## SSM

springmvc是spring的一个模块

MVC是设计模式（是对编写经验的总结）

## springmvc流程

第一步 发起请求到前端控制器

第二部 前段控制器请求HandlerMapping 根据xml配置 注解进行查找 返回HandlerExecuteChain

第三部 前段控制器 调用处理器适配器执行Handler

第四部 处理器适配器取执行Handler

第五步 Handler执行完 给适配器返回 Model and View

第六步 处理器向前段控制器返回Model and View包括了model view

第七部 前段控制器请求视图解析器解析Model and View （根据逻辑视图解析成为真正的视图）

第八步 视图解析器向前段控制器返回View

第九布 前端控制器进行视图渲染 把视图模型填充到request域

第十部 前端控制器向用户响应视图

主件：

前端控制器 DispatcherServlet (重要相当于中央处理器)

#减少了其他主件的耦合度

接受数据请求 响应结果 相当于转发器

处理器映射器 HandlerMapping

根据请求的url找到对应的Handler 通过配置文件xml或注解 查找

处理器适配器 HandlerAdapter

按照特定的规则（HandlerAdapter的要求）去执行Handler

视图解析器 Viewresolver

进行视图的解析 根据逻辑视图名 转化成真正的试图

视图 View

View是一个接口 实现不同的视图类型（jsp pdf execl 等等）

注解处理器映射器 处理器适配器

处理器映射器 注解的处理器映射器 <mvc:annotation-driven>可以替代下面的配置路径

处理器适配器 注解的处理器适配器

<context:component-scan base-package="com.frank.springmvc.controller" />

还要写入InternalResourceViewResolver 定向处理ModelAndView

<!-- 配置视图解析器解析路径 -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver" id="internalResourceViewResolver">

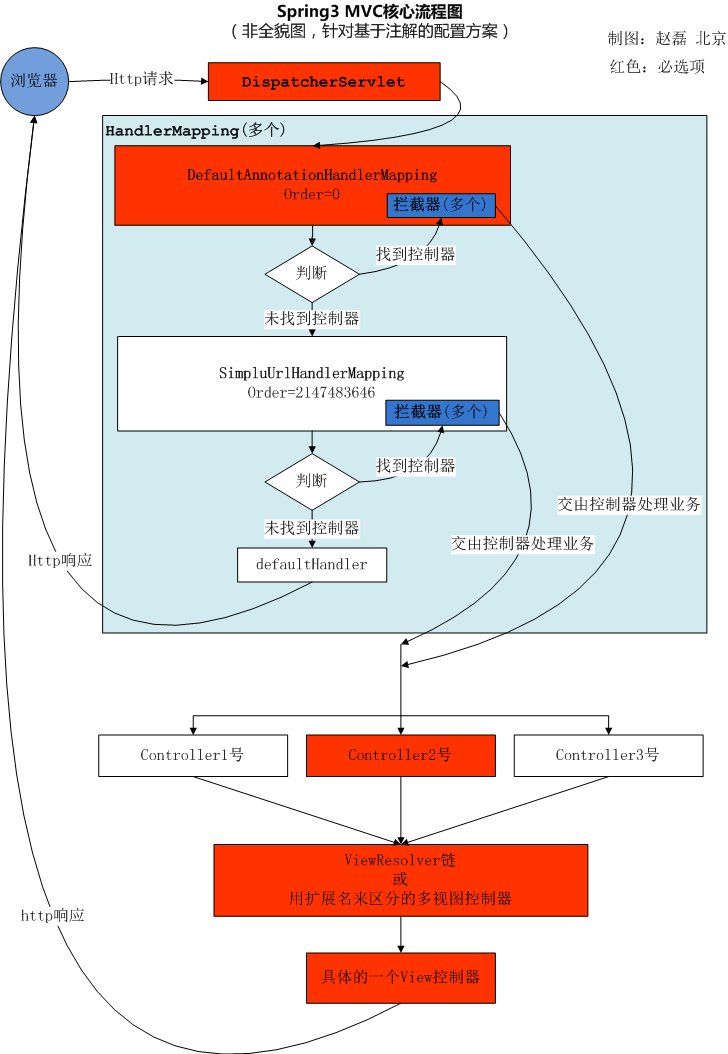
<!-- 定义视图存放路径 -->

<property name="prefix" value="/WEB-INF/jsp/" />

<!-- 定义视图后缀 -->

<property name="suffix" value=".jsp" />

</bean>



使用Spring MVC,配置DispatcherServlet是第一步。

DispatcherServlet是一个Servlet,所以可以配置多个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，依据某某规则分发到目标Controller(我们写的Action)来处理。

