1.Spring 两大核心

控制反转（IOC）或依赖注入（DI），面向切面编程（AOP）

（1）面向切面编程（AOP）

AOP将哪些与业务无关的，但对多个对象产生一个像的公共行为和逻辑，抽取并封装为一个可重用的模块。这个模块被命名为‘切面’（Aspect）,减少系统中的重复代码，降低了模块间的耦合，同时提高了系统的可维护性。可用于权限认证、日志，事务处理。

使用场景：

Transactions 事务

Cache 缓存

Synchronization 同步

Authentioncation 权限

Context passing 内容传递

Error handing 错误处理

Lazy loading 懒加载

logging　　 日志

Performance optimazation 性能优化

profilling and monitoring 纪录跟踪优化

Persistence 持久化

Resource pooling 　　　　 资源池

Debugging 调试

AOP实现的关键在于 代理模式，AOP代理主要分为静态代理和动态代理。静态代理的代表为AspectJ；动态代理则以Spring AOP为代表。

（1）AspectJ是静态代理的增强，所谓静态代理，就是AOP框架会在编译阶段生成AOP代理类，因此也称为编译时增强，他会在编译阶段将AspectJ(切面)织入到Java字节码中，运行的时候就是增强之后的AOP对象。

（2）Spring AOP使用的动态代理，所谓的动态代理就是说AOP框架不会去修改字节码，而是每次运行时在内存中临时为方法生成一个AOP对象，这个AOP对象包含了目标对象的全部方法，并且在特定的切点做了增强处理，并回调原对象的方法。

Spring AOP中的动态代理主要有两种方式，JDK动态代理和CGLIB动态代理：

①JDK动态代理只提供接口的代理，不支持类的代理。核心InvocationHandler接口和Proxy类，InvocationHandler 通过invoke()方法反射来调用目标类中的代码，动态地将横切逻辑和业务编织在一起；接着，Proxy利用 InvocationHandler动态创建一个符合某一接口的的实例, 生成目标类的代理对象。

②如果代理类没有实现 InvocationHandler 接口，那么Spring AOP会选择使用CGLIB来动态代理目标类。CGLIB（Code Generation Library），是一个代码生成的类库，可以在运行时动态的生成指定类的一个子类对象，并覆盖其中特定方法并添加增强代码，从而实现AOP。CGLIB是通过继承的方式做的动态代理，因此如果某个类被标记为final，那么它是无法使用CGLIB做动态代理的。

（3）静态代理与动态代理区别在于生成AOP代理对象的时机不同，相对来说AspectJ的静态代理方式具有更好的性能，但是AspectJ需要特定的编译器进行处理，而Spring AOP则无需特定的编译器处理。

InvocationHandler 的 invoke(Object proxy,Method method,Object[] args)：proxy是最终生成的代理实例; method 是被代理目标实例的某个具体方法; args 是被代理目标实例某个方法的具体入参, 在方法反射调用时使用。

（2）控制反转（IOC）

将需要的对象通过注入的方式来实现对象的调用。调用者可以直接使用该对象的方法

new 一个对象-------------》工厂类------------------》DI

2.Spring的优点

1）降低了组件之间的耦合性，实现了软件各层之间的解耦

2）对主流框架提供了集成支持，如hibernate,mabatis，SpringMVC,Struts 等，开发者也可以自由选择 Spring全部或者部分组件

3）Spring属于低侵入模式，代码污染极低

4）提供了众多服务支持，如事务管理，消息队列等

3.Spring管理事务的方式有几种方式？

两种；（1）编程式事务 在代码中硬编码

（2）声明式事务 在配置文件中配置

声明式事务又分为两种

基于XML的声明式事务

基于注解的声明式事务

4.Spring bean 的作用域

Spring容器中的bean可以分为5个范围

1） singleton: 范围默认，这种范围不管接受到多少个请求，每个容器中只有一个bean的实例，单利的模式 由bean factory自身维护

2）prototype: 为每一个bean请求提供一个实例

3）request : 在请求bean范围内为每一个我来自客户端的网络请求创建一个实例，在请求完成后，bean失效并会被垃圾回收

4）Session;确保每个session中有一个实例，在session过期后，备案会随之失效

5）global-session:global-session 和Protlet应用相关。当你的应用部署在Portlet容器中工作时，它包含很多Portlet。如果想要声明让所有的portlet共用全局的存储变量的话，那么全局变量需要存储在global-session中。

