Ibatis讲义

讲师：陈伟俊

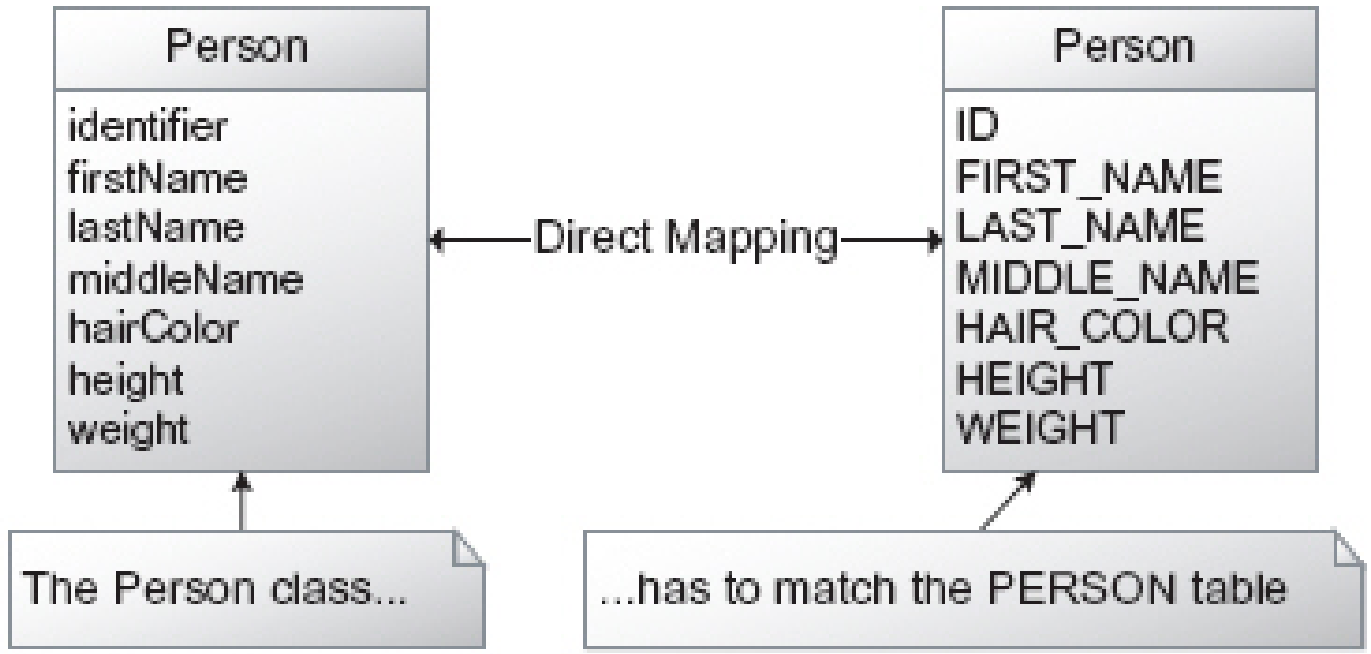
Ibatis概述

- iBATIS一词来源于“internet”和“abatis”的组合，是一个由Clinton Begin在2001年发起的开放源代码项目。最初侧重于密码软件的开发，现在是一个基于Java的持久层框架。
- 这里的“半自动化”，是相对Hibernate等提供了全面的数据库封装机制的“全自动化” ORM 实现而言，“全自动”ORM 实现了 POJO 和数据库表之间的映射，以及 SQL 的自动生成和执行。而ibatis 的着力点，则在于POJO 与 SQL之间的映射关系。也就是说，ibatis 并不会为程序员在运行期自动生成 SQL 执行。具体的 SQL 需要程序员编写，然后通过映射配置文件，将SQL所需的参数，以及返回的结果字段映射到指定 POJO。
- 注意：ibatis本是apache的一个开源项目，2010年这个项目由apache sofeware foundation 迁移到了google code，并且改名为mybatis