1. **<web-app>**
2. **<servlet>**
3. **<servlet-name>**example**</servlet-name>**
4. **<servlet-class>**org.springframework.web.servlet.DispatcherServlet**</servlet-class>**
5. **<load-on-startup>**1**</load-on-startup>**
6. **</servlet>**
7. **<servlet-mapping>**
8. **<servlet-name>**example**</servlet-name>**
9. **<url-pattern>**\*.form**</url-pattern>**
10. **</servlet-mapping>**
11. **</web-app>**

# myBatis

对于原生态jdbc程序问题的总结

可以将sql代码 配置到xml文件之中 实现分离 方便修改

mybatis

mybatis是一个dao持久层框架 是 apache的顶级项目

mybatis可以使我们吧主要精力放在sql编写下 通过mybatis提供的映射方式 自由灵活的生成需要的sql语句

mybatis可以将preparedstatement中输入的参数自动化进行映射 将查询结果集映射成java对象 （输出映射）

### mybatis框架 流程

SqlMapConfig.xml

全局的配置文件

配置了数据源 事务mybatis运行环境

配置映射文件 （配置sql语句）

mapper.xml 映射文件

SqlSessionFactory（会话工厂）

创建SqlSession

SqlSession（会话）

用于操作数据库（发出 sql的增删改查）

Executor（执行器）

SqlSession通过执行器操作数据库

MappedStatement（底层对象的封装）

对操作数据库的储存封装

输入 参数 （可以是java hashmap pojo）

输出 结果 （可以是java hashmap pojo）

先配置配置文件 名称是不固定的

根据配置文件 创建工厂

创建SqlSession 面向用户的接口

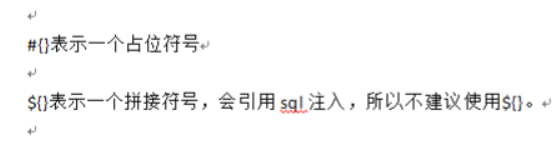
执行器 是一个接口 （基本执行器 缓存执行器）

