# 第三阶段

第三阶段和第四阶段影响工作做项目 ，第一第二阶段影响的是面试。

老师周三，周四，周五晚上一般都在

分清重要和非重要的知识，以实用为主，不是很重要的知识不要去重点学习。

day01_6.7 mybatis基础入门 day02_6.8 mybatis映射文件 day03_6.9 上午代理模式+了解源码 +　下午权限模块的dao实现 day04_6.10 spring的ioc day05_6.11 spring的aop（难点） day06_6.12 通过aop定义事务管理器 day07_6.14 springmvc基础 day08_6.15 springmvc高级 day09_6.16 ssm框架整合-----重要 day10_6.17 自习完成权限模式以及前端 day11_6.18 上午log4j日志 下午自习-------系统管理大模块必须完成 day12_6.19 redis缓存数据库------重要 day13_6.21 工具_maven项目管理工具 day14_6.22 工具_maven聚合工程_jjw组件 day15_6.23 工具_远程接口访问与定时任务和异步任务 day16_6.24 工具_短信和邮件发送 day17_6.25 工具_excel与Word文件生成 day18_6.26 工具_fastdfs文件存储服务器 day19_6.27 项目时间 0707_activiti工作流引擎 0708_springboot基础 0709_springboot高级 0710_mybatisplus框架 7月底结束 每天看部分面试题，提高面试的竞争力，提高表达能力

# day01_6.7 mybatis入门

## 1. Mybatis有什么用？

Mybatis是数据库的工具，对jdbc的轻量级的封装，属于半自动化orm框架（OOP+SQL）。

oop：面向对象编程 orm： 对象关系映射持久层框架，比如：mybatis,hibernate (执行效率太低，封装的太完善了，纯自动化，不用写sql语句了)

利用xml（注解可能会遇到bug）配置文件进行配置。

提高开发效率，必定会降低运行效率，因此有部分注重效率的公司仍使用jdbc原生的东西。

# 2. 50分钟入门案例

- 安装mybatis的运行环境，准备jar包，准备jdbc的驱动包
- 数据库环境，数据库表的创建，对应的实体类
- 创建mybatis的核心配置
- 创建实体类对应的映射文件，并把映射文件加入到配置文件中
- 进行测试（运行mybatis的步骤）
  - 通过ClassLoader类加载 类路径(src/resources)下的文件
    - Test.class.getClassLoader().getResourceAsStream("mybatis-config.xml"),建议使用本方式，通用
    - in = Resources.getresourceAsStream("mybatis-config.xml")
  - 运行mybatis的核心对象SqlSessionFactoryBuilder,来加载mybatis-config.xml进行解析到Configuration对象中（可以理解为DataSource）。SqlSessionFactory ssf = new SqlSessionFactoryBuilder().builder(in,"dev"); 第二个参数为数据库的环境
  - 数据库会话对象(HttpSession)（可以理解为Connection）sqlSession sqlSession = ssf.openSession();
  - 调用会话对象中的方法，进行操作（执行数据库查询操作）
    - sqlSession.selectOne("",1)第一个参数为： 第二个参数为主键
  - 关闭数据库的连接sqlSession.close()
- 总结
  - sqlSessionFactoryBuilder加载Mybatis-config文件，根据数据库连接信息创建连接池对象
  - 解析sysuser-mapper.xml 文件，把该文件中的select标签进行解析 aa.bb.selectUserByKey -----> MappedStatement
  - 执行selectOne方法，传入字符串，找到MappedStatement对象，传入参数，执行sql语句

