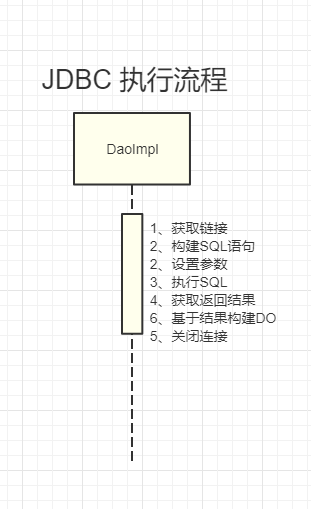
**myBatis 体系结构源码解读**

**课程概要：**

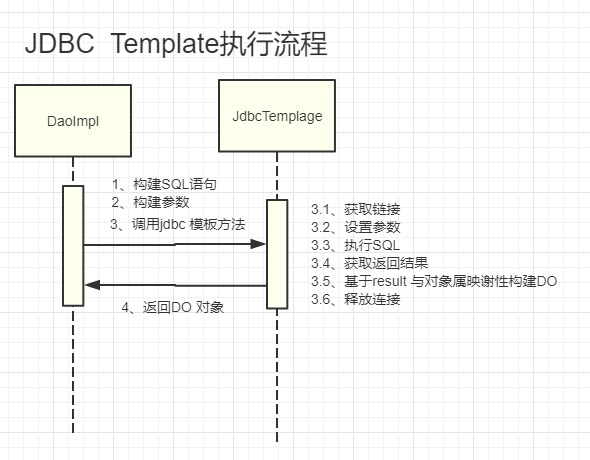
* 数据库访问层的4种实现方案对比
* myBatis整体架构介绍
* myBatis装载过程与执行过程