5.Spring 中自动装配的方式有哪些？

1.不用装配

2.Byname

通过属性得分名字查找java bean 依赖的对象进行注入。

3.byType

通过属性的类型查找javaBean依赖的对象进行注入

4.construtor

和 byType一样，也是通过类型查找并依赖对象。与ByType的区别在于它不是通过Setter方法注入，而是通过构造方法注入

5.autodetect

在 byType和construts之间自动的选择注入方式

6.default 由上级标签<beans>的default-autowrie属性确定

6.Spring 框架中都用到了哪些设计模式？

（1）工厂模式：BeanFactory就是简单工厂模式的体现，用来创建对象的实例；

（2）单例模式：Bean默认为单例模式。

（3）代理模式：Spring的AOP功能用到了JDK的动态代理和CGLIB字节码生成技术；

（4）模板方法：用来解决代码重复的问题。比如. RestTemplate, JmsTemplate, JpaTemplate。

（5）观察者模式：定义对象键一种一对多的依赖关系，当一个对象的状态发生改变时，所有依赖于它的对象都会得到通知被制动更新，如Spring中listener的实现--ApplicationListener。

7.Spring事务的实现方式和实现原理：

Spring事务的本质其实就是数据库对事务的支持，没有数据库的事务支持，spring是无法提供事务功能的。真正的数据库层的事务提交和回滚是通过binlog或者redo log实现的。

（1）Spring事务的种类：

spring支持编程式事务管理和声明式事务管理两种方式：

①编程式事务管理使用TransactionTemplate。

②声明式事务管理建立在AOP之上的。其本质是通过AOP功能，对方法前后进行拦截，将事务处理的功能编织到拦截的方法中，也就是在目标方法开始之前加入一个事务，在执行完目标方法之后根据执行情况提交或者回滚事务。

声明式事务最大的优点就是不需要在业务逻辑代码中掺杂事务管理的代码，只需在配置文件中做相关的事务规则声明或通过@Transactional注解的方式，便可以将事务规则应用到业务逻辑中。

声明式事务管理要优于编程式事务管理，这正是spring倡导的非侵入式的开发方式，使业务代码不受污染，只要加上注解就可以获得完全的事务支持。唯一不足地方是，最细粒度只能作用到方法级别，无法做到像编程式事务那样可以作用到代码块级别。

（2）spring的事务传播行为：

spring事务的传播行为说的是，当多个事务同时存在的时候，spring如何处理这些事务的行为。

① PROPAGATION\_REQUIRED：如果当前没有事务，就创建一个新事务，如果当前存在事务，就加入该事务，该设置是最常用的设置。

② PROPAGATION\_SUPPORTS：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就以非事务执行。‘