# 3.Mybatis的核心对象

**Mybatis 运行sql语句，就只能接受一个参数，如果有多个参数，则自动封装为map，默认的key为0，1，可以使用注解的方式，将对应的属性值作为key进行封装@Param**

- **SqlSessionFactoryBuilder**:构建者模式的应用，用来做复杂对象的构建，用来解析配置文件和映射文件，使用build方法创建对象DefaultSqlSessionFactory(Configuration), 该对象生命周期很短暂，用完就会被销毁。

```
InputStream in = TestSelectByKey.class.getClassLoader().getResourceAsStream("mybatis-config.xml");// 获取配置文件的文件流对象
SqlSessionFactory ssf = new SqlSessionFactoryBuilder().build(in, "dev");
```

- **SqlSessionFactory**----->DefaultSqlSessionFactory(Configuration): 会话工厂对象，全局唯一的单例对象，因为该对象内部存在数据库连接池，有文件解析结果Map

`SqlSession sqlSession = ssf.openSession();`

- **SqlSession对象**----->DefaultSqlSession: sql会话对象，用来执行数据库操作，用完就会关闭，在每次访问数据库时都需要创建它，每个线程都有自己的SqlSession对象，因此该对象不是共享的，也不是线程安全的。调用insert,update,selectList,selectOne,delete等方法执行增、删、查、改等操作；调用getMapper(xxx.class)来实例化接口执行增删查改；注意：SqlSession会话可以执行多次sql语句，当关闭了SqlSession对象后，需要重新创建。
  - selectOne
  - selectList
  - insert
  - update
  - delete
  - getMapper

getMapper方法：

```
1. 创建dao接口，SysuserMapper()接口
2. xml映射文件mapper 中的namespace="" 中间写接口的全名称
3. 映射文件中mapper下的增删改查标签的id值与方法名完全一致
4. 测试，使用session.getMapper(.class) 参数为接口的类对象
  SysuserMapper mapper = session.getMapper(.class);
  Sysuser sysuser = mapper.selectById(1); 查询
```

# 4. mybatis的配置文件

配置文件的引入，需要按照一定的顺序

该顺序为：(properties?, settings?, typeAliases?, typeHandlers?, objectFactory?, objectWrapperFactory?, reflectorFactory?, plugins?, environments?, databaseIdProvider?, mappers?)中间可以跳过，但是前后顺序必须保持一致

1. 数据库连接的配置
   - 可以直接在配置文件中定义数据库的连接参数

```
<dataSource type="POOLED">
    <property name="driver" value="com.mysql.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://127.0.0.1:3306/crm?useSSL=true"/>
    <property name="username" value="root"/>
    <property name="password" value="root"/>
</dataSource>
```

   - 可以配置properties,通过引入外部的properties文件进行配置

```
<!--加载类路径下properties文件-->
<properties resource="jdbc.properties">
</properties>

<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>
注：配置文件的内容为：
jdbc.driver=com.mysql.jdbc.Driver
```

```
jdbc.url=jdbc:mysql://127.0.0.1:3306/crm?useUnicode=true&characterEncoding=utf8&useSSL=true&serverTimezone=UTC
jdbc.username=root
jdbc.password=root
```

2. settings: 修改mybatis的运行行为（一般项目中必须使用到该配置）

```
<!--修改mybatis运行的一些默认行为-->
    <settings>
        <!--开启日志 stdout_loggin：代表将日志文件输出到控制台中-->
        <setting name="logImpl" value="STDOUT_LOGGING"/>
        <!--开启驼峰命名映射-->
        <setting name="mapUnderscoreToCamelCase" value="true">
    </settings>
```

3. typeAlias:mybatis 类型别名的配置，==不同的包下也不能出现同名的类(项目中必用)

```
<typeAliases>
    <!--在创建Configuration对象,对指定包下的所有的类,做别名映射,把类名小写作为别名,映射文件中的resultType忽略大小写-->
    <package name="com.javasm"></package>
    <!--对单个的类使用别名-->
    <!--<typeAlias type="com.javasm.sys.entity.SysUser" alias="sysuser"></typeAlias>-->
</typeAliases>
```