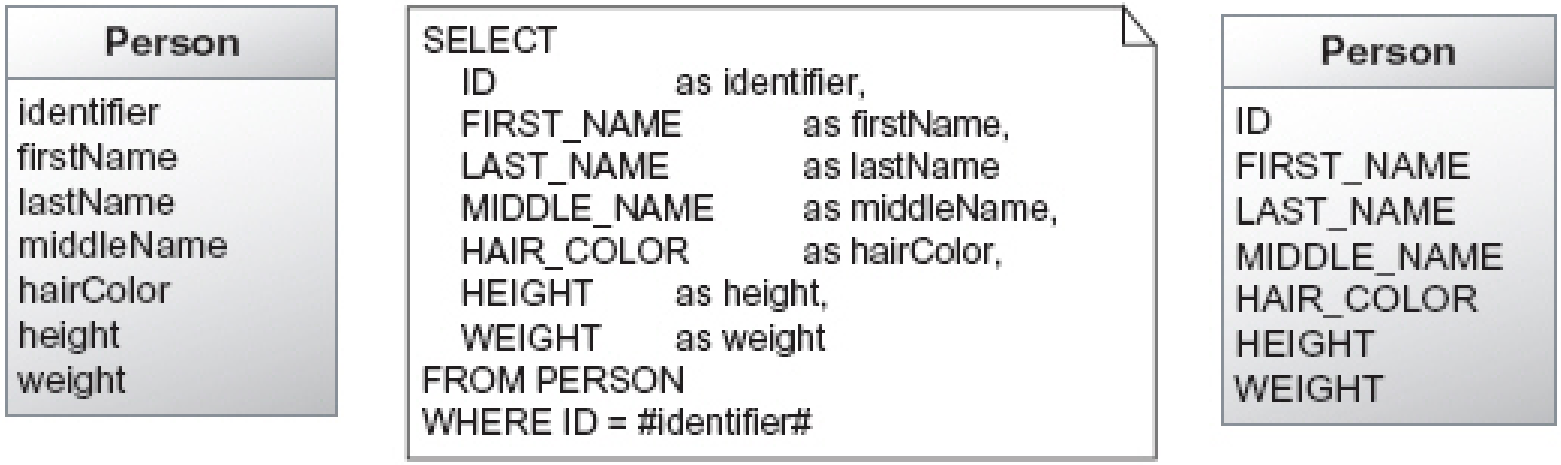
与传统的JDBC进行比较

- 减少了**61%**的代码量
- 最简单的持久化框架
- 架构级性能增强
- **SQL**代码从程序代码中彻底分离，可重用
- 增强了项目中的分工
- 增强了移植性