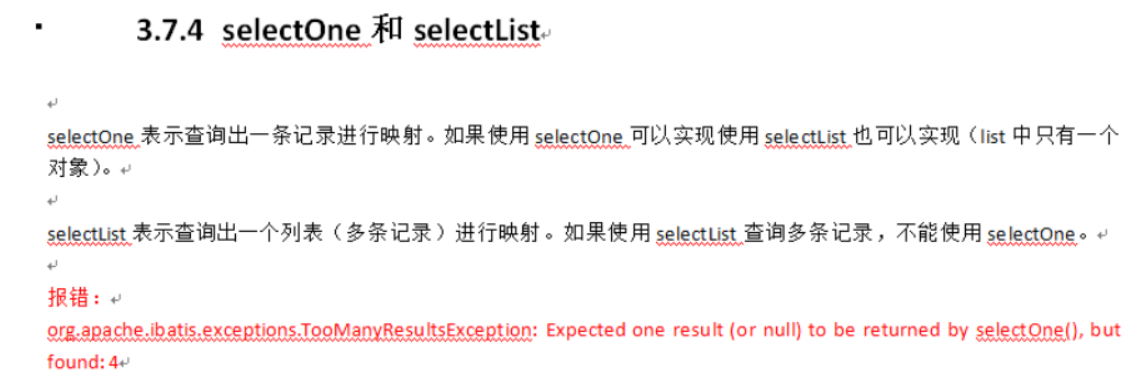
namespace 命名空间

对sql的分类化管理 理解为sql隔离



如果想在Result Type中获得JAvabeanlist 还是定义为JavaBean





ParamterType: 传入数据的类型

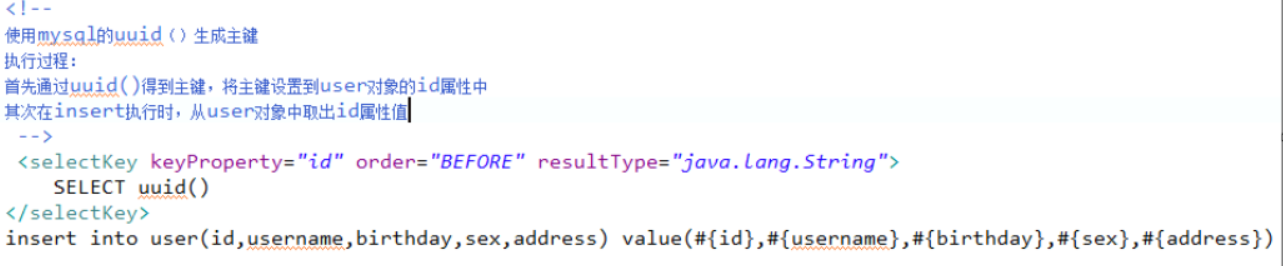


对于last\_insert\_id 获取自增主件的方法：

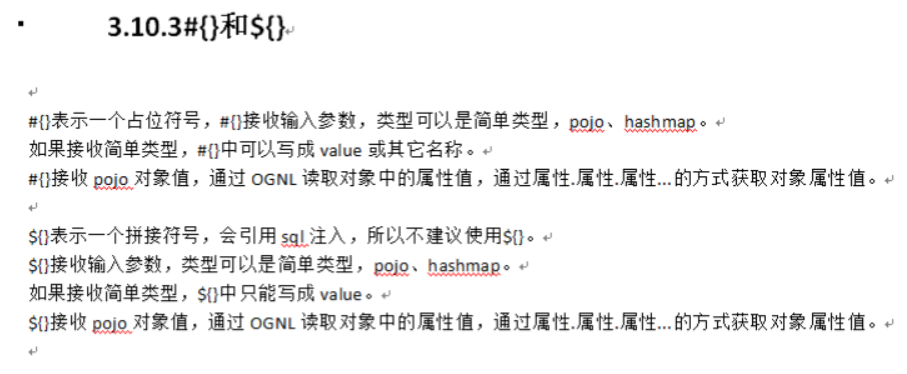


selectKey 标签中也要写 ResultType 指定last\_insert\_id返回的类型

对于非自增主件的返回：



对于#{}${}的详细：



SqlSession是一个接口

SqlSession是线程不安全的 由于定义了多个接口里有属性和域对象

所以应当把SqlSession放在方法体中实现 因为每个方法执行时 内存区域不同



<typeAliases> 标签定义别名（重点）

使用typeAliases标签定义别名 为了减少使用项目的pojo全称

\*批量定义： 传入package标签 name=“package名”



类型处理器：

typeHandlers 对jdbc类型和java数据类型之前的转化

例如 String 转化为 varchar

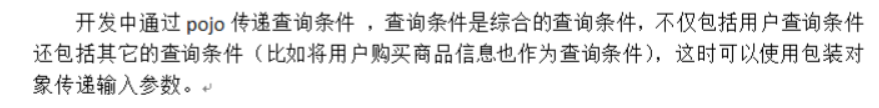
通过mapper接口文件 加载对应的xml sql文件



### 批量加载（推荐）



### 专递pojo包装类型



类似于javabean导航

hashmap的映射：这里hashmap是别名



使用resultMap

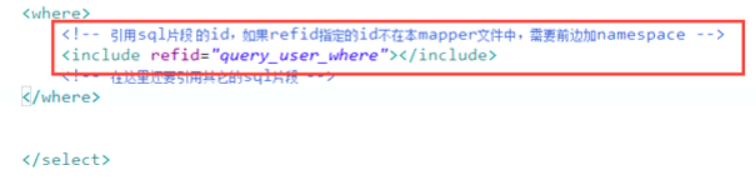
### 对于SQL语句的灵活性封装：（标签类似于jstl）

### C:\Users\Liuxang\AppData\Local\Temp\1528302881(1).png

### 定义sql片段 是的代码更加灵活：



在语句中 include标签refid=”之前定义的SQL的id”



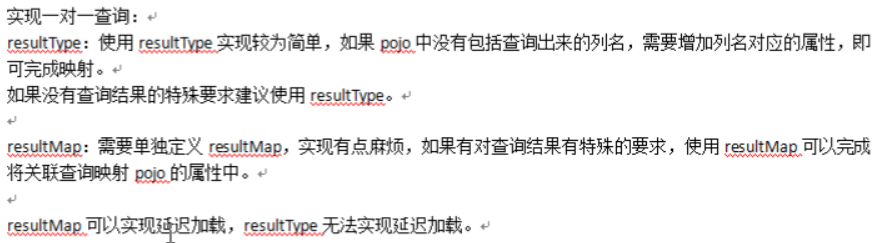
### 使用foreach(这里是小写！！！)循环 将重复性高的sql片段进行拼接