③ PROPAGATION\_MANDATORY：支持当前事务，如果当前存在事务，就加入该事务，如果当前不存在事务，就抛出异常。

④ PROPAGATION\_REQUIRES\_NEW：创建新事务，无论当前存不存在事务，都创建新事务。

⑤ PROPAGATION\_NOT\_SUPPORTED：以非事务方式执行操作，如果当前存在事务，就把当前事务挂起。

⑥ PROPAGATION\_NEVER：以非事务方式执行，如果当前存在事务，则抛出异常。

⑦ PROPAGATION\_NESTED：如果当前存在事务，则在嵌套事务内执行。如果当前没有事务，则按REQUIRED属性执行。

（3）Spring中的隔离级别：

① ISOLATION\_DEFAULT：这是个 PlatfromTransactionManager 默认的隔离级别，使用数据库默认的事务隔离级别。

② ISOLATION\_READ\_UNCOMMITTED：读未提交，允许另外一个事务可以看到这个事务未提交的数据。

③ ISOLATION\_READ\_COMMITTED：读已提交，保证一个事务修改的数据提交后才能被另一事务读取，而且能看到该事务对已有记录的更新。

④ ISOLATION\_REPEATABLE\_READ：可重复读，保证一个事务修改的数据提交后才能被另一事务读取，但是不能看到该事务对已有记录的更新。

⑤ ISOLATION\_SERIALIZABLE：一个事务在执行的过程中完全看不到其他事务对数据库所做的更新。

8、解释一下Spring AOP里面的几个名词：

（1）切面（Aspect）：被抽取的公共模块，可能会横切多个对象。 在Spring AOP中，切面可以使用通用类（基于模式的风格） 或者在普通类中以 @AspectJ 注解来实现。

（2）连接点（Join point）：指方法，在Spring AOP中，一个连接点 总是 代表一个方法的执行。

（3）通知（Advice）：在切面的某个特定的连接点（Join point）上执行的动作。通知有各种类型，其中包括“around”、“before”和“after”等通知。许多AOP框架，包括Spring，都是以拦截器做通知模型， 并维护一个以连接点为中心的拦截器链。

（4）切入点（Pointcut）：切入点是指 我们要对哪些Join point进行拦截的定义。通过切入点表达式，指定拦截的方法，比如指定拦截add\*、search\*。

（5）引入（Introduction）：（也被称为内部类型声明（inter-type declaration））。声明额外的方法或者某个类型的字段。Spring允许引入新的接口（以及一个对应的实现）到任何被代理的对象。例如，你可以使用一个引入来使bean实现 IsModified 接口，以便简化缓存机制。

（6）目标对象（Target Object）： 被一个或者多个切面（aspect）所通知（advise）的对象。也有人把它叫做 被通知（adviced） 对象。 既然Spring AOP是通过运行时代理实现的，这个对象永远是一个 被代理（proxied） 对象。

（7）织入（Weaving）：指把增强应用到目标对象来创建新的代理对象的过程。Spring是在运行时完成织入。

切入点（pointcut）和连接点（join point）匹配的概念是AOP的关键，这使得AOP不同于其它仅仅提供拦截功能的旧技术。 切入点使得定位通知（advice）可独立于OO层次。 例如，一个提供声明式事务管理的around通知可以被应用到一组横跨多个对象中的方法上（例如服务层的所有业务操作）

9.Spring通知有哪些类型？

（1）前置通知（Before advice）：在某连接点（join point）之前执行的通知，但这个通知不能阻止连接点前的执行（除非它抛出一个异常）。

（2）返回后通知（After returning advice）：在某连接点（join point）正常完成后执行的通知：例如，一个方法没有抛出任何异常，正常返回。

（3）抛出异常后通知（After throwing advice）：在方法抛出异常退出时执行的通知。

（4）后通知（After (finally) advice）：当某连接点退出的时候执行的通知（不论是正常返回还是异常退出）。

（5）环绕通知（Around Advice）：包围一个连接点（join point）的通知，如方法调用。这是最强大的一种通知类型。 环绕通知可以在方法调用前后完成自定义的行为。它也会选择是否继续执行连接点或直接返回它们自己的返回值或抛出异常来结束执行。 环绕通知是最常用的一种通知类型。大部分基于拦截的AOP框架，例如Nanning和JBoss4，都只提供环绕通知。