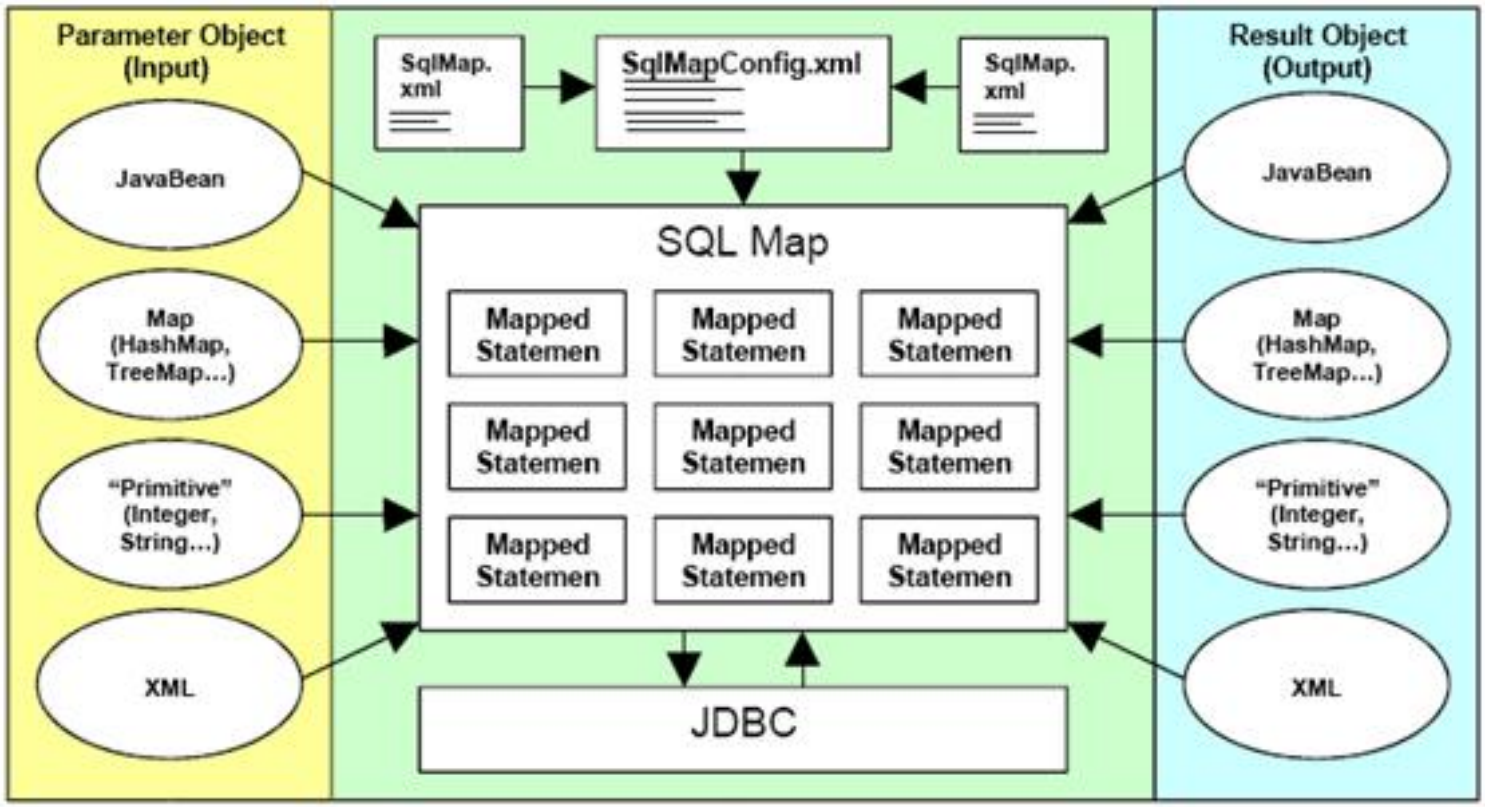
Hibernate映射关系



iBATIS映射关系


Person
identifier
firstName
lastName
middleName
hairColor
height
weight
SELECT
ID as identifier,
FIRST_NAME as firstName,
LAST_NAME as lastName
MIDDLE_NAME as middleName,
HAIR_COLOR as hairColor,
HEIGHT as height,
WEIGHT as weight
FROM PERSON
WHERE ID = #identifier#
Person
ID
FIRST_NAME
LAST_NAME
MIDDLE_NAME
HAIR_COLOR
HEIGHT
WEIGHT

工作流程


Parameter Object
(Input)
JavaBean
Map
(HashMap,
TreeMap...)
"Primitive"
(Integer,
String...)
XML
SqlMap.
xml
SqlMapConfig.xml
SqlMap.
xml
SQL Map
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
Mapped
Statemen
JDBC
Result Object
(Output)
JavaBean
Map
(HashMap,
TreeMap...)
"Primitive"
(Integer,
String...)
XML

搭配环境

- 新建工程
- 引入相关jar包
- 类路径下创建核心配置文件SqlMapConfig.xml

SqlMapConfig.xml示例一

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
   "http://www.ibatis.com/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
      <!-- 事务管理 -->
      <transactionManager type="JDBC">
        <!-- 数据源 -->
        <dataSource type="SIMPLE">
              <property value="oracle.jdbc.driver.OracleDriver" name="JDBC.Driver" />
              <property value="jdbc:oracle:thin:@localhost:1521:orcl" name="JDBC.ConnectionURL" />
              <property value="scott" name="JDBC.Username" />
              <property value="tiger" name="JDBC.Password" />
        </dataSource>
      </transactionManager>

      <!--映射文件路径，可能会有多个sqlMap映射文件-->
      <sqlMap resource="Dept.xml" />
</sqlMapConfig>
```

SqlMapConfig.xml示例二

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sqlMapConfig PUBLIC "-//iBATIS.com//DTD SQL Map Config 2.0//EN"
   "http://www.ibatis.com/dtd/sql-map-config-2.dtd">
<sqlMapConfig>
        <!-- 引用外部配置文件 -->
        <properties resource="dataSource.properties"/>

        <!-- 事务管理 -->
        <transactionManager type="JDBC">
                <!-- 数据源 -->
                <dataSource type="SIMPLE">
                <property value="${driver}" name="JDBC.Driver" />
                <property value="${url}" name="JDBC.ConnectionURL" />
                <property value="${username}" name="JDBC.Username" />
                <property value="${password}" name="JDBC.Password" />
                </dataSource>
        </transactionManager>

        <!--映射文件路径，可能会有多个sqlMap映射-->
        <sqlMap resource="Dept.xml" />
        <sqlMap resource="Emp.xml" />
</sqlMapConfig>
```

dataSource.properties

dataSource.properties

dataSource.properties

name	value
driver	oracle.jdbc.driver.OracleDriver
url	jdbc:oracle:thin:@localhost:1521:orcl
username	scott
password	tiger

对数据表做增删改查

- **<sqlmap>.xml**包含了我们将要运行的SQL语句。
- SqlMap的标签
  - ✓<select> 查询
  - ✓<insert> 插入数据
  - ✓<update> 更新数据库中信息
  - ✓<delete> 删除

