会话工厂：通过 MyBatis 的环境等配置信息构建会话工厂 SqlSessionFactory。

4）创建会话对象：由会话工厂创建 SqlSession 对象，该对象中包含了执行 SQL 语句的所有方法。  
5）Executor 执行器：MyBatis 底层定义了一个 Executor 接口来操作数据库，它将根据 SqlSession 传递的参数动态地生成需要执行的 SQL 语句，同时负责查询缓存的维护。  
6）MappedStatement 对象：在 Executor 接口的执行方法中有一个 MappedStatement 类型的参数，该参数是对映射信息的封装，用于存储要映射的 SQL 语句的 id、参数等信息。  
7）输入参数映射：输入参数类型可以是 Map、List 等集合类型，也可以是基本数据类型和 POJO 类型。输入参数映射过程类似于 JDBC 对 preparedStatement 对象设置参数的过程。  
8）输出结果映射：输出结果类型可以是 Map、 List 等集合类型，也可以是基本数据类型和 POJO 类型。输出结果映射过程类似于 JDBC 对结果集的解析过程。



## JDBC执行流程

1、导入SQL相关的包

1. 注册mysql驱动
2. 创建一个查询
3. 从resultSet中获取数据
4. 关闭连接

## 模糊查询like语句怎么写

1. 在java代码中添加sql通配符
2. 在sql语句中拼接

## 七、Dao 接口的工作原理是什么？

Dao 接口即 Mapper 接口。接口的全限名，就是映射文件中的 namespace 的值；接口的方法名，就是映射文件中 Mapper 的 Statement 的 id 值；接口方法内的参数，就是传递给 sql 的参数。

Mapper 接口是没有实现类的，当调用接口方法时，接口全限名+方法名拼接字符串作为 key 值，可唯一定位一个 MapperStatement。在 Mybatis 中，每一个<select>、<insert>、<update>、<delete>标签，都会被解析为一个MapperStatement 对象。

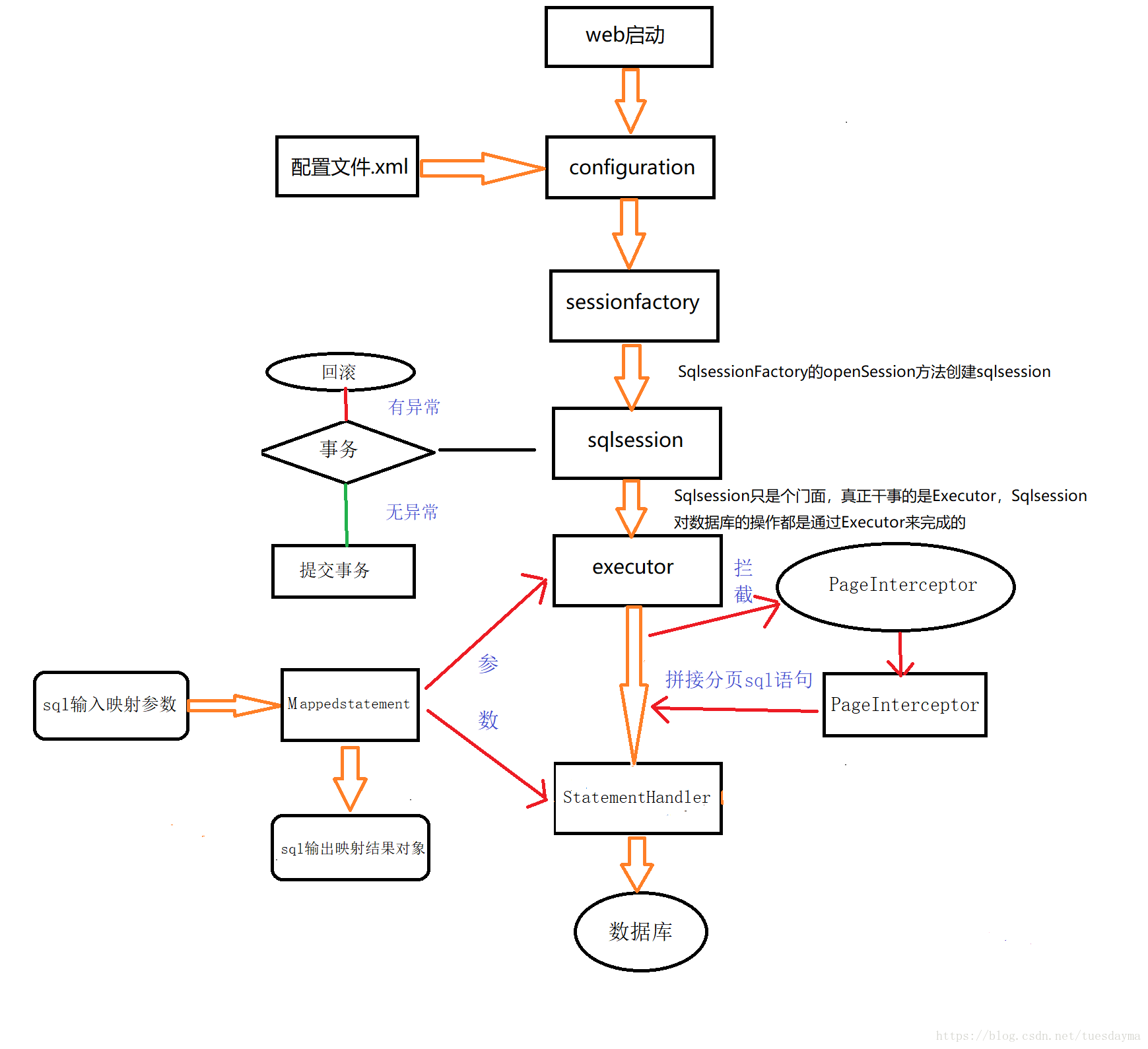
举例：com.mybatis3.mappers.StudentDao.findStudentById，可以唯一找到 namespace 为 com.mybatis3.mappers.StudentDao 下面 id 为findStudentById 的 MapperStatement。