同一个aspect，不同advice的执行顺序：

①没有异常情况下的执行顺序：

around before advice

before advice

target method 执行

around after advice

after advice

afterReturning

②有异常情况下的执行顺序：

around before advice

before advice

target method 执行

around after advice

after advice

afterThrowing:异常发生

java.lang.RuntimeException: 异常发生

1.1Spring MVC工作流程

映射器：主要是根据浏览器上输入的url来映射所有满足要求的Handle（控制器类）

适配器：主要是决定调用哪个Handler来实现具体的业务逻辑

1.2Spring MVC VS Struts2

1）springmvc的入口是一个servlet，即前端控制器；

struts2入口是一个filter过虑器，即前端过滤器，

2）springmvc是基于方法开发（控制器类是单例的，不可能维护实体变量），传递参数是通过方法形参，可以设计为单例；

struts2是基于类开发（维护一个实体变量），传递参数是通过类的属性，只能设计为多例

3）springmvc通过参数解析器是将request对象内容进行解析成方法形参，将响应数据和页面封装成ModelAndView对象，最后又将模型数据通过request对象传输到页面；

struts采用值栈存储请求和响应的数据，通过OGNL存取数据

1. springmvc开发运行速度快于struts2

mybats

组件

SqlSessionFactoryBuilder：构造器

SqlSessionFactory：工厂接口，由Builder生成

基础文件配置，用于配置数据库信息等

typeAlias:定义元素别名，代表com.learn.ssm.chapter3.pojo.Role类可以使用自定义别名

environment:配置使用数据库的链接信息

dataSource:设置mubatis内部提供连接池的方式

mapper:引入自定的映射器配置文件

映射文件配置，配置结合业务所实现的SQL语句关系

SqlSession：会话，发送sql，返回结果

核心接口，等同于JDBC里面的Connection链接对象，

1、获取Mapper接口

2、发送SQL个数据库

3、控制数据库事务

DefaultSqlSession

单线程使用

SqlSessionManager

多线程使用

SQL Mapper：映射器，由java和xml文件构成，构造映射规则，发送SQL，执行完后返回结果

组成部分

java实体对象

interface Mapper类

xml映射文件

namespace：mapper接口对应的全路径名称，例：com.sun.ssm.dao.UserDao

resultMap：定义的model实体对象，id=map配置的名称，type=对象实体的全路径名称（com.sun.ssm.model.User），注意id的名称为下面sql查询配置所使用到实体类型

<id column="id" property="id" jdbcType="BIGINT"/> 设置主键类型

<result column="user\_name" property="userName" jdbcType="VARCHAR"/>

column对应数据库列名

property对应实体对象属性名

jdbcType数据库映射类型

select、update、delete、add对应数据库的增删改查语句

id：对应interface接口里面的方法名称

resultMap：sql语句返回的类型，对应resultMap设置的id名称

parameterType：如果sql语句需要单个参数，设置请求参数类型

#{参数设置}

单个参数：SELECT \* FROM t\_user WHERE id = #{userId} ---对应---selectUserById(@Param("userId") Long userId);

多个参数：SELECT \* FROM t\_user WHERE user\_email = #{emailOrPhone} AND user\_state = #{state} ---对应---selectUserByPhoneOrEmail(@Param("emailOrPhone") String emailOrPhone, @Param("state") Short state);

注解形式实现(此种形式不建议使用)

创建新的接口 interface2

@select("select \*from tb where id=#{id}")

配置文件mapper 链接由resource改为class，地址为接口全路径

生命周期

SqlSessionFactoryBuilder用于创建SqlSessionFactory，创建成功后即失效

SqlSessionFactory等同于多个数据库的连接池（整个mysql），在整个mybatis的生命周期中，如果创建多个，则会耗光连接池，造成应用崩溃，所以一般为单例模式，整个应用进行共享

SqlSession对应单个数据库进行的命令操作，如果使用完，则会把链接返还给SqlSessionFactory连接池

Mapper由session创建，一个mapper请求对应一个业务处理

配置

配置文件

配置文件节点顺序不能颠倒

配置文件xml

configuration：根节点

properties：属性

settings：设置

typeAliases：类型命名

typeHandlers：类型处理器

objectFactory：对象工厂

plugins：插件

environments：配置环境

environment：环境变量

transactionManager：事务管理器

dataSource：数据源

databaseIdProvider：数据库厂商标识

mappers：对象映射器

xml 属性节点介绍

properties：属性（优先级：程序代码＞文件＞子元素）

property子元素

<property name="data.driver" value="com.mysql.jdbc.Driver">

<property name="data.url" value="jdbc:mysql://localhost:3306">

然后在dataSource里面使用<property name="driver" value="${data.driver}">等同于占位符

properties文件

创建一个jdbc.properties文件

data.url=jdbc:mysql://......