SqlMapClient的CRUD方法

- queryForObject
- queryForList
- insert
- delete
- update

一个单表增删改查示例

- 对部门表进行增删改查操作

表结构

- 创建部门表和序列
- create table t_dept
  (
  dept_no number(6) primary key,
  dept_name varchar2(50),
  dept_loc varchar2(100)
  );

- create sequence t_dept_seq;

实体类

```
public class Dept {

    private int deptno;
    private String dname;
    private String loc;

    构造器略

    GET和SET略
}
```

类路径下Dept表的映射文件Dept.xml-1

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
  "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="DeptMap">

<!-- 给完整类名起一个别名-->
<typeAlias alias="Dept" type="com.icss.po.Dept" />

<!-- 类属性和表字段映射 -->
<resultMap class="Dept" id="DeptResult">
    <result property="deptno" column="dept_no"/>
    <result property="dname" column="dept_name"/>
    <result property="loc" column="dept_loc"/>
</resultMap>
```

类路径下Dept表的映射文件Dept.xml-2

```
<!-- 查询所有数据写法1：引用定义好的映射 -->
<select id="selecAlltDept" resultClass="Dept" resultMap="DeptResult">
select d.dept_no,d.dept_name,d.dept_loc from t_dept d
</select>

<!-- 查询所有数据写法2 ：直接用别名对应类属性-->
<select id="selecAlltDept2" resultClass="Dept">
select dept_no as deptno,dept_name as dname,dept_loc as loc from t_dept
</select>

<!-- 通过ID查询单条数据 -->
<select id="selectDeptById" parameterClass="int" resultClass="Dept"
    resultMap="DeptResult">
select * from t_dept where dept_no=#deptno#
</select>
```

类路径下Dept表的映射文件Dept.xml-3

```
<!-- 传入多个参数，用Map传递 -->
<select id="selectDeptByManyNo" parameterClass="java.util.Map" resultClass="Dept"
    resultMap="DeptResult">
select * from t_dept where dept_no between #beginNo# and #endNo#
</select>

<!-- 获得记录总数 -->
<select id="selectDeptCount" resultClass="int">
select count(*) from t_dept
</select>

<!-- 模糊查询 -->
<select id="selectDeptByName" parameterClass="String" resultClass="Dept"
    resultMap="DeptResult">
select * from t_dept where dept_name like '%$dname$%'
</select>
```

类路径下Dept表的映射文件Dept.xml-4

```
<!-- 插入一个实体 -->
<insert id="insertDept" parameterClass="Dept">
insert into t_dept values (#deptno#,#dname#,#loc#)
</insert>

<!-- 插入一个实体：利用序列生成主键 -->
<insert id="insertDeptBySequence" parameterClass="Dept">
insert into t_dept values (t_dept_seq.nextval,#dname#,#loc#)
</insert>
```

类路径下Dept表的映射文件Dept.xml-5

```
<!-- 更新一个实体 -->
<update id="updateDept" parameterClass="Dept">
update t_dept set dept_name=#dname#,dept_loc=#loc#
where dept_no=#deptno#
</update>

<!-- 删除一个实体 -->
<delete id="deleteDept" parameterClass="int">
delete from t_dept where dept_no=#deptno#
</delete>

</sqlMap>
```

#和$两种语法

- #可以进行预编译，进行类型匹配，#变量名# 会转化为jdbc 的 类型
- $不进行数据类型匹配，$变量名$就直接把 $name$替换为 name的内容
- 例如：
- select * from tablename where id = #id# ，假设id的值为12,其中如果数据库字段id为字符型，那么#id#表示的就是'12'，如果id为整型，那么#id#就是 12
- 会转化为jdbc的 select * from tablename where id=？，把？参数设置为id的值
- select * from tablename where id = id ，如果字段id为整型，Sql语句就不会出错，但是如果字段id为字符型，
- 那么Sql语句应该写成 select * from table where id = 'id'

#和$两种语法

- #方式能够很大程度防止sql注入.
- $方式无法防止sql注入.
- $方式一般用于传入数据库对象．例如传入表名.
- 所以ibatis用#比$好,一般能用#的就别用$.
- 另外,使用##可以指定参数对应数据库的类型
- 如:
- select * from tablename where id = #id:number#
- 在like 操作时候要特别注意不能用#
- 总结以下：
- $号使用在具体pojo类也就是非基本类型的取值，而#号使用在具体有基本类型的取值

Java类Dao代码-1

```
//核心SQL对象
private static SqlMapClient sqlMapClient = null;

static {
	try {
		//读取配置文件返回输入流
		Reader reader = Resources.getResourceAsReader("SqlMapconfig.xml");
		//初始化核心SQL对象
		sqlMapClient = SqlMapClientBuilder.buildSqlMapClient(reader);

		reader.close();
	} catch (IOException e) {
		e.printStackTrace();
	}
}
```

Java类Dao代码-2