4. environments:配置数据库环境信息


```
配置数据库环境
<environments default="dev">
  <!--在一个项目中可能有多个分库分表操作，后续有专门的mycat中间件做分库操作-->
  <!--第一个数据库环境-->
  <environment id="dev">
    <!--JDBC:开启事务-->
    <transactionManager type="JDBC">
</transactionManager>
    <!--UNPOOLED|POOLED|JNDI-->
    <!--PooledDataSource 一般都会使用数据库连接池，jndi一般用于和别的框架的配合使用-->
    <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>
```

5. mappers：配置映射文件路径，再创建Configuration对象时，解析映射文件中的select,insert,update,delete标签到Configuration中的mappedStatements集合中，格式如下：namespace.id----->MappedStatement

```
 <mappers>
     <mapper resource="com\javasm\sys\mapper\sysuser-mapper.xml"></mapper>
 </mappers>
```

# 5. mybatis映射文件

- mapper: namespace="aa.bb" 必须唯一
- select, 查询标签

```
<!--
parameterType:参数类型:实体类,简单类型,map;
resultType:结果类型:实体类,简单类型,map;永远不会时list,set
如果parameterType是简单类型,#{随便写}
如果parameterType是实体对象,#{属性名}
如果parameterType是map,#{key}
-->
<select id="唯一标识" parameterType="参数类型" resultType="结果类型">
```

- insert 插入数据的标签，执行insert语句,改标签没有resultType属性.

```
<!--添加操作-->
<!--sysuser表是自增主键,有需要要获取新增记录id则需要设置useGeneratedKeys="true",keyProperty="uid"-->
<insert id="addUser" parameterType="Sysuser" useGeneratedKeys="true" keyProperty="uid">
  insert into sysuser(uname, upwd, uphone, uwechat, uemail, create_by) values
  (#{uname}, #{upwd}, #{uphone}, #{uwechat}, #{uemail},#{createBy})
</insert>
```

- update 更新数据

- delete 删除数据

```
<!--mybatis底层根据标签名,决定底层执行PreparedStatement对象的不同方法,executeQuery,executeUpdate-->
<update id="updateUserByUid" parameterType="Sysuser">
  update sysuser set upwd=#{upwd},uphone=#{uphone} where uid=#{uid}
</update>

<delete id="delUser" parameterType="int">
    delete from sysuser where uid=#{uid}
</delete>
```

- 注意点
  - 注意sql语句中使用#{}
  - 注意#{写法}
  - 注意id不能重复

# 6. 多参数传递

1. 多个参数封装到实体类中

   #{写对象的属性名}，对象的属性一定需要有get方法

```
<select id="selectUsersByUnameAndPwd" parameterType="Sysuser" resultType="sysuser">
  select * from sysuser where uname=#{uname} and upwd=#{upwd}
</select>
```

2. 多个参数封装到Map对象中，重要

```
 <!--#{map的key}-->
<select id="selectUsersByUnameAndPwd2" parameterType="map" resultType="sysuser">
  select * from sysuser where uname=#{uname_key} and upwd=#{upwd_key}
</select>
```

3. 引入mapper接口后，可以在接口的实参加@Param注解，指定key的值

```
List<Sysuser> login(@Param("uname2") String uname, @Param("upwd2") String upwd);
  <select id="login" parameterType="map" resultType="Sysuser">
     select * from sysuser where uname=#{uname2} and upwd=#{upwd2}
  </select>
  <select id="login" parameterType="map" resultType="Sysuser">
     select * from sysuser where uname=#{0} and upwd=#{1}
  </select>
  <select id="login" parameterType="map" resultType="Sysuser">
     select * from sysuser where uname=#{param1} and upwd=#{param2}
  </select>
```

## 7. #{}与${}的区别

两者都是用来获取查询参数,可用于sql语句中.