### **数据库访问层的4种实现方案对比**

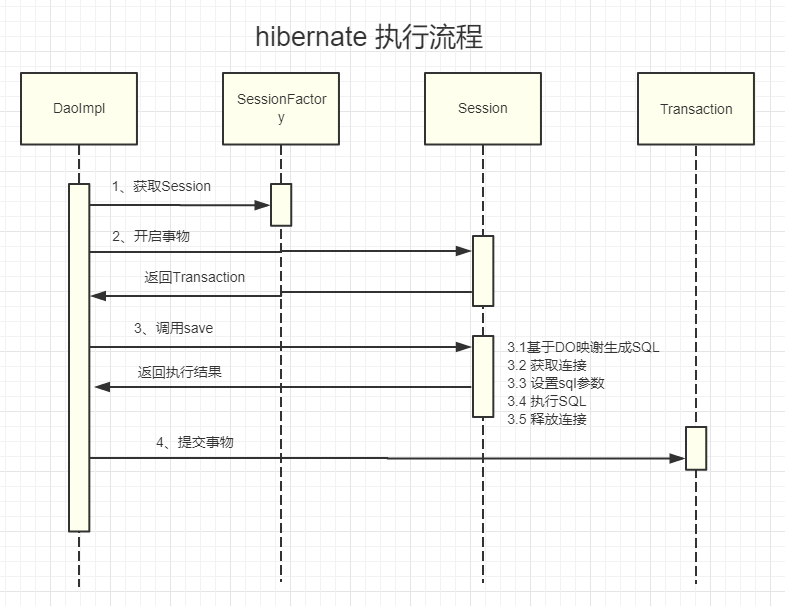
jdbc 执行时序图



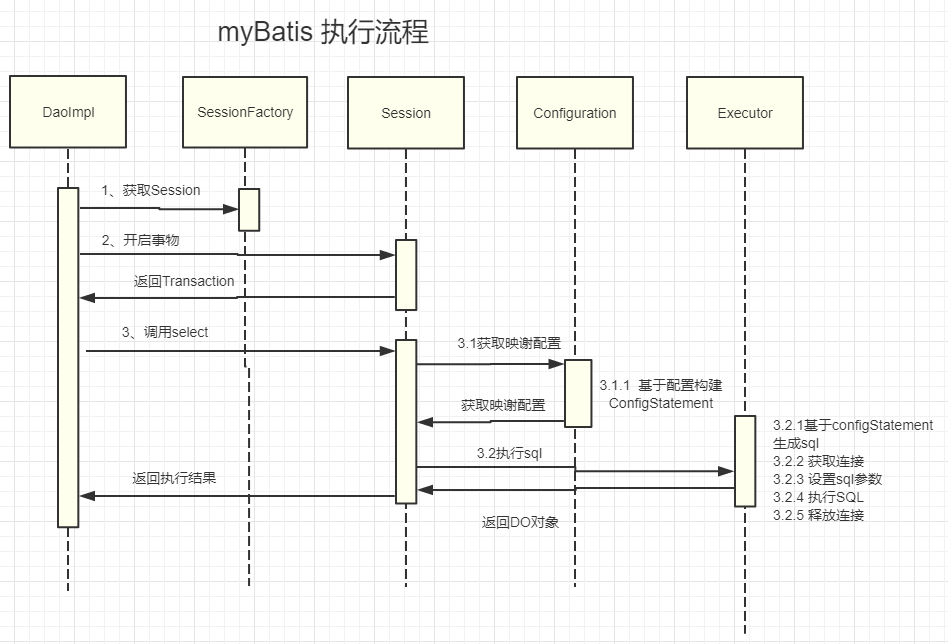
**JdbcTemplate 执行时序图**



**hibernate 执行流程**



**myBatis 执行流程**



**总结：**

|  |  |  |
| --- | --- | --- |
|  | 优点 | 缺点 |
| jdbc | 简单、纯粹 | 1、需要手动关闭链接 2、结果集不能自动映谢 |
| jdbcTemplate | 简单、纯粹、自动会话管理、结果集映谢 | 1、手动拼装SQL管理混乱 |
| hirbernate | 编程效率高，无需编写sql。数据库更换成本低、较完善的二级缓存、自动防SQL注入 | 完全掌握的门槛高、性能优化较麻烦、复杂映谢 |
| myBatis | 学习成本低、可以进行更为细致的SQL优化，减少查询字段、统一的SQL管理 | 功能相对简陋、需要手动编写维护SQL、表结构变更之后需要手动维护SQL与映谢 |

**myBatis 的定位**

myBatis 专注于sql 本身，其为sql 映谢而非完整的ORM，需要自己编写sql 语句，这是其优点也是缺点。优点是：优化方便，可更好利用sql编写经验。缺点是当数据修改之后调整麻烦耗费时间长.

**试用场景**：适用于对性能要求较高，有大批量的查询修改，并且业务实现没有过多依懒数据关系模型，比如：电商、O2O等互联网项目。

**互联网项目对DAO层的要求:**

1、对数据库的访问更新纯粹

2、尽可能不要使用数据库做运算

3、SQL语句可以针对性的优化(减少查询字段、查条件排序例 、查询条件尽可能命中索引)

### myBatis整体架构介绍

**编写一个 myBatis 示例**

新建myBatis-config.xml 文件

新建UserInfo.xml 文件

编写测试用例

示例代码执行流程：

1、配置加载

2、会话创建

3、Sql解析

4、SQL执行

5、结果映谢

**myBatis 体系结构图**



1、配置加载

画出配置加载相关UML图

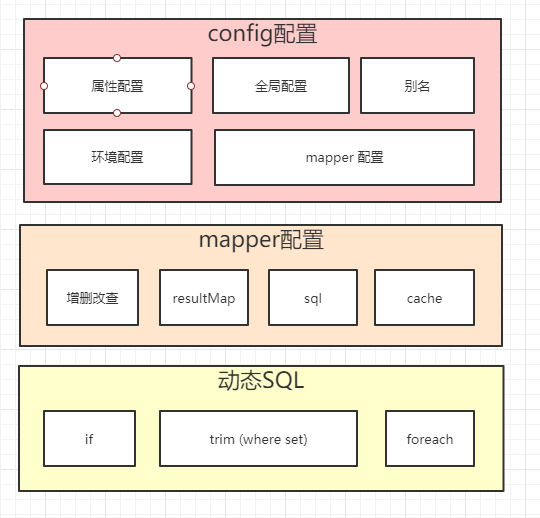
画出配置加载时序图

## **myBatis 应用与配置**

**课程该要：**

* config 上下文配置
* mapper 映谢文件配置
* 动态SQL配置

**myBatis 应用知识结构图：**



### **Config 上下文配置**

**1、属性配置**

<**properties resource="app.properties"**>

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

</**properties**>

三种设置方式：

1. 构建sessionFactory 时传递 (优先级：高)
2. 基于resource 属性加载 或 url 加载 (优先级：中)
3. 基于 <propertite> 属性设置 (优先级：低)