```
//增加数据
public void add(Dept dept) throws SQLException {
        sqlMapClient.insert("insertDept", dept);
}

//增加数据，利用序列生成主键
public void addBySequence(Dept dept) throws SQLException {
        sqlMapClient.insert("insertDeptBySequence",dept);
}
```

Java类Dao代码-3

```
//修改数据
public void update(Dept dept) throws SQLException {
        sqlMapClient.update("updateDept",dept);
}

//删除数据
public void delete(int deptno) throws SQLException {
        sqlMapClient.delete("deleteDept", deptno);
}
```

Java类Dao代码-4

```
//查询所有数据
public List<Dept> query() throws SQLException {
        //传入的是在Dept.xml中select标签的id，返回查询结果
        List<Dept> list = sqlMapClient.queryForList("selecAlltDept");

        return list;
}

//通过ID查询单条数据
public Dept queryById(int deptno) throws SQLException {
        Dept dept = (Dept) sqlMapClient.queryForObject("selectDeptById", deptno);
        return dept;
}
```

Java类Dao代码-5

```
//通过开始和结束编号查询多条数据
public List<Dept> queryByManyNo(int beginNo, int endNo) throws Exception {
        //创建一个MAP集合来设置参数键值对
        Map map = new HashMap();
        map.put("beginNo", beginNo);
        map.put("endNo", endNo);

        List<Dept> list = sqlMapClient.queryForList("selectDeptByManyNo",map);

        return list;
}
```

Java类Dao代码-6

```
//返回记录总数
public int getCount() throws Exception {

        return (Integer) sqlMapClient.queryForObject("selectDeptCount");
}

//模糊查询
public List<Dept> queryByName(String dname) throws SQLException {
        List<Dept> list = sqlMapClient.queryForList("selectDeptByName",dname);
        return list;
}
```

<sqlMap>中的namespace

- <sqlMap namespace="DeptMap">中的namespace是名称空间设置，本例没有起到作用，其实是可以省略的，它真正的用处是当多个映射文件中的<select><insert><update><delete>的id重复时，可以通过名称空间加以区分，但是这项功能必须在核心配置文件中加入以下代码开启名称空间
- <!-- setttings为共通配置，以下是开启名称空间 -->
- <settings useStatementNamespaces="true"/>
- 那就意味着如果在DAO中引用SQL的语句ID，就要加入名称空间前缀，例如DeptMap. selecAlltDept

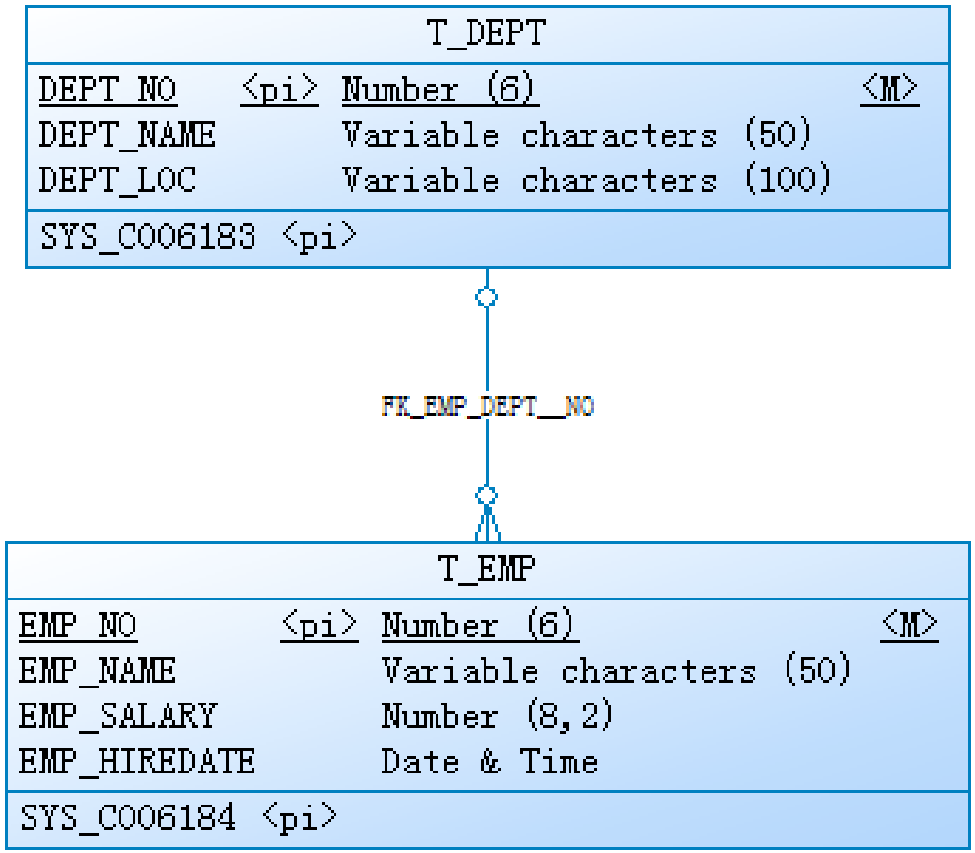
配置log4j.properties输出SQL语句

- 如果想在执行ibatis程序时，控制台输出执行的SQL语句，可以在导入log4j.jar和commons-logging.jar，
- 然后在类路径下创建log4j.properties文件
- 配置文件内容略

关联查询

- 创建关系表员工表，引用部门表部门编号
- create table t_emp
  (
  emp_no number(6) primary key,
  emp_name varchar2(50),
  emp_salary number(8,2),
  emp_hiredate date,
  dept_no number(6),
  constraints FK_EMP_DEPT__NO foreign key
  (dept_no)
  references t_dept (dept_no)
  );

关联查询：部门-员工概念模型



关联查询：Emp.java实体类