Mapper 接口里的方法，是不能重载的，因为是使用 全限名+方法名 的保存和寻找策略。Mapper 接口的工作原理是 JDK 动态代理，Mybatis 运行时会使用 JDK动态代理为 Mapper 接口生成代理对象 proxy，代理对象会拦截接口方法，转而执行 MapperStatement 所代表的 sql，然后将 sql 执行结果返回。

## Mybatis如何做分页的

Mybatis 使用 RowBounds 对象进行分页，它是针对 ResultSet 结果集执行的内存分页，而非物理分页。可以在 sql 内直接书写带有物理分页的参数来完成物理分页功能，也可以使用分页插件来完成物理分页。

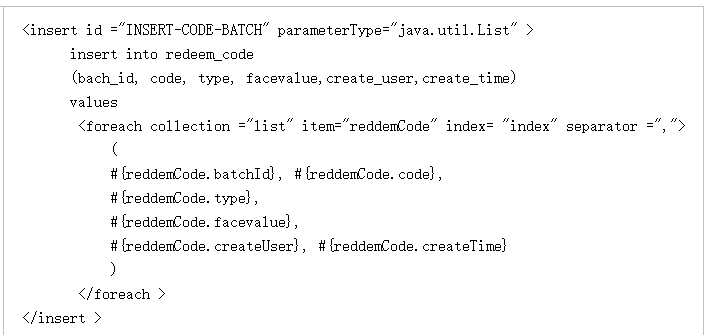
分页插件的基本原理是使用 Mybatis 提供的插件接口，实现自定义插件，在插件的拦截方法内拦截待执行的 sql，然后重写 sql，根 据 dialect 方言，添加对应的物理分页语句和物理分页参数。



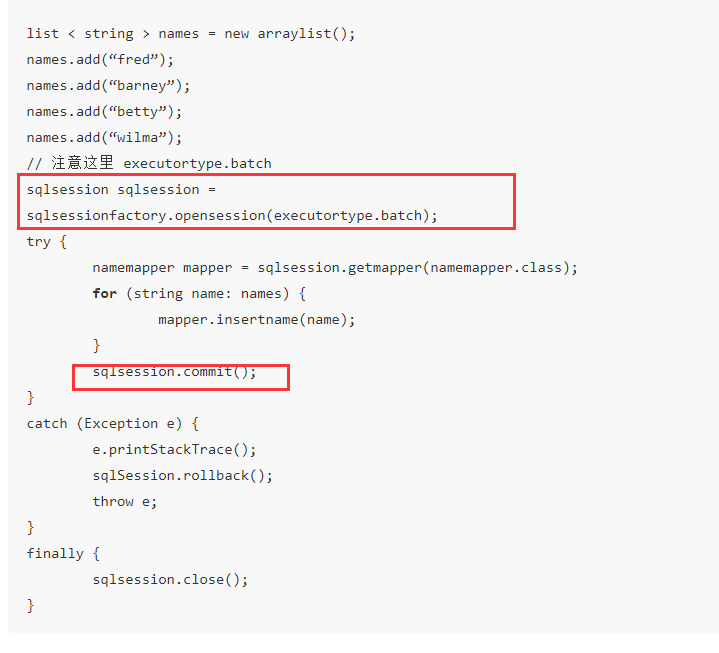
在StatementHandler之前进行拦截，对MappedStatement进行一系列的操作（大致就是拼上分页sql）