例如 select \* from table where id=1,id=3,id=5,id=10

### 有关于一对一 一对多查询

在一对一查询时 使用resultType比较简单



在使用resuktType时 只要定义了 相对应的属性 就可以 直接进行映射

对于ResultMap的定义：

type标签：指定主pojo

id标签： 名称



association 标签（映射到单个pojo） ： 作为主pojo中 单个子pojolist的对应标签 （一个主pojo中有一个子pojo）

property：要映射到主pojo的什么属性 名称）

javaType: 子pojo的java类型



id： 唯一对应标签

一对多映射：

使用resultMap

使用collection标签 property=“主pojo中需要映射到的字pojo名称” ofType=“要存放的子pojo路径”



其中extends属性表示可以继承其他resultMap 使操作简单化 减少代码量

extends=”之前的resultMap的id”

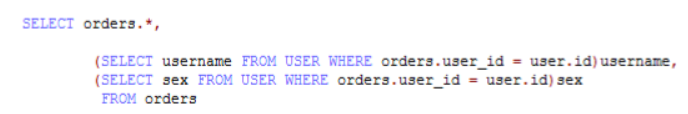
在resultMap之中可以在collection标签下 添加collection标签 进行多表之间的list映射

如何选用resultMap resultType

在需要list映射时 使用resultMap

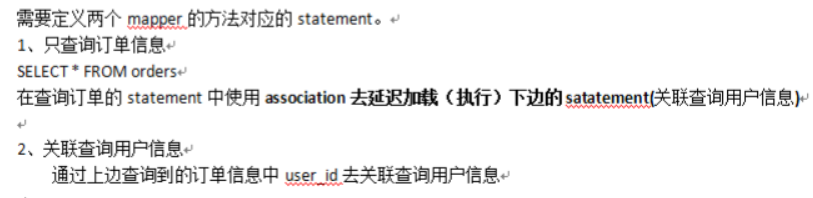
## 延迟映射：

对于多表查询 可以使用延迟加载



对于这样的sql 可以先加载select order.\* from orders

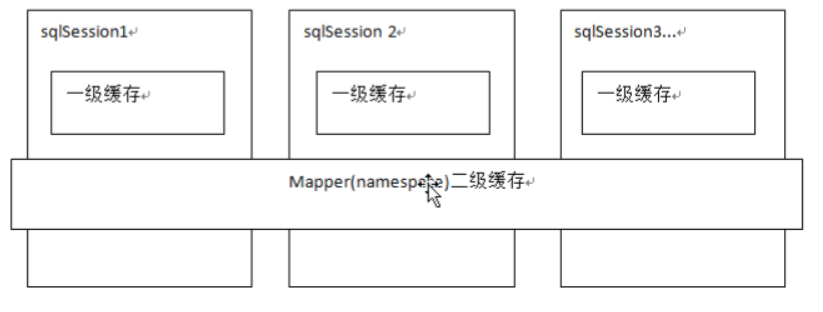
再分布加载 select username from user where …





相关博客：<https://blog.csdn.net/eson_15/article/details/51668523>

### 一级缓存

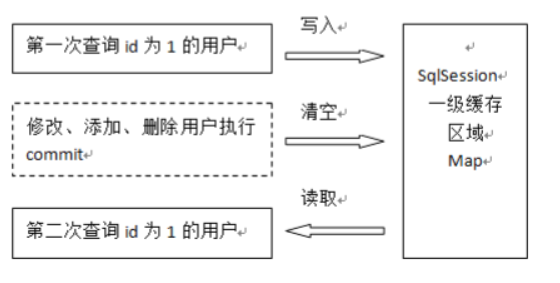


对于一级缓存 ：

一级缓存是在SqlSession级别存在 在数据库操作sqlsession时 在对象中一个数据结构HashMap用于储存数据 不用的sqlsession之间的一级缓存互相不影响

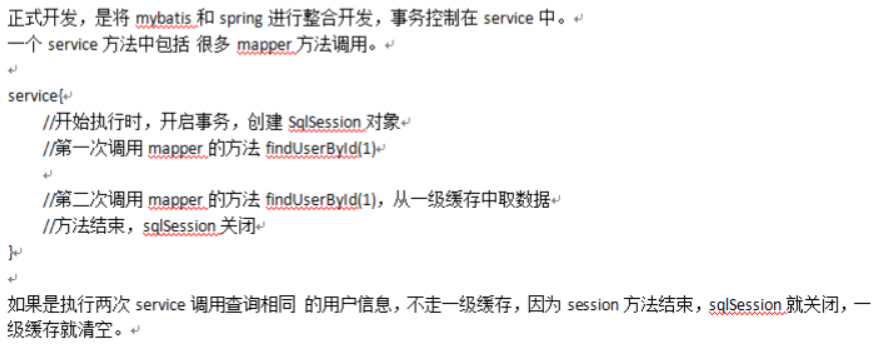
二级缓存：

二级缓存时mapper级别的缓存 不同的SQL session去操作同一个mapper对象的同一个sql语句 多个sqlsession共享同一个二级缓存 二级缓存是跨mapper