```
public class Emp {
    private int empNo;
    private String empName;
    private double empSalary;
    private Date empHiredate;
    private Dept dept;//持有一个Dept类的对象

        略……
}
```

关联查询：Emp映射文件：Emp.xml

```
<!-- 给完整类名起一个别名-->
<typeAlias alias="Emp" type="com.icss.po.Emp" />

<!-- 类属性和表字段映射 -->
<resultMap class="Emp" id="EmpResult">
        <result property="empNo" column="emp_no" />
        <result property="empName" column="emp_name" />
        <result property="empSalary" column="emp_salary" />
        <result property="empHiredate" column="emp_hiredate" />
        <!-- 以下是组合关系映射 -->
        <result property="dept.deptno" column="dept_no" />
        <result property="dept.dname" column="dept_name" />
</resultMap>
```

关联查询：Emp映射文件：Emp.xml 续

```
<!-- 查询所有员工 -->
<select id="selectAllEmp" resultClass="Emp" resultMap="EmpResult">
        select
        e.emp_no,e.emp_name,e.emp_salary,e.emp_hiredate,d.dept_no,d.
dept_name
        from t_emp e,t_dept d
        where e.dept_no=d.dept_no(+)
</select>

<!-- 插入一个新员工 -->
<insert id="insertEmp" parameterClass="Emp">
        insert into t_emp values
(#empNo#,#empName#,#empSalary#,#empHiredate#,#dept.deptno#)
</insert>
```

关联查询：测试代码

```
public class TestQueryEmp {
    public static void main(String[] args) throws Exception {
        Reader reader = Resources.getResourceAsReader("SqlMapConfig.xml");

        SqlMapClient sqlMapClient =
    SqlMapClientBuilder.buildSqlMapClient(reader);

        List<Emp> list = sqlMapClient.queryForList("selectAllEmp");

        for (Emp e : list) {
            System.out.println(e);
        }
    }
}
```

ibatis事务处理

- 如果不加事务处理，默认sqlMapClient对象执行增删改之后会自动提交事务，如果希望手动提交事务，则需要以下写法

```
sqlMapClient.startTransaction();//开始事务

//其他增删改操作……

sqlMapClient.commitTransaction();//提交事务
sqlMapClient.endTransaction();//结束事务，一定要写，否则记录不解锁
```

动态SQL语句

- 在应用中我们经常会做一些动态的拼接条件，但是如果是JDBC我们可以用程序拼接SQL语句，如果ibatis，我们可以使用动态SQL语句。
- 例如按照员工姓名和工资来搜索员工信息，如果如果姓名和工资的检索值为空，则忽略这个检索条件
- 一般来说，我们都会用where 1=1类似这种写法来实现，但是ibatis就需要动态语句实现
- 在Ibatis中，动态的条件元素包含一下几种：二元条件元素、一元条件元素和其他条件元素

二元条件元素

- 将一个属性值和静态值或另一个属性值比较，如果条件为真，元素将被包容在查询SQL语句中。
- **二元条件元素的属性：**
- perpend——可被覆盖的SQL语句组成部分，添加在语句的前面(可选)
- property——是比较的属性(必选)
- compareProperty——另一个用于和前者比较的属性(必选或选择compareValue)
- compareValue——用于比较的值(必选或选择compareProperty)

二元条件元素

- 二元条件元素标签

<isEqual>	比较属性值和静态值或另一个属性值是否相等。
<isNotEqual>	比较属性值和静态值或另一个属性值是否不相等。
<isGreaterThan>	比较属性值是否大于静态值或另一个属性值。
<isGreaterEqual>	比较属性值是否大于等于静态值或另一个属性值。
<isLessThan>	比较属性值是否小于静态值或另一个属性值。
<isLessEqual>	比较属性值是否小于等于静态值或另一个属性值。

二元条件元素

- 示例：如果传入empSalary大于0则加入emp_salary>#empSalary#这个条件语句