**2、全局参数配置**

<settings>

<setting name="mapUnderscoreToCamelCase" value="true" />

</settings>

具体参考：http://www.mybatis.org/mybatis-3/zh/configuration.html#settings

**3、环境配置**

<**environments default="development"**>

<**environment id="development"**>

<**transactionManager type="JDBC"** />

<**dataSource type="POOLED"** >

<**property name="driver" value="${jdbc.driver}"** />

<**property name="url" value="${jdbc.url}"** />

<**property name="username" value="${jdbc.username}"** />

<**property name="password" value="${jdbc.password}"** />

</**dataSource**>

</**environment**>

</**environments**>

事物管理器：JDBC|MANAGED

**4、数据源:**

**unpooled** 普通连接，每次获取时都会重新建立一个新的连接.属性下如下：

* **driver** :数据库驱动类
* **url**: URL地址
* **username**:用户名。
* **password** :登录数据库的密码。

**pooled:** 连接池模式，所有连接从连接池当中获取，由连接池来来进行连接的建立与回收关于等操作，除支持unpooled属性外还支持属性如下:

* **poolMaximumActiveConnections** ： 最大活跃数，默认值：10
* **poolMaximumIdleConnections** ：最大空闲连接数
* **poolMaximumCheckoutTime** :获取连接超时等待最大（checked out）时间，默认值：20000 毫秒
* **poolTimeToWait** ： 单次获取连接 最大等待时间 默认：20000 毫秒（即 20 秒）。
* **poolMaximumLocalBadConnectionTolerance** 获取连接重试次数 默认：3
* **poolPingQuery** 用于检测连接是否断开的测试 语句
* **poolPingEnabled** 是否通过执行poolPingQuery 语句做检测，默认值：false。
* **poolPingConnectionsNotUsedFor** 连接检测间隔时间 ，默认60000。

**5 、typeAliases 别名配置**

<**typeAliases**>

<**typeAlias type="com.tuling.dao"**/>

<**typeAlias type="com.tuling.dao.UserInfo" alias="UserInfo"**/>

</**typeAliases**>

**6、mappers 文件引入**

<mappers>

<mapper resource="userInfo.xml" />

<package name="com.tuling.dao"/>

</mappers>

基于 mapper 引入指定资源文件: resource| url |class

基于package 引入:扫描指定包路径当下的url

mapper 映谢文件配置

mapper 常用元素

* select – 映射查询语
* insert – 映射插入语句
* update – 映射更新语句
* sql – 可被其他语句引用的可重用语句块。
* delete – 映射删除语句
* resultMap 用来描述如何从数据库结果集中来加载对象。
* cache – 给定命名空间的缓存配置。
* cache-ref – 其他命名空间缓存配置的引用。

### **1、<select** **> 查询标签**

其表示一个查询语句映谢，其简单示例如下：

示例

<select id="selectUser"

resultType="com.tuling.mybatis.test.UuserInfo">

select \* from user\_info where id = #{id}

</select>

**其支持属性如下:**

<select

id="selectUser" // statement id

parameterType="int" // 参数类型

resultType="hashmap"// 返回结果类别

resultMap="personResultMap" // 返回结果映谢

flushCache="false" //

useCache="true"

timeout="10000"

fetchSize="256"

statementType="PREPARED">

参数的引用的办法

#{id, mode=in, jdbcType=INT, jdbcTypeName=MY\_TYPE}

### **2、<insert> <update> <delete> 示例 标签**

<insert id="addUser"

parameterType="com.tuling.mybatis.test.UuserInfo">

INSERT INTO user\_info (user\_name,nick\_name,password) VALUES

(#{userName},#{nickName},#{password})

</insert>

<update id="updateUser"

parameterType="com.tuling.mybatis.test.UuserInfo" >

update user\_info set user\_name=#{userName} where id=#{id}

</update>

<delete id="deleteUser" parameterType="int">

DELETE from user\_info where id=#{id}

</delete>

属性说明：

<insert

id="insertUuser"

parameterType="com.tuling.mybatis.test.UuserInfo"

flushCache="true"

statementType="PREPARED"

keyProperty=""

keyColumn=""

useGeneratedKeys=""

timeout="20">

<update

id="updateUuser"

parameterType="com.tuling.mybatis.test.UuserInfo"

flushCache="true"

statementType="PREPARED"

timeout="20">

<delete

id="deleteUuser"

parameterType="com.tuling.mybatis.test.UuserInfo"

flushCache="true"

statementType="PREPARED"

timeout="20">

### **3、<sql> 标签**

将重复的sql 语句定文为一个字段

<**sql id="base\_colume"**> id,user\_name,nick\_name</**sql**>

可通过 <include > 进行引入 如：

<include refid="base\_colume"/>

### **4、<resultMap> 标签**

resultMap 是myBatis 对象的映谢

* 动态SQL配置

### **<if> 标签**

### **trim (where, set) 标签**

### **foreach 标签**