data.username=root

<properties resource="jdbc.properties">

程序代码传递

如果需要对字符串加密，则在程序里面读取配置文件，然后写入解密后的内容

settings：设置

<setting name=“cacheEnabled” value=“true”>是否启用缓存

等等。。。还有很多配置项

cacheEnabled--true | false--true（name--value--默认值）该配置影响的所有映射器中配置的缓存的全局开关

lazyLoadingEnabled--true | false--false 延迟加载的全局开关。当开启时，所有关联对象都会延迟加载。 特定关联关系中可通过设置fetchType属性来覆盖该项的开关状态

aggressiveLazyLoading--true | false--true 当启用时，对任意延迟属性的调用会使带有延迟加载属性的对象完整加载；反之，每种属性将会按需加载。

multipleResultSetsEnabled--true | false--true 是否允许单一语句返回多结果集（需要兼容驱动）。

useColumnLabel--true | false--true 使用列标签代替列名。不同的驱动在这方面会有不同的表现， 具体可参考相关驱动文档或通过测试这两种不同的模式来观察所用驱动的结果。

useGeneratedKeys--true | false--false 允许 JDBC 支持自动生成主键，需要驱动兼容。 如果设置为 true 则这个设置强制使用自动生成主键，尽管一些驱动不能兼容但仍可正常工作（比如 Derby）。

autoMappingBehavior--NONE, PARTIAL, FULL--PARTIAL 指定 MyBatis 应如何自动映射列到字段或属性。 NONE 表示取消自动映射；PARTIAL 只会自动映射没有定义嵌套结果集映射的结果集。 FULL 会自动映射任意复杂的结果集（无论是否嵌套）。

defaultExecutorType--SIMPLE, REUSE, BATCH--SIMPLE 配置默认的执行器。SIMPLE 就是普通的执行器；REUSE 执行器会重用预处理语句（prepared statements）； BATCH 执行器将重用语句并执行批量更新。

defaultStatementTimeout--int-- 设置超时时间，它决定驱动等待数据库响应的秒数。

safeRowBoundsEnabled--true | false--False允许在嵌套语句中使用分页（RowBounds）。

mapUnderscoreToCamelCase--true | false--False是否开启自动驼峰命名规则（camel case）映射，即从经典数据库列名 A\_COLUMN 到经典 Java 属性名 aColumn 的类似映射。

localCacheScope--SESSION | STATEMENT--SESSION MyBatis 利用本地缓存机制（Local Cache）防止循环引用（circular references）和加速重复嵌套查询。 默认值为 SESSION，这种情况下会缓存一个会话中执行的所有查询。 若设置值为 STATEMENT，本地会话仅用在语句执行上，对相同 SqlSession 的不同调用将不会共享数据。

jdbcTypeForNull-- 当没有为参数提供特定的 JDBC 类型时，为空值指定 JDBC 类型。 某些驱动需要指定列的 JDBC 类型，多数情况直接用一般类型即可，比如 NULL、VARCHAR 或 OTHER。

lazyLoadTriggerMethods --equals,clone,hashCode,toString 指定哪个对象的方法触发一次延迟加载。

defaultScriptingLanguage--org.apache.ibatis.scripting.xmltags.XMLDynamicLanguageDriver 指定动态 SQL 生成的默认语言。

callSettersOnNulls--true | false--false 指定当结果集中值为 null 的时候是否调用映射对象的 setter（map 对象时为 put）方法，这对于有 Map.keySet() 依赖或 null 值初始化的时候是有用的。注意基本类型（int、boolean等）是不能设置成 null 的。

