## [Spring知识点整理](https://www.cnblogs.com/april-chen/p/11527003.html)

## **1.bean什么时候被实例化**

第一：如果你使用BeanFactory作为Spring Bean的工厂类，则所有的bean都是在第一次使用该Bean的时候实例化  
第二：如果你使用ApplicationContext作为Spring Bean的工厂类，则又分为以下几种情况：  
（1）：如果bean的scope是singleton的，并且lazy-init为false（默认是false，所以可以不用设置），则ApplicationContext启动的时候就实例化该Bean，并且将实例化的Bean放在一个map结构的缓存中，下次再使用该Bean的时候，直接从这个缓存中取  
（2）：如果bean的scope是singleton的，并且lazy-init为true，则该Bean的实例化是在第一次使用该Bean的时候进行实例化  
（3）：如果bean的scope是prototype的，则该Bean的实例化是在第一次使用该Bean的时候进行实例化

## **Bean类型**

## - singleton: 单例 默认模式，在Spring IOC容器中只存在一个共享的Bean实例。 - prototype：原型模式 每次使用时都会创建一个新的Bean实例。 - request：一次Http请求一个实例 - session：同一个Http Session共享一个Bean实例 - global session：在一个全局的Http Session中，容器会返回该Bean的同一个实例，仅在使用Portlet context时有效。



## **3.BeanFactory和ApplicationContext的区别**

（1） BeanFactory：是Spring里面最底层的接口，包含了各种Bean的定义，读取bean配置文档，管理bean的加载、实例化，控制bean的生命周期，维护bean之间的依赖关系。ApplicationContext接口作为BeanFactory的派生，除了提供BeanFactory所具有的功能外，还提供了更完整的框架功能：  
①继承MessageSource，因此支持国际化。  
②统一的资源文件访问方式。  
③提供在监听器中注册bean的事件。  
④同时加载多个配置文件。  
⑤载入多个（有继承关系）上下文 ，使得每一个上下文都专注于一个特定的层次，比如应用的web层。  
（2）①BeanFactroy采用的是延迟加载形式来注入Bean的，即只有在使用到某个Bean时(调用getBean())，才对该Bean进行加载实例化。这样，我们就不能发现一些存在的Spring的配置问题。如果Bean的某一个属性没有注入，BeanFacotry加载后，直至第一次使用调用getBean方法才会抛出异常。  
（3）BeanFactory通常以编程的方式被创建，ApplicationContext还能以声明的方式创建，如使用ContextLoader。  
（4）BeanFactory和ApplicationContext都支持BeanPostProcessor、BeanFactoryPostProcessor的使用，但两者之间的区别是：BeanFactory需要手动注册，而ApplicationContext则是自动注册。

## **[spring mvc常用知识点总结](https://www.cnblogs.com/panxuejun/p/6480778.html)**

1.spring mvc是靠spring 启动的。通过springjar包的org.springframework.web.servlet.DispatcherServlet这个servlet类具体启动的。  
<servlet-name>springmvc</servlet-name>  
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

这个dispatcherservlet这个类，有个contextConfigLocation参数，指定springmvc的配置文件xml。  
2.理清一下思路：每个tomcat下的程序war包都有一个web.xml。web.xml可以启动servlet。  
web.xml只能配置servlet相关信息，如servletmapping等等，不能配置bean的注入，扫描包路径，注解驱动等

.配置解析

　　1.Dispatcherservlet

　　DispatcherServlet是前置控制器，配置在web.xml文件中的。拦截匹配的请求，Servlet拦截匹配规则要自已定义，把拦截下来的请求，依据相应的规则分发到目标Controller来处理，是配置spring MVC的第一步。  
4. <!-- scan the package and the sub package -->  
<context:component-scan base-package="test.SpringMVC"/>  
  
<!-- don't handle the static resource -->  
<mvc:default-servlet-handler />  
  
<!-- if you use annotation you must configure following setting -->  
<mvc:annotation-driven />  
  
<!-- configure the InternalResourceViewResolver -->  
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"  
id="internalResourceViewResolver">  
<!-- 前缀 -->  
<property name="prefix" value="/WEB-INF/jsp/" />  
<!-- 后缀 -->  
<property name="suffix" value=".jsp" />  
</bean>  
</beans>

5.spring mvc配置文件里的配置标签，必背：  
<context:component-scan base-package="test.SpringMVC"/>  
<mvc:annotation-driven />  
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"  
  
<!-- scan the package and the sub package -->  
<context:component-scan base-package="test.SpringMVC"/>  
  
<!-- don't handle the static resource -->  
<mvc:default-servlet-handler />  
  
<!-- if you use annotation you must configure following setting -->  
<mvc:annotation-driven />  
  
<!-- configure the InternalResourceViewResolver -->  
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"  
id="internalResourceViewResolver">  
<!-- 前缀 -->  
<property name="prefix" value="/WEB-INF/jsp/" />  
<!-- 后缀 -->  
<property name="suffix" value=".jsp" />  
</bean>  
</beans>

