1. 简介
2. Mybatis概念
3. 优点
4. 缺点

二、MyBatis的框架设计

1. 接口层---和数据库交互的方式
2. 数据处理层
3. 框架支撑层
4. 引导层

三、MyBatis的主要构件及其相互关系

1. Mybatis主要的核心部件
2. 组件关系

四、SqlSession 的工作过程分析

1. 加载配置
2. 开启会话对象
3. 调用执行器Executor
4. StatementHandler访问数据库

五、缓存机制

1. 缓存简介
2. 一级缓存工作机制
3. 二级缓存工作机制

六、事务管理机制

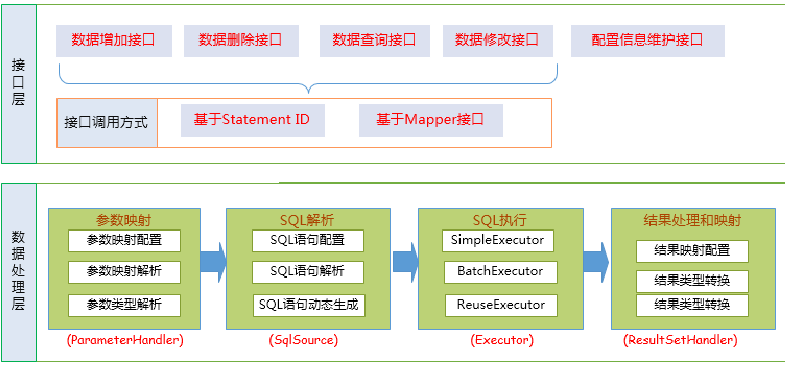
七、连接池管理机制

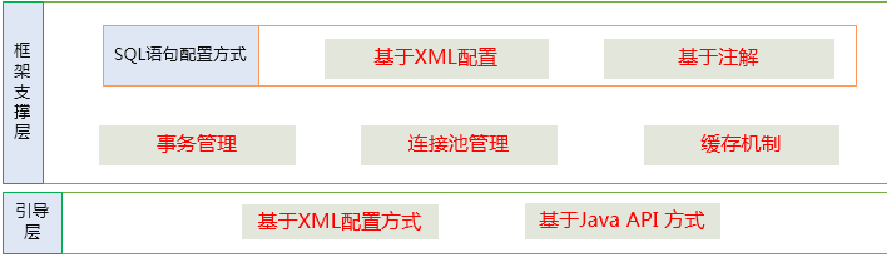
博客地址：http://blog.csdn.net/u010349169/article/category/2363143

1. 简介
2. Mybatis概念

MyBatis 是支持**定制化 SQL**、**存储过程**以及**高级映射**的优秀的持久层框架。MyBatis 避免了几乎所有的 JDBC 代码和手工设置参数以及抽取结果集。MyBatis 使用简单的 XML 或注解来配置和映射基本体，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。

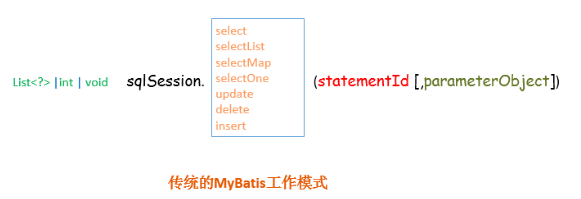
1. 优点
2. 框架精简。两个jar文件加上configuration配置文件，以及SQL映射文件，没有第三方依赖。
3. 灵活。mybatis不会对应用程序或者数据库的现有设计强加任何影响。 sql写在xml里，便于统一管理和优化。
4. 解除sql与程序代码的耦合，将业务逻辑和数据访问逻辑分离。
5. 提供映射标签，支持对象与数据库的orm字段关系映射。提供对象关系映射标签，支持对象关系组建维护。
6. 提供xml标签，支持编写动态sql。
7. 缺点
8. 编写SQL语句时工作量很大，尤其是字段多、关联表多时，更是如此。
9. SQL语句依赖于数据库，导致数据库移植性差，不能更换数据库。
10. 二级缓存机制不佳
11. Mybatis的框架设计 -





