参数化

> sql命令拼接困难 拼接sql指令的操作 有 **sql注入**的风险 参数化就可以解决这些问题

操作步骤

1. 重设sql命令为带参sql命令
2. 选择Statement(通讯对象)的子接口对象 替换原对象 PreparedStatement(有准备能力的通讯对象)

```
public class BaseDao{
    //省略
    //2. 选择Statement(通讯对象)的子接口对象 替换原对象 PreparedStatement(有准备能力的通讯对象)
    protected PreparedStatement sta=null;
    //省略
}
```

3. 连接对象创建通讯对象的方法 要选择 conn.prepareStatement(sql) sql命令提前赋予对象
4. 执行操作前 需要为sql命令中包含的参数供值
5. 执行方法选择 无参方法 sql命令已经提前设置好了

```
public int add(Department department) {
    //1.重设sql命令为带参sql命令
    String sql="insert into department values(?,?,?);";
    int count=0;//受影响的行数
    try {
        //开启了连接
        super.open_db();
        //创建有准备能力的通讯对象
        //3.连接对象创建通讯对象的方法 要选择 conn.prepareStatement(sql) sql命令提前赋予对象
        super.sta=super.conn.prepareStatement(sql);
        //4.sql命令中只要带有参数 就要在执行前为参数供值
        super.sta.setInt(1,department.getDeptId());
        super.sta.setString(2,department.getDeptName());
        super.sta.setString(3,department.getDeptAddress());
        //5.执行方法选择 无参方法 sql命令已经提前设置好了
        count=super.sta.executeUpdate();
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        //回收所有开启过的对象
        super.close_db();
    }
    return count;
}
```

通用查询

> 查询数据的筛选条件是多样的 组合复杂 如果每一个方法只应对一种查询条件的组合 方法的数量将极为庞大 所以需要设计一个具有普适性的查询方法

操作步骤

1. 实现动态sql

```
System.out.println("请输入要查找的部门编号:(*代表不输入)");
String id=input.next();
if (!id.equals("*")){
    //用户输入的编号信息
    pars.put("deptId",id);
}
System.out.println("请输入要查找的部门名称:(*代表不输入)");
String name=input.next();
if (!name.equals("*")){
    //用户输入的编号信息
    pars.put("deptName",name);
}

System.out.println("请输入要查找的部门地址:(*代表不输入)");
String address=input.next();
if (!address.equals("*")){
    //用户输入的编号信息
    pars.put("deptAddress",address);
}
ddao.query(pars);
```

```
@Override
public List<Department> query(Map<String,Object> pars) {
    //组建sql命令 where 1=1 消耗了 where子句关键字 后续条件均可使用and拼接
    StringBuilder sql=new StringBuilder("select *  from department  where 1=1
");
    if (pars.containsKey("deptId")){
        sql.append(" and deptId=? ");
    }
    if (pars.containsKey("deptName")){
        sql.append(" and deptName like ? ");
    }
    if (pars.containsKey("deptAddress")){
        sql.append(" and deptAddress like ? ");
    }
    sql.append(";");
    //***尚未完成的内容****
    return null;
}
```

2. 为sql命令中的参数供值

> 额外创建一个有序集合 用于维持 参数 和 参数值顺序的一致性 Map集合中的键集合 和值集合都是无序的

> 选择的循环方式 是for循环 为了更好的配合参数索引 和集合下标进行使用

```
@Override
public List<Department> query(Map<String,Object> pars) {
    //增加一个集合  该集合用于维持sql参数  和 参数值顺序的一致
    List<Object> ps=new ArrayList<>();
    //组建sql命令
    StringBuilder sql=new StringBuilder("select *  from department  where 1=1
");
    if (pars.containsKey("deptId")){
        sql.append(" and deptId=? ");
        ps.add(pars.get("deptId"));
    }
    if (pars.containsKey("deptName")){
        sql.append(" and deptName like ? ");
        ps.add("%"+pars.get("deptName")+"%");
    }
    if (pars.containsKey("deptAddress")){
        sql.append(" and deptAddress = ? ");
        //当一个条件被拼接到sql命令中时  同时 将该条件参数对应的数据值 添加到ArrayList
集合中 保持顺序
        ps.add(pars.get("deptAddress"));
    }
    sql.append(";");
    try {
        super.open_db();
        //创建有准备能力的通讯对象
        super.sta=super.conn.prepareStatement(sql.toString());
        //遍历键值对集合的值集合 为参数供值
        for (int i = 0; i <ps.size() ; i++) {
            //不同的参数 类型不同  难以判定该使用哪一个特定类型的set方法
            super.sta.setObject(i+1,ps.get(i));
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        super.close_db();
    }
    return null;
}
```

3. 完成查询结果集的解析 并返回集合

```
@Override
public List<Department> query(Map<String,Object> pars) {
    //增加一个集合  该集合用于维持sql参数  和 参数值顺序的一致
    List<Object> ps=new ArrayList<>();
    //设置一个集合 用于存储解析结果集产生的对象
    List<Department> depts=new ArrayList<>();
    //组建sql命令
    StringBuilder sql=new StringBuilder("select *  from department  where 1=1
");
    if (pars.containsKey("deptId")){
        sql.append(" and deptId=? ");
        ps.add(pars.get("deptId"));
```

```
    }
    if (pars.containsKey("deptName")){
        sql.append(" and deptName like ? ");//1
        ps.add("%"+pars.get("deptName")+"%");//0
    }
    if (pars.containsKey("deptAddress")){
        sql.append(" and deptAddress = ? ");//3
        ps.add(pars.get("deptAddress"));//2
    }
    sql.append(";");
    System.out.println(sql);
    try {
        super.open_db();
        //创建有准备能力的通讯对象
        super.sta=super.conn.prepareStatement(sql.toString());
        //遍历键值对集合的值集合 为参数供值
        for (int i = 0; i <ps.size() ; i++) {
            super.sta.setObject(i+1,ps.get(i));
        }
        //执行命令 得到结果集 并对结果进行解析
        super.rs=super.sta.executeQuery();
        while (super.rs.next()){
            Department dept=new Department();
            dept.setDeptId(super.rs.getInt(1));//deptId
            dept.setDeptName(super.rs.getString(2));//deptName
            dept.setDeptAddress(super.rs.getString(3));//deptAddress
            //组装好的对象存入集合中
            depts.add(dept);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    } finally {
        super.close_db();
    }
    return depts;
}
```

通查综合案例 员工表的通用查询