logPrefix--指定 MyBatis 增加到日志名称的前缀 String

logImpl--指定 MyBatis 所用日志的具体实现，未指定时将自动查找。 SLF4J | LOG4J | LOG4J2 | JDK\_LOGGING | COMMONS\_LOGGING | STDOUT\_LOGGING | NO\_LOGGING

proxyFactory--CGLIB | JAVASSIST--JAVASSIST指定 Mybatis 创建具有延迟加载能力的对象所用到的代理工具。

typeAliases：别名

可以用简写的形式自定义类型名称

比如：int--Integer（mybits--java）

使用代码进行注册

自定义别名

xml：<typeAlias alias="user" type="类的全称地址">

扫描形式：<package name="需要扫描的包"> 如果当前包里面有2个类，都会自动转成pacal命名法的别名（首字母小写，其余大写）

代码里面可以针对类设置 在class类上面增加 @Alias("sun")，则在扫描时，名称为自定义名称

typeHandler：类型转换器

用于承担 数据库与java之间的类型转换，jdbcType--javaType，mybatis会进行自动处理

自定义Handler（可以用于枚举转int，等等）

集成baseTypeHandler接口，自己实现4个方法

在配置文件中使用

在全局xml使用：<typeHandler jdbcType="数据库类型" javaType="java类型" jandler="handler实现类的包名全路径">

在mapper映射xml文件使用：<resule property="java类型" colum="数据库类型" typeHandler="handler实现类的包名全路径">

如果自定义处理类过多，使用扫描包形式加载

<package name="包的全路径">

此种形式无法定义类型映射，可以在自定义实现类class上面使用注解形式添加映射，如：@MappedTypes（String.class） @MappedjdbcTypes（jdbcType.VARCHAR）

objectFactory：对象工厂

一般使用默认就好，创建结果集时，mybatis会创建结果集的实例，用于返回给用户。

plugins：插件

environments：运行环境

用于配置数据库的链接信息

transactionManager：事务管理器

JdbcTransaction，以jdbc的事务形式进行处理

ManagedTransaction，使用mybits自定义的对象事务处理，提交和回滚不进行操作，把事务交给容器处理，默认会关闭链接。

dataSource

分为3种形式链接数据库，可以进行同时配置<dataSource type="POOLED">...

UNPOOLED

使用非连接池形式，每次请求，创建一个新的连接

POOLED

使用连接池形式，节省资源，可以设置最大连接数

poolMaxmumActiveConnections：任意时间活跃的链接，默认10

poolMaxmumIdleConnections：任意时间空闲的连接数

poolMaxmumCheckoutTime：池中检出的时间，默认20000（20秒）

poolTimeToWait：如果获取链接时间过长，会打印状态日志

poolPingQuery：侦测查询，用于判断数据库是否可以正常查询

poolPingEnabled：是否启用侦测查询，默认不开启（如果开启，请设置一个效率非常快的SQL，例如 select 1）

poolPingConnectionNotUsedFor：配置侦测查询的频度

JNDI

databaseIdProvider：数据库厂商标识

用于配置支持多种数据库操作

<databaseIdProvider type="DB\_VENDOR"><property name="MYSQL" value="mysql" /><property name="Oracle" value="oracle" /><property name="DB2" value="db2" /></databaseIdProvider >

配置完成之后，在Mapper映射器里面 对sql语句进行不同数据的查询切换操作，例如<select id="aa" ... datanaseId="oracle">或者<select id="aa" ... datanaseId="mysql">

mappers ：对象映射器

先编写接口代码，创建对应增删改查接口，映射文件里面首行<mapper>标签会使用namespace定义接口代码全路径，下面设置sql查询语句（select、delete、....）