1. 接口层---和数据库交互的方式 -- 两种
2. 使用传统的MyBatis提供的API

通过传递Statement Id 和查询参数给 SqlSession 对象，使用 SqlSession对象完成和数据库的交互。



1. 使用Mapper接口
2. **对应关系**：

接口类名 对应 <mapper>的属性namespace

方法名 对应 SQL语句的id名

方法入参 对应 parameter属性

方法返回类型 对应 resultType/resultMap属性

1. **实现过程 -- 要点：动态机制的实现**

根据MyBatis 的配置规范配置好后，通过SqlSession.getMapper(XXXMapper.class) 方法，MyBatis 会根据相应的接口声明的方法信息，通过动态代理机制生成一个Mapper 实例。使用Mapper 接口的某一个方法时，MyBatis 会根据这个方法的方法名和参数类型，确定Statement Id，底层还是通过SqlSession.select("statementId",parameterObject);或者SqlSession.update("statementId",parameterObject); 等等来实现对数据库的操作。

1. 数据处理层 -- 核心层
2. 参数映射和动态SQL语句生成
3. MyBatis 通过传入的参数值，使用 Ognl 来动态地构造SQL语句，使得MyBatis 有很强的灵活性和扩展性。
4. **参数映射**指的是对于java 数据类型和jdbc数据类型之间的转换。这里有包括两个过程：查询阶段，我们要将java类型的数据，转换成jdbc类型的数据，通过 preparedStatement.setXXX() 来设值；另一个就是对resultset查询结果集的jdbcType 数据转换成java 数据类型。
5. SQL语句的执行以及封装查询结果集成List<E>

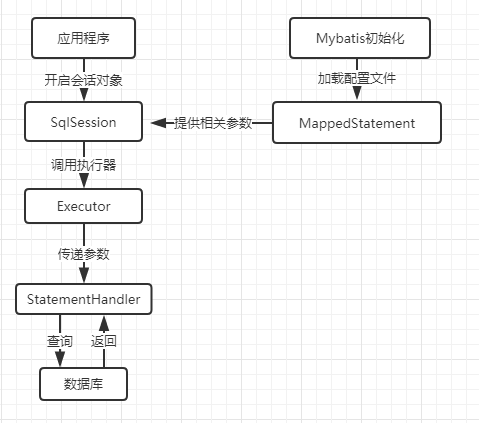
动态SQL语句生成之后，MyBatis 将执行SQL语句，并将可能返回的结果集转换成List<E> 列表。MyBatis 在对结果集的处理中，支持结果集关系一对多和多对一的转换，并且有两种支持方式，一种为嵌套查询语句的查询，还有一种是嵌套结果集的查询。

比如：resultType是一个对象，但是返回结果集就是一个List<对象名>或者是单个对象。

1. 框架支撑层
2. 事务管理机制 --见文末
3. 连接池管理机制 --见文末
4. 缓存机制 -- 见文末
5. SQL语句的配置方式
6. 基于XML配置SQL语句
7. 基于注解配置SQL语句 -- 建立在接口Mapper的使用上，传统的API不能使用注解。
8. 引导层

引导层是配置和启动MyBatis 配置信息的方式。MyBatis 提供两种方式来引导MyBatis ：基于XML配置文件的方式和基于Java API 的方式

1. MyBatis的主要构件及其相互关系
2. Mybatis主要的核心部件 -- 不是全部
3. SqlSession 作为MyBatis工作的主要顶层API，表示和数据库交互的会话，完成必要数据库增删改查功能。
4. Executor MyBatis执行器，是MyBatis 调度的核心，负责SQL语句的生成和查询缓存的维护。
5. StatementHandler 封装了JDBC Statement操作，负责对JDBC statement 的操作，如设置参数、将Statement结果集转换成List集合。
6. ParameterHandler 负责对用户传递的参数转换成JDBC Statement 所需要的参数。
7. ResultSetHandler 负责将JDBC返回的ResultSet结果集对象转换成List类型的集合。
8. TypeHandler 负责java数据类型和jdbc数据类型之间的映射和转换。
9. MappedStatement MappedStatement维护了一条<select|update|delete|insert>节点的封装。
10. SqlSource 负责根据用户传递的parameterObject，动态地生成SQL语句，将信息封装到BoundSql对象中，并返回。
11. BoundSql 表示动态生成的SQL语句以及相应的参数信息。
12. Configuration MyBatis所有的配置信息都维持在Configuration对象之中。
13. 组件关系 -- 见组件关系图
14. SqlSession 的工作过程分析