```
<select id="selectEmpByCondition2" parameterClass="Emp" resultMap="EmpResult2">
        select * from t_emp
        <dynamic prepend="where">
                <isGreaterThan prepend="and" property="empSalary" compareValue="0">
                        emp_salary>#empSalary#
                </isGreaterThan>
        </dynamic>
</select>
```

一元条件元素

- 一元条件元素检查属性的状态是否符合特定的条件
- 一元条件元素的属性：
- prepend——可被覆盖的SQL语句组成部分，添加在语句前面(可选)
- property——被比较的属性(必选)

一元条件元素

- 一元条件元素的标签

<isPropertyAvailable>	检查是否存在该属性（存在parameter bean的属性）
<isNotPropertyAvailable>	检查是否不存在该属性（不存在parameter bean的属性）
<isNull>	检查属性是否为null
<isNotNull>	检查属性是否不为null
<isEmpty>	检查Collection.size()的值，属性的String或String.valueOf()值,是否为null或空（“”或size() < 1）
<isNotEmpty>	检查Collection.size()的值，属性的String或String.valueOf()值,是否不为null或不为空（“”或size() > 0）

一元条件元素

- 示例：1如果empName不为空，加入emp_name like '%'|| #empName# || '%'，2如果empHiredate不为空加入emp_hiredate=#empHiredate#

```
<select id="selectEmpByCondition" parameterClass="Emp"
    resultMap="EmpResult2">
        select * from t_emp
        <dynamic prepend="where">
                <isNotEmpty prepend="and" property="empName">
                        emp_name like '%'|| #empName# || '%'
                </isNotEmpty>
                <isNotEmpty prepend="and" property="empHiredate">
                        emp_hiredate=#empHiredate#
                </isNotEmpty>
        </dynamic>
</select>
```

一元条件元素

测试代码：

```
Emp emp = new Emp();
emp.setEmpName("tom");
emp.setEmpHiredate(Date.valueOf("2012-11-17"));

List<Emp> list =
sqlMapClient.queryForList("selectEmpByCondition",emp);
```

其他元素条件

- Parameter Present：这些元素检查参数对象是否存在
- Parameter Present条件的属性
- prepend — 可被覆盖的SQL语句组成部分，添加在语句的前面（可选
- 元素的标签：

<isParameterPresent>	检查是否存在参数对象（不为null）
<isNotParameterPresent>	例子： <isNotParameterPresent prepend=”AND”> EMPLOYEE_TYPE = ‘DEFAULT’ </isNotParameterPresent>

其他元素条件

- 示例：如果没有传入Emp，则增加emp_no=3语句

```
<select id="selectEmpByCondition3" parameterClass="Emp"
resultMap="EmpResult2">
        select * from t_emp
        <dynamic prepend="where">
                <isNotParameterPresent>
                        emp_no=3
                </isNotParameterPresent>
        </dynamic>
</select>
```

其他元素条件

- **Iterate：这属性遍历整个集合，并为List集合中的元素重复元素体的内容。**
- Iterate的属性：
- prepend — 可被覆盖的SQL语句组成部分，添加在语句的前面（可选）
  property — 类型为java.util.List的用于遍历的元素（必选）
  open — 整个遍历内容体开始的字符串，用于定义括号（可选）
  close —整个遍历内容体结束的字符串，用于定义括号（可选）
  conjunction — 每次遍历内容之间的字符串，用于定义AND或OR（可选）

其他元素条件

- 元素的标签：

iterate>	遍历类型为java.util.List的元素。 例子： <iterate prepend="AND" property="userNameList" open="(" close=")" conjunction="OR"> username=#userNameList[]# </iterate> 注意：使用<iterate>时，在List元素名后面包括方括号[]非常重要，方括号[]将对象标记为List，以防解析器简单地将List输出成String。

其他元素条件

- 示例：传入一组编号，按照编号做in条件查询