引入方式：

文件引入：<mapper resource="xml全路径">

包名引入：<package name="包全路径">

类注册引入：<mapper class="映射接口类的全路径">

userMapper.xml：暂时不考虑

映射器

一个映射xml对应一个接口类型，然后对应对象

select

默认sql返回的列名需要和对象属性一一对应上，例如，role\_name对应roleName，按照驼峰命名法则映射，如果列名直接使用role\_name as roleName形式返回，则不用使用默认驼峰映射

xml节点属性配置

id：方法名称，供Mybatis使用，如果相同命名空间id不唯一，则会报错

parameterTyppe：传递参数类型，可以是int、string、或者类

List<map>：接口类使用map，传递查询参数，类似.net的HashTable，不方便阅读维护

@Param("占位符名称1")：接口类也可以使用多个Param进行传递，xml映射文件可以直接使用变量，例如：<select id="findName" resultType="返回对象类">select id,role\_Name as roleName,create\_Time as createTime from tb\_User where role\_Name like concat('%',#{roleName},'%') and id=#{id}</select>，参数过多时不方便使用

使用javaBean形式（java class类对象）：传入一个class类型作用请求，paramType="类的全路径"

混合使用：javaBean与Param同时使用。@Param("user") User user,@Param("name") Name name，在xml占位符中则为#{user.age}

resultType：返回结果类型，可以是int、string、或者类

resultMap：返回结果自定义使用，比如说使用typeHandler

xml：映射配置文件上方 会使用resultMap定义一个数据库查询类型与POJO对象的映射配置节点，下方select查询语句中使用resultMap="上方定义节点的id属性的名称"

自定义Handler：

flushCache：调用完SQL之后，是否清空查询的本地缓存和二级缓存，默认false，不进行清理

useCache：是否启动二级缓存，默认true，启用二级缓存

timeout：设置当前语句查询超时时间

fetchSize：获取记录返回总条数，如果设定100，最大返回100条数据

statementType：设置mybatis使用JDBC的哪种Statement工作，默认PREPARED,（个人不太理解）

resultSetType:

databaseId：使用多种数据库引擎支持的ID

resultOrdered：

resultSets：使用于多个结果集情况，一次查询返回多个dataTable表格数据

RowBounds：myBatis内置分页类对象。在DAO接口类中，定义RowBounds请求参数，xml不需要进行配置，因为已经内置在sqlSession中。（因为他的原理是先把sql数据全部查出来在内存里面进行分页，大数据量并不合适）

insert

xml节点属性配置（大部分同select相同，只记录不同项）

useGeneratedKeys：主键回填，是否启用使用jdbc里面的同名方法获取内部生成的主键（类似sqlserver自增主键），默认false（启用之后，keyProperty与keyColumn选一个）。使用方法：<insert ..... useGeneratedKeys="true" keyProperty="id">，意思是-如果插入完数据，则会回填keyProperty设置的属性值

keyProperty：会根据useGeneratedKeys的返回值，标记一个属性返回值，默认不启用

keyColumn：不能和keyProperty同时使用

结构层级：insert → selectKey（keyProperty：回填主键名称，resultType：回填值类型，order：是插入语句执行前使用还是执行后使用，默认BEFORE），设置了order之后，这样在insert语句真正执行前，id主键会按照自定义的规则生成，然后插入到对应列中

update、delete（比较简单，一般使用单个参数或者javaBean参数形式，返回结果为int型影响行数）

sql：允许定义一部分sql，在各个地方引用

在mapper根节点下，设置sql节点，里面包含通用变量，则在select或者insert等操作时统一调用，例如：<sql id="fileds"> id,name,age</sql>，在select语句里面，<select id="findOne" .....>select <include refid="fileds" /> fom User </select>，insert 同理。

节点还支持变量传递 <sql id="fileds"> ${U}.id,${U}.name,${U}.age</sql>，整体使用：<select id="findOne" .....> select <include refid="fileds"> <propety name="U" value="tb" /> </include> from TbUser as tb</select>。（refid的值为自定义sql的id，里面propety节点代表属性内容， name值为自定义sql使用占位符变量，value值为 查询语句中表的别名，name与value为映射关系）

resultMap：描述数据库结果集映射到的对象

xml节点

resultMap

constructor：适用于对象里面带参数的构造函数

idArg

arg

id

result

association

collection

discriminator

case