1. 加载配置信息 -- 生成一个MappedStatement对象
2. MyBatis在初始化的时候，会将MyBatis的配置信息全部加载到内存中，使用

org.apache.ibatis.session.Configuration实例来维护。使用者可以使用

sqlSession.getConfiguration()方法来获取。

1. 加载到内存中会生成一个对应的MappedStatement对象，然后会以

key="com.louis.mybatis.dao.EmployeesMapper.selectByMinSalary" ，value为

MappedStatement对象的形式维护到Configuration的一个Map中。当以后需要使用的时候，只需要通过Id值来获取就可以了。

1. 配置来源于两个地方，一处是配置文件，一处是Java代码的注解，将SQL的配置信息加载成为一个个MappedStatement对象（包括了传入参数映射配置、执行的SQL语句、结果映射配置），存储在内存中。
2. 开启一个数据库访问会话---创建SqlSession对象
3. mybatis使用SqlSession对象封装了对数据库的会话访问，实现事务控制和数据查询。
4. 为SqlSession传递一个配置的Sql语句 的Statement Id和参数，然后返回结果
5. SqlSession根据Statement ID, 在mybatis配置对象Configuration中获取到对应的MappedStatement对象，然后调用mybatis执行器Executor来执行具体的操作。
6. Executor根据SqlSession传递的参数执行query()方法
7. 根据传递的参数，完成SQL语句的动态解析，生成BoundSql对象，供StatementHandler使用。
8. 为查询创建缓存，以提高性能。
9. 创建JDBC的Statement连接对象，传递给StatementHandler对象，返回List查询结果。
10. StatementHandler对象负责设置Statement对象中的查询参数、处理JDBC返回的resultSet，将resultSet加工为List 集合返回
11. 对于JDBC的PreparedStatement类型的对象，创建的过程中，我们使用的是SQL语句字符串会包含 若干个? 占位符，我们其后再对占位符进行设值。StatementHandler通过

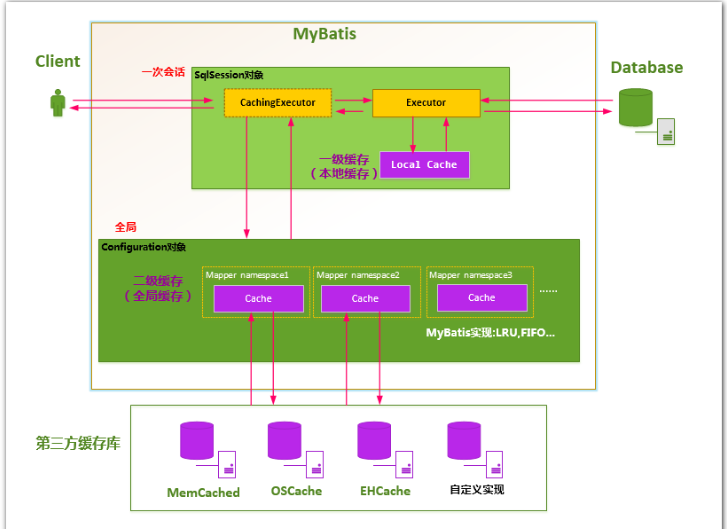
parameterize(statement)方法对Statement进行设值；

1. StatementHandler通过List<E> query(Statement statement, ResultHandler resultHandler)方法来完成执行Statement，和将Statement对象返回的resultSet封装成List；
2. 缓存机制
3. 缓存简介
4. 一级缓存

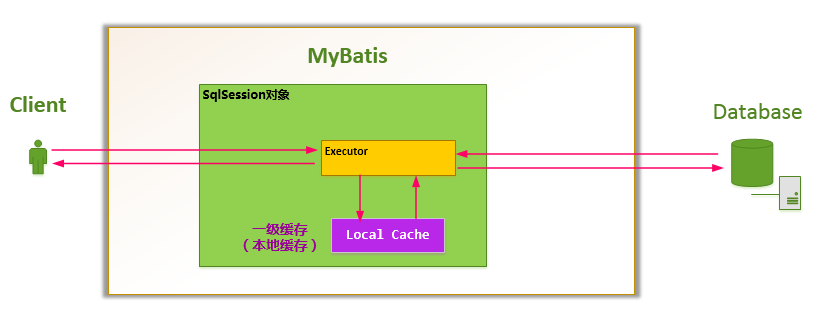
一级缓存是Session会话级别的缓存，位于表示一次数据库会话的SqlSession对象之中，又被称之为本地缓存。一级缓存是MyBatis内部实现的一个特性，用户不能配置，默认情况下自动支持的缓存，用户没有定制它的权利（不过这也不是绝对的，可以通过开发插件对它进行修改）。