```
<select id="selectEmpByCondition4"
parameterClass="java.util.List" resultMap="EmpResult2">
        select * from t_emp
        where emp_no in
        <iterate open="(" close=")" conjunction=",">
                #value[]#
        </iterate>
</select>
```

其他元素条件

- 测试：

```
List<Integer> condition = new ArrayList<Integer>();
condition.add(2);
condition.add(3);
condition.add(9);

List<Emp> list = sqlMapClient.queryForList("selectEmpByCondition4",condition);
```

<sql>和<include>

- 可以编写一些语句片段<sql>标签，然后在其他语句标签汇中用<include>引用

- 示例：

```
<!-- sql片段 -->
<sql id="sql_select">
    select *
 </sql>

<sql id="sql_count">
    select count(*)
</sql>
```

<sql>和<include>

```
<!-- 包含sql片段 -->
<select id="selectEmpAll" resultMap="EmpResult2">
        <include refid="sql_select" />
        from t_emp
</select>

<select id="selectEmpCount" resultClass="int">
        <include refid="sql_count" />
        from t_emp
</select>
```

数据分页

- ibatis的数据分页比较简单，例如下例就是获得从第11条数据开始的5条数据，注意索引从0开始

参数分别是：语句ID，传入参数，起始位置，偏移数量

```
List<Emp> list = sqlMapClient.queryForList("selectAllEmp",null,10,5);
```

- **但是需要注意的是**，此为前台分页，是先把数据全部查询出来之后再分页，大数据量容易造成内存奔溃！

数据分页

- 推荐使用SQL语句本身进行数据分页，如下例：

```
<!-- 查询所有员工（带分页）-->
<select id="selectPageEmp" resultClass="Emp"
    resultMap="EmpResult2" parameterClass="java.util.Map">
    select *  from (select rownum rnum, e.* from t_emp e)
    where rnum between 10 and 15
</select>
```

数据分页

- 查询测试：

```
//查询11到15条
Map map = new HashMap();
map.put("startNum", 10);
map.put("endNum", 15);

List<Emp> list = sqlMapClient.queryForList("selectPageEmp",map);

for (Emp e : list) {
        System.out.println(e);
}
```

Spring整合ibatis代码片段-1

```
<!-- 数据源 -->
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close" scope="singleton">
        <property name="driverClassName" value="oracle.jdbc.driver.OracleDriver"
/>
        <property name="url" value="jdbc:oracle:thin:@localhost:1521:xe" />
        <property name="username" value="scott" />
        <property name="password" value="tiger" />
</bean>
```

Spring整合ibatis代码片段-2

```
<!-- 通过Spring获得sqlMapClient -->
<bean id="sqlMapClient"
class="org.springframework.orm.ibatis.SqlMapClientFactoryBean">
        <property name="dataSource">
                <ref local="dataSource" />
        </property>
        <property name="configLocation" value="classpath:SqlMapConfig.xml" />
</bean>
```

Spring整合ibatis代码片段-3

```
<!-- 配置事务管理器 -->
<bean id="transactionManager”
        class="org.springframework.jdbc.datasource.DataSourceTransaction
Manager">
        <property name="dataSource" ref="dataSource" />
</bean>

<!-- 启用注释功能 -->
<tx:annotation-driven transaction-manager="transactionManager" />
```

Spring整合ibatis代码片段-4

```
<!-- DetpDao -->
<bean id="deptDao" class="com.cwj.dao.DeptDaoImpl">
        <property name="sqlMapClient" ref="sqlMapClient" />
</bean>

<!-- DeptService配置事务切入  -->
<bean id="deptService" class="com.cwj.service.DeptServiceImpl">
        <property name="dao" ref="deptDao" />
</bean>
```

Spring整合ibatis代码片段-5

```
@Transactional(rollbackFor=Exception.class)
public class DeptServiceImpl implements IDeptService{

    service方法……

}
```

Spring整合ibatis方式二代码片段

```
<!-- DetpDao需要注入sqlMapClient和dataSource -->
    <bean id="deptDao" class="com.cwj.dao.DeptDaoImpl">
        <property name="sqlMapClient" ref="sqlMapClient" />
        <property name="dataSource" ref="dataSource"/>
    </bean>
```

Spring整合ibatis方式二代码片段

```
public class DeptDaoImpl extends SqlMapClientDaoSupport implements IDeptDao {

    //增加数据
    public void add(Dept dept) throws SQLException {
        this.getSqlMapClientTemplate().insert("insertDeptBySequence", dept);

    }

    //查询所有数据
    public List<Dept> query() throws SQLException {

        return this.getSqlMapClientTemplate().queryForList("selecAlltDept");

    }

}
```