在使用同一个sqlsession时 如果调用了commit方法修改了数据

那么会自动清空之前的一级缓存 防止脏读

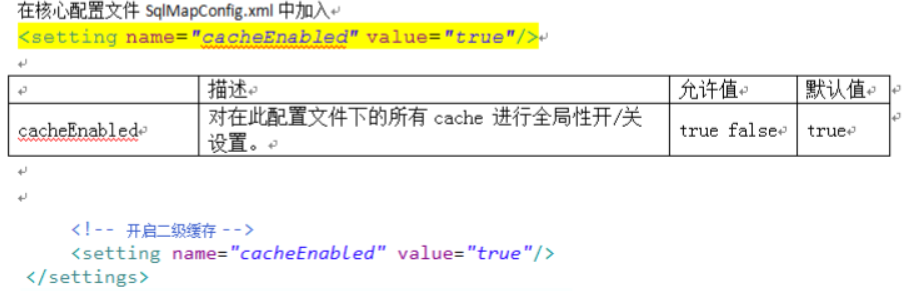


由于sqlsession线程不安全只能定义在方法体之中 所以只能在同一个service中使用一级缓存

二级缓存：（按照namespace划分）

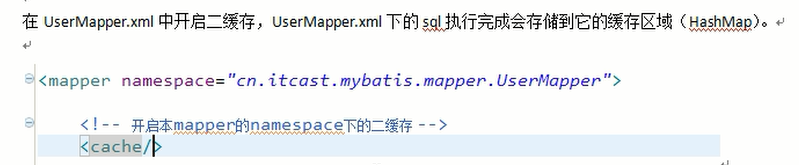
二级缓存时mapper级别 只要访问同一个mapper下的相同sql语句 就可以使用二级缓存

开区二级缓存 要在batis-config.xml 中 和 具体的mapper.xml 中开启二级缓存



在全局配置文件中 设置开启

在XXXMapper.xml中设置二级缓存开启(这样会存储到二级缓存区域中)



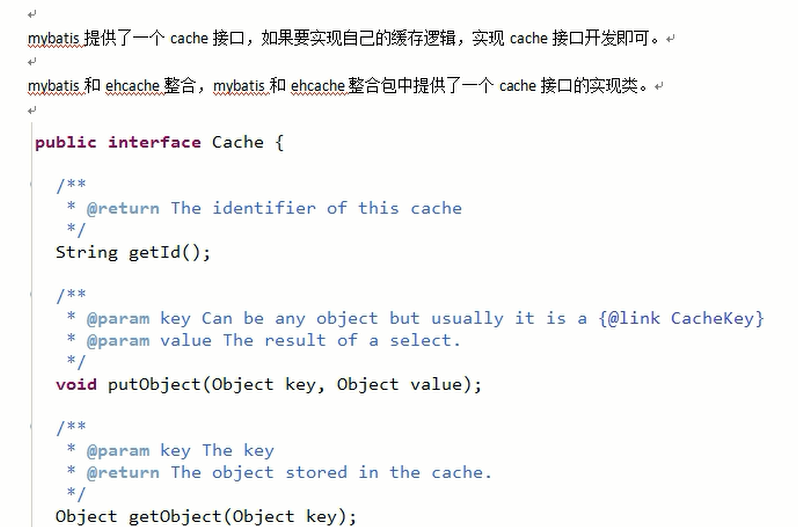
二级缓存如同一级缓存 在sqlsession执行commit之后 就会自动清除之前的二级缓存

\*\* XXXMappr.java 要实现序列化接口：因为二级缓存可能不储存在内存中而是硬盘里

### 分布式缓存：

默认二级缓存会存储在当前的服务器上 但是应用时应当把多个服务器上的缓存文件整合储存 需要使用redis形式的缓存模式

整合缓存：



在XXXMaper.xml 中的cache标签中使用type属性更改 缓存模式

### 二级缓存的应用场景;

对于访问请求多的实时性不高的可以使用二级缓存

实时性要求高的 意义不大（因为经常要commit 需要清除）

查询比较耗时的sql需要使用

二级缓存的局限性：

细粒度不要使用

# spring mybatis的整合