1. 二级缓存

二级缓存是Application应用级别的缓存，它的是生命周期很长，跟Application的声明周期一样，也就是说它的作用范围是整个Application应用。



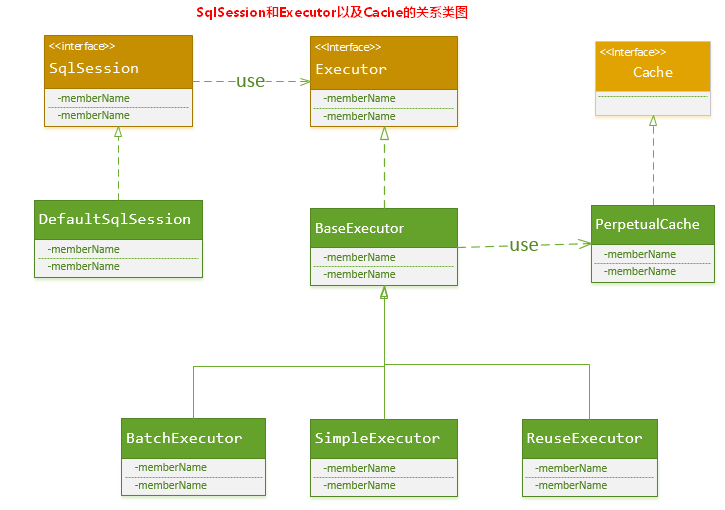
1. 一级缓存工作机制



1. 结构设计 -- SqlSession、Executor、Cache

实际上, MyBatis只是一个MyBatis对外的接口，SqlSession将它的工作交给了Executor执行器这个角色来完成，负责完成对数据库的各种操作。当创建了一个SqlSession对象时，MyBatis会为这个SqlSession对象创建一个新的Executor执行器，而缓存信息就被维护在这个Executor执行器中，MyBatis将缓存和对缓存相关的操作封装成了Cache接口中。

* Session级别的一级缓存实际上就是使用PerpetualCache维护



1. PerpetualCache对象 -- 作用是维护缓存

内部就是通过一个简单的HashMap<k,v> 来实现的，没有其他的任何限制。

1. 一级缓存的生命周期
2. MyBatis在开启一个数据库会话时，会创建一个新的SqlSession对象，SqlSession对象中会有一个新的Executor对象，Executor对象中持有一个新的PerpetualCache对象；当会话结束时，SqlSession对象及其内部的Executor对象还有PerpetualCache对象也一并释放掉。
3. 如果SqlSession调用了close()方法，会释放掉一级缓存PerpetualCache对象，一级缓存将不可用。
4. 如果SqlSession调用了clearCache()，会清空PerpetualCache对象中的数据，但是该对象仍可使用。
5. SqlSession中执行了任何一个update操作(update()、delete()、insert()) ，都会清空PerpetualCache对象的数据，但是该对象可以继续使用。
6. 一级缓存的工作流程
7. 对于某个查询，根据statementId,params,rowBounds来构建一个key值，根据这个key值去缓存Cache中取出对应的key值存储的缓存结果。
8. 判断从Cache中根据特定的key值取的数据数据是否为空，即是否命中；
9. 如果命中，则直接将缓存结果返回；

如果没命中：

3.1 去数据库中查询数据，得到查询结果；

3.2 将key和查询到的结果分别作为key,value对存储到Cache中；

3.3. 将查询结果返回；

1. 结束。
2. 性能分析
3. MyBatis对会话（Session）级别的一级缓存设计的比较简单，就简单地使用了HashMap来维护，并没有对HashMap的容量和大小进行限制。