## mybatis如何批量插入

1. 使用foreach标签



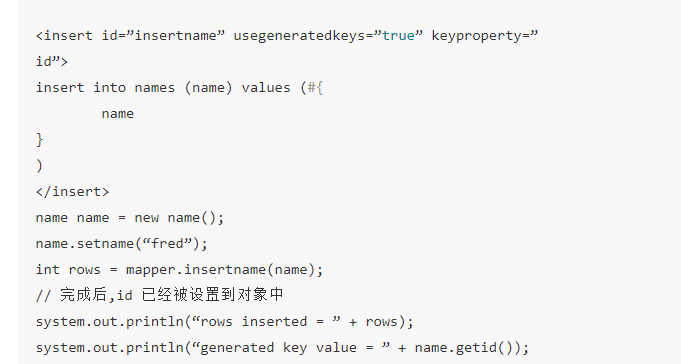
1. Java代码编写



## 如何获取插入数据的主键

1、insert 方法总是返回一个 int 值 ，这个值代表的是插入的行数。

2、如果采用自增长策略，自动生成的键值在 insert 方法执行完后可以被设置到传入的参数对象中。



## Mapper中如何传递参数，有哪几种方式？

第一种：DAO 层的函数

public UserselectUser(String name,String area);

对应的 xml,#{0}代表接收的是 dao 层中的第一个参数，#{1}代表 dao 层中第二参数，更多参数一致往后加即可。

第二种： 使用 @param 注解:

第三种：多个参数封装成 map

## 十一、简述mybatis动态SQL

1、Mybatis 动态 SQL ，可以让我们在 XML 映射文件内，以 XML 标签的形 式编写动态 SQL ，完成逻辑判断和动态拼接 SQL 的功能。

2、Mybatis 提供了 9 种动态 SQL 标签：<if/>、<choose/>、<when/>、<otherwise/>、<trim/>、<when/>、<set/>、<foreach/>、<bind/>。

1. 其执行原理为，使用 OGNL 的表达式，从 SQL 参数对象中计算表达式的值,根据表达式的值动态拼接 SQL ，以此来完成动态 SQL 的功能。

## 十二、Mybatis的延迟加载

Mybatis 仅支持 association 关联对象和 collection 关联集合对象的延迟加载，association 指的就是一对一，collection 指的就是一对多查询。在 Mybatis配置文件中，可以配置是否启用延迟加载 lazyLoadingEnabled=true|false。

它的原理是，使用 CGLIB 创建目标对象的代理对象，当调用目标方法时，进入拦截器方法，比如调用 a.getB().getName()，拦截器 invoke()方法发现 a.getB()是null 值，那么就会单独发送事先保存好的查询关联 B 对象的 sql，把 B 查询上来，然后调用 a.setB(b)，于是 a 的对象 b 属性就有值了，接着完成 a.getB().getName()方法的调用。这就是延迟加载的基本原理。

当然了，不光是 Mybatis，几乎所有的包括 Hibernate，支持延迟加载的原理都是一样的。

## 十三、Mybatis 一二级缓存

1）一级缓存: 基于 PerpetualCache 的 HashMap 本地缓存，其存储**作用域为Session**，当 Session flush 或 close 之后，该 Session 中的所有 Cache 就将清空，默认打开一级缓存。

2）二级缓存与一级缓存其机制相同，默认也是采用 PerpetualCache，HashMap存储，不同在于其存储**作用域为 Mapper(Namespace)，**并且可自定义存储源，如 Ehcache。默认不打开二级缓存，要开启二级缓存，使用二级缓存属性类需要实现 Serializable 序列化接口(可用来保存对象的状态),可在它的映射文件中配置<cache/> ；

3）对于缓存数据更新机制，当某一个作用域(一级缓存 Session/二级缓存Namespaces)的进行了 C/U/D 操作后，默认该作用域下所有 select 中的缓存将被 clear。

## 十四、 简述 Mybatis 的插件运行原理，以及如何编写一个插件

Mybatis 仅可以编写针对 ParameterHandler、ResultSetHandler、StatementHandler、Executor 这 4 种接口的插件，Mybatis 使用 JDK 的动态代理，为需要拦截的接口生成代理对象以实现接口方法拦截功能，每当执行这 4 种接口对象的方法时，就会进入拦截方法，具体就是 InvocationHandler 的 invoke()方法，当然，只会拦截那些你指定需要拦截的方法。

编写插件：实现 Mybatis 的 Interceptor 接口并复写 intercept()方法，然后在给插件编写注解，指定要拦截哪一个接口的哪些方法即可，记住，别忘了在配置文件中配置你编写的插件。