6.spring mvc的那些controller标签，是注入到spring的ioc容器中的。不是spring mvc自己的ioc，spring mvc没有自己的ioc容器，都是spring的

7.@RequestBody

　　该注解用于读取Request请求的body部分数据，使用系统默认配置的HttpMessageConverter进行解析，然后把相应的数据绑定到要返回的对象上 ,再把HttpMessageConverter返回的对象数据绑定到 controller中方法的参数上

　　@ResponseBody

　　该注解用于将Controller的方法返回的对象，通过适当的HttpMessageConverter转换为指定格式后，写入到Response对象的body数据区

8.requestParam正确用法@RequestParam(value = "invitCode", required = false) string invitCode  
requestParam修饰的东西全在小括号里面（）。

9.四、自动匹配参数  
//match automatically  
@RequestMapping("/person")  
public String toPerson(String name,double age){  
System.out.println(name+" "+age);  
return "hello";  
}  
　自动装箱

　　1.编写一个Person实体类  
package test.SpringMVC.model;  
  
public class Person {  
public String getName() {  
return name;  
}  
public void setName(String name) {  
this.name = name;  
}  
public int getAge() {  
return age;  
}  
public void setAge(int age) {  
this.age = age;  
}  
private String name;  
private int age;  
  
}

　　2.在Controller里编写方法  
  
//boxing automatically  
@RequestMapping("/person1")  
public String toPerson(Person p){  
System.out.println(p.getName()+" "+p.getAge());  
return "hello";  
}

10.在SpringMVC中，bean中定义了Date，double等类型，如果没有做任何处理的话，日期以及double都无法绑定。

解决的办法就是使用spring mvc提供的@InitBinder标签

11.比较简单的可以直接应用springMVC的注解@initbinder和spring自带的WebDataBinder类和操作

[java] view plain copy  
在CODE上查看代码片派生到我的代码片

@InitBinder  
public void initBinder(WebDataBinder binder) {  
SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd");  
dateFormat.setLenient(false);  
binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, true));  
}

还要在springMVC配置文件中加上

[html] view plain copy  
在CODE上查看代码片派生到我的代码片

<!-- 解析器注册 -->  
<bean class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter">  
<property name="messageConverters">  
<list>  
<ref bean="stringHttpMessageConverter"/>  
</list>  
</property>  
</bean>  
<!-- String类型解析器，允许直接返回String类型的消息 -->  
<bean id="stringHttpMessageConverter" class="org.springframework.http.converter.StringHttpMessageConverter"/>

## Mybatis知识点总结

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

xml节点内部配置：

selectKey：自定义主键，可以根据自定义的设置规则，生成主键值，进行回填和插入，例如：下图

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

节点注释

resultMap：type属性对应对象的类型

id：设置对应的主键

result：字段映射节点，例如：<result property="对象属性" column="数据库列名" />

association：级联查询，适用于一对一的情况

例子：<association property="当前对象子对象的class类型" column="对应关联属性的列名" select="要调用其他接口类的方法全路径" />

property：通过调用select对应的接口方法，返回的对象类型

column：子查询所需要传入的父查询ID，例如：select (select top 1 name from CHIRLD as b where b.pid=a.id),\* from PARENT as A，等同于传入子查询的a.id的功能

collection：级联查询，适用于一对多的情况

例子：<collection property="当前对象子对象的class类型" column="对应关联属性的列名" select="要调用其他接口类的方法全路径" />

节点属性等同于association

discriminator：级联查询，选择器查询，根据不同的值调用不用的查询器

例子：<discriminator javaType="选择列的数据类型" column="选择列的名称" > <case value="如果是当前值，1" resultMap="当前页面其他查询器ID1" /><case value="如果是当前值，2" resultMap="当前页面其他查询器ID2" /> </discriminator>，当前映射xml文件会存在多个resultMap节点，其他从节点：<resultMap type="返回对象类型的全路径" id="当前页面其他查询器ID1" extends="继承与哪个resultMap(为主节点ID)"> <association property="返回子对象class类型(首字母小写)" column="对应关联属性的列名" select="要调用其他接口类的方法全路径" /></resultMap>，这样通过不同的case 则会调用不同类型的接口查询。

javaType：用于指定传入列的类型

column：传入列的列名

还需要case子节点

value：具体值

resultMap：其他resultMap从节点的ID

还需要指定其他从resultMap节点

type：返回对象类型

id：查询器id

extends：继承于哪个resultMap(为resultMap主节点ID)