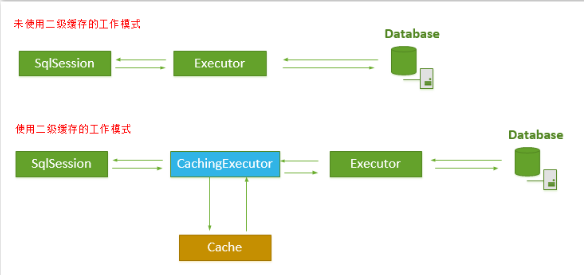
1.1 一般而言SqlSession的生存时间很短。一般情况下使用一个SqlSession对象执行的操作不会太多，执行完就会消亡；

1.2 对于某一个SqlSession对象而言，只要执行update操作（update、insert、delete），都会将这个SqlSession对象中对应的一级缓存清空掉，所以一般情况下不会出现缓存过大，影响JVM内存空间的问题；

1.3 可以手动地释放掉SqlSession对象中的缓存。

1. 一级缓存是一个粗粒度的缓存，没有更新缓存和缓存过期的概念
2. MyBatis认为的完全相同的查询，不是指使用sqlSession查询时传递给算起来Session的所有参数值完完全全相同，你只要保证statementId，rowBounds,最后生成的SQL语句，以及这个SQL语句所需要的参数完全一致就可以了。
3. 二级缓存工作机制
4. 工作模式

当开一个会话时，一个SqlSession对象会使用一个Executor对象来完成会话操作，MyBatis的二级缓存机制的关键就是对这个Executor对象做文章。如果用户配置了"cacheEnabled=true"，那么MyBatis在为SqlSession对象创建Executor对象时，会对Executor对象加上一个装饰者：CachingExecutor，这时SqlSession使用CachingExecutor对象来完成操作请求。CachingExecutor对于查询请求，会先判断该查询请求在Application级别的二级缓存中是否有缓存结果，如果有查询结果，则直接返回缓存结果；如果缓存中没有，再交给真正的Executor对象来完成查询操作，之后CachingExecutor会将真正Executor返回的查询结果放置到缓存中，然后在返回给用户。

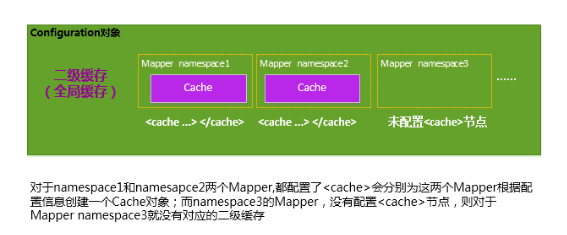


1. 二级缓存的划分

MyBatis并不是简单地对整个Application就只有一个Cache缓存对象，它将缓存划分的更细，即是Mapper级别的，即每一个Mapper都可以拥有一个Cache对象。

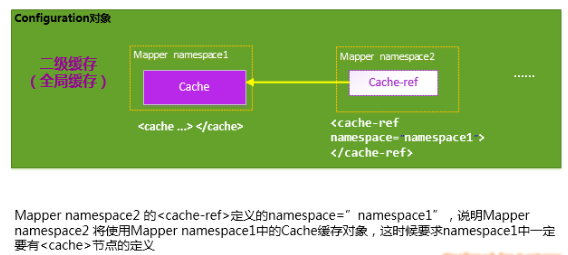
1. 为每一个Mapper分配一个Cache缓存对象（使用<cache>节点配置）

MyBatis将Application级别的二级缓存细分到Mapper级别，即对于每一个Mapper.xml,如果在其中使用了<cache> 节点，则MyBatis会为这个Mapper创建一个Cache缓存对象



1. 多个Mapper共用一个Cache缓存对象（使用<cache-ref>节点配置）

如果你想让多个Mapper公用一个Cache的话，你可以使用<cache-ref namespace="">节点，来指定你的这个Mapper使用到了哪一个Mapper的Cache缓存。



1. 使用二级缓存的条件 --- 详细到具体一条查询语句是否使用缓存
2. MyBatis支持二级缓存的总开关：全局配置变量参数 cacheEnabled=true
3. 该select语句所在的Mapper，配置了<cache> 或<cached-ref>节点，并且有效
4. 该select语句的参数 useCache=true

C:\Users\HP\AppData\Local\Temp\1511927905(1).png

1. 缓存使用的优先级

二级缓存 > 一级缓存 > 数据库

1. 二级缓存的实现方式
2. MyBatis自身提供的缓存实现；
3. 用户自定义的Cache接口实现；
4. 跟第三方内存缓存库的集成；