- #{}表达式
  - mybatis内对#{}解析为？占位符，数据更加的安全，因此对于条件查询语句必须使用#{}，避免sql注入
  - 如果parameterType是简单类型的话,#{随便写}
  - 如果parameterType是实体类,#{类的成员变量名}
  - 如果parameterType是map,#{map的key}
- mybatis内对${}不解析?占位符,而是直接进行sql拼接,因此不适合做条件查询

```
<select id="selectUsers" parameterType="map" resultType="sysuser">
    select * from sysuser order by ${soreField} ${sorted}
</select>
```

# 8. 常见异常

看最下方的Caused by: 查看错误

1.别名重复,包下有同名的类

```
org.apache.ibatis.type.TypeException: The alias 'SysUser' is already mapped to the value 'com.javasm.utils.Sysuser'.
```

2.反射异常,#{写错}

```
ReflectionException: There is no getter for property named 'aa' in 'class com.javasm.sys.entity.Sysuser'
```

3.映射文件中id重复

```
java.lang.IllegalArgumentException: Mapped Statements collection already contains value for aa.bb.selectUsers2
```

4.绑定异常,#{写错}

```
org.apache.ibatis.binding.BindingException: Parameter 'a' not found. Available parameters are [0, 1, param1, param2]
```

5.绑定异常,MappedStatement不存在,(dao接口中的方法在映射文件中没有标签)

```
org.apache.ibatis.binding.BindingException: Invalid bound statement (not found): com.javasm.sys.mapper2.SysuserMapper.del
```

6.绑定异常,接口未注册(映射文件中的namespace没有对应接口的名称),可能没引入映射文件

```
org.apache.ibatis.binding.BindingException: Type interface com.javasm.sys.mapper2.SysuserMapper is not known to the MapperRegistry.
```

**注意:**

数据库的字段是 date.,datetime(now()),timestamp(CURRENT_TIMESTAMP) 在实体类中一般用string类型的属性

类名的所有字母小写作为别名，常用使用的属性的别名已经内嵌进去了，映射文件中的resultType忽略大小写

在数据库中不要使用a_name 类似的命名方式，因为实体类需要aName,使用get或者set获取属性值是 getaName setaName,此时无法正常使用set和get方法

总结:

三个核心对象的api.尤其是SqlSession中的getMapper方法.

配置文件中的settings,typeAlias必须会.

映射文件中全部必须会.

# 9. test测试的常用用法

- 注解@FixMethodOrder(MethodSorters.NAME_ASCENDING)，当需要执行多个测试方法时，按照指定的顺序进行执行，可以避免在数据库中生成多余的记录数。
- 注解@BeforeClass，一般用来初始化类中定义的静态属性（最先运行）**全局初始化方法**
- 注解@AfterClass，在整个运行完毕的时候执行，**全局销毁方法**，或者关闭对象（最后运行）
- 注解@Before 用来初始化普通的属性，**注解测试初始化方法**
- 注解@After，在该测试模块执行完毕后进行执行，**注解测试销毁方法**一般用来关闭对象的连接

```
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class TestCURD {
    private static SqlSessionFactory ssf =null;
    private  SqlSession session =null;
    /*beforeClass与afterClass定义的方法必须是static方法*/
    @BeforeClass
    public static void beforeClass()
    {
```

```
9           ssf= SSF.getFactory();
10      }
11
12      @AfterClass
13      public static void afterClass(){
14          ssf=null;
15      }
16
17      @Before
18      public void init(){
19          session= ssf.openSession();
20      }
21
22      @After
23      public void close(){
24          session.close();
25      }
26  /*定义测试方法*/
27      @Test
28      public void test1_addUser() throws IOException {
29          Sysuser u = new Sysuser();
30          u.setUname("huawei");
31          u.setUpwd("123123");
32          //受影响的行数
33          int rows = session.insert("aa.bb.addUser", u);
34          System.out.println(rows);
35          Integer uid = u.getUid();
36          System.out.println("uid:"+uid);
37          session.commit();
38          //set autocommit false
39      }
40  }
```