子节点：association 或者 collection

property：返回对象属性名称

select：使用哪个查询接口类的全路径

column：传入这个选择的内容是主sql的哪个字段（多个参数，使用逗号分隔）

级联查询的性能问题

因为固定映射写死，所以获取对象则会将级联的数据给查询出来，因为有的时候不需要，所以造成性能浪费。

解决问题如下：

延迟加载

使用setting节点里面的 lazyLoadingEnabled（是否启用） 和 aggressiveLazyLoading （是否执行加载）节点组合

fetchType：设置在级联查询中的节点属性，（association、collection存在，discriminator选择器没有），有两个值：eager、lazy，仍需要配合lazyLoadingEnabled使用，不过需要设置aggressiveLazyLoading=false

eager：获取主对象之后立刻加载对应数据

lazy：获取对象之后，延迟加载关联数据

sql关联查询

使用sql关联将所有从表数据进行查询，设置好别名，wf\_id，如果wf\_id不为空，则进行调用级联查询，在主resultMap节点里面使用级联查询节点配置，缺点是，配置可能过长，不好理解

多对多级联，本质上多对多的关联查询只是表结构，实际业务总会以某张表作为主表进行关联的

cache：给定某个命名空间的缓存配置

将查询数据存入二级缓存

<cache type="使用缓存类的全路径"></cache>

type：自定义缓存类，可以使用Redis、MongoDB等缓存工具，必须要实现ibatis.cache.Cache接口

size：缓存对象的个数，默认1024

flushInterval：刷新缓存的时间（单位毫秒），insert、update、delete才有效

eviction：缓存策略，（默认LRU）

LRU：最近最少使用-移出最长时间不使用的对象

FIFO：先进先出

SORT：软引用，移出基于垃圾回收器状态和软引用规则的对象

WEAK：弱引用，比SORT更积极的回收对象，回收规则通sort一致

readOnly：缓存是否只读，默认false

blocking：使用使用阻塞式缓存，默认不使用

cache-ref：其他命名空间缓存配置的引用

存储过程：

参数：

参数传入的内容存在null形式，这个时候执行sql，可能会报错，如果需要进行自定义Handler处理，则需要在变量里面指定处理器，例如：#{age , javaType=int , jdbcType=NUMERIC , typeHandler=MyTypeHandler}，此时age传入的参数变量，则会经过myTypeHandler处理器进行处理。

存储过程：

输入参数（IN）：外部传入存储过程

输出参数（OUT）：存储过程返回结果

输入输出参数（INOUT）：传入的参数，经过存储过程处理，再返回出去

使用方法，传入参数#{age, mode=IN} 或者#{age, mode=INOUT}

使用方法：

设置POJO对象，用于对应存储过程的传入参数和传出参数

CURD都可以使用存储过程

在对应语句执行器上使用 statementType="CALLABLE"，代表使用存储过程

{ call 存储过程名称(参数1，参数2，参数3)}

参数1、2为输入参数，#{userName，mode=IN，jdbcType=VARCHAR}

userName为POJO对象的属性名称

此处不指定存储过程参数名称是因为，调用存储过程的参数是按照顺序传入的

参数3为输出参数，#{count，mode=OUT，jdbcType=INTEGER}

游标：

缓存

mybatis框架对于缓存分为一、二级缓存两层

一级缓存，存在SqlSession，默认开启

参数和sql没有变化，用同一个SqlSession的Mapper对象，则会将内存中进行读取

不同SqlSession不同享缓存

二级缓存

存在SqlSessionFactory，使用不同的sqlsession对象，查询完成之后，需要使用sqlsession.commit()，进行提交，提交之后才会进行缓存到Factory里面

开始二级缓存，在mapper的xml配置文件中使用<cache />标签

动态SQL

语法元素

if：条件判断

在select查询器里面，select \* from tb where disabled=0 <if test="roleName != null"> and name='#{name}'</if>

test：语句为sql判断格式

节点里面内容为要拼接的sql部分语句

choose：多条件判断，类似 case when then

使用原理同if， <choose><when test=""></when><when test=""></when><otherwise></otherwise> </choose>

otherwise：等同于最后一个else

trim：处理拼装的条件，例如：去掉多余的and 、or等

<trim prefix="" prefixOverrides=""><if test="roleName != null"> and name='#{name}'</if></trim>

在其他语句最外层包一层trim，用于去掉收尾指定的多余参数

prefix：在里面语句生成之后，前面追加固定字符串

prefixOverrides：去掉生成之后的最前方的固定字符串

suffixOverrides：去掉生成之后的最后方的固定字符串

set：用于更新语句

等同于 update Tb <set> <if></if><if></if> </set> where id=#{id}

set标签会自动把最后一个逗号给去掉

foreach：循环

通常应用于in操作，<foreach item="" index="" collection="" open="" separator="" close="">#{roleNo}</foreach>

item：当前元素对象

index：当前元素的下标

collection：传入的参数对象属性，通常为list，set等

open：包裹元素的前置固定字符串

separator：连接参数字符串

close：包裹元素的后置固定字符串

bind：自定义变量

<bind name="name1" value="'%'+ 参数+'%'" />，下面sql使用为 select \* from Tb where name like #{name1}，等同于传入变量使用

特点在于提升了可读性和移植性

运行原理