概念

Java DataBase Connectivity,Java数据库连接,Java语言操作数据库

JDBC本质

其实是官方(sun公司)定义的一套操作所有关系型数据库的**规则**,即**接口**.各个数据库厂商去实现这套接口,提供数据库驱动jar包.我们可以使用这套接口(JDBC)编程,真正执行的代码是驱动jar包中的实现类

快速入门

注册 -> 连接 -> 定义 -> 执行对象 -> 执行 -> 处理 -> 释放

1. **导入驱动jar包**
   1.复制mysql-connector-java-5.1.49-bin.jar到项目的libs目录下
   2.右键复制过的文件 --> Add As Library
2. **注册驱动 *Class.forName***
   `Class.forName("com.mysql.jdbc.Driver");`
3. **获取数据库来连接对象 *Connection* *DriverManager***
   `Connection conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/db3", "root", "root");`
4. **定义sql**
   `String sql = "update account set balance = 500 where id = 1";`
5. **获取执行sql语句的对象 *Statement***
   `Statement stmt = conn.createStatement();`
6. **执行sql,接受返回结果**
   `int count = stmt.executeUpdate(sql);`
7. **处理结果**
8. **释放资源**
   `stmt.close();`
   `conn.close();`

代码演示

```
Statement stmt = null;
Connection conn = null;
try {
    // 1.注册驱动
    Class.forName("com.mysql.jdbc.Driver");
    // 2.定义sql
    String sql = "insert into account values (null,'王五',3000)";
    // 3.获取Connection对象
    conn = DriverManager.getConnection("jdbc:mysql://localhost/db3?characterEncoding=utf8", "root", "root");
    // 4.获取执行sql对象的 Statement
    stmt = conn.createStatement();
    // 5.执行sql
    int count = stmt.executeUpdate(sql);
    // 6.处理结果
    System.out.println(count);
    if (count > 0) {
        System.out.println("添加成功");
    } else {
        System.out.println("添加失败");
    }
} catch (Exception e) {
    e.printStackTrace();
```

```
} finally {
    // 7.释放资源
    // 避免空指针异常
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

详解各个对象

1.DriverMananger

概念

驱动管理对象

功能

1.注册驱动

告诉程序该使用哪一个数据库驱动jar
static void registerDriver(Driver driver)
注册与给定的驱动程序DriverManager
存在静态代码块
mysql5之后的驱动jar包可以省略注册驱动的步骤

2.获取数据库连接

static Connection getConnection(String url,String user,String password)

url
指定连接的路径
语句
jdbc:mysql://ip地址(域名):端口号/数据库名
细节
结果连接的是本机mysql服务器,并且mysql服务默认端口是3306,则url可以简写为jdbc:mysql:///数据库名 **三个///**

user
用户名
password
密码

2.Connection

概念

数据库连接对象

1.获取执行sql的对象

Statement createStatement()
PreparedStatement preparedStatement(String sql)

2.管理事务

开启事务

setAutoCommit(boolean autoCommit)

调用该方法设置参数为false,即开启事务

提交事务 **commit()**

回滚事务 **rollback()**

在catch里面,catch设置一个大的异常Exception

3.Statement

执行sql对象

1.boolean execute(String sql)

可以执行任意的sql(了解)

2.int executeUpdate(String sql)

执行**DML**(insert,update,delete)语句/**DDL**(create,alter,drop)语句

返回值:影响的行数,可以通过这个影响的行数判断DML语句是否执行成功,返回值>0的则执行成功,反之则失败

3.ResultSet executeQuery(String sql)

执行DQL(select)语句

4.ResultSet

概念

结果集对象,封装查询结果,其中的方法获取每列的值

方法

boolean next():

游标向下移动一行

判断当前行是否是最后一行末尾(是否有数据)

如果是,则返回false

如果不是则返回true

getXxx(参数)

获取数据

Xxx

代表数据类型例如

int getInt()

String getString()

参数

int

代表列的编号,从1开始

String

代表列名称

```
// 6.1 让游标向下移动一行
while (res.next()) {
    // 6.2获取数据
    int id = res.getInt(1);                 // 参数为列序号
    String name = res.getString("name");    // 参数为列名称
    double balance = res.getDouble(3);      // 参数为列序号
    System.out.println(id + "----" + name + "----" + balance);
}
```

注意

使用步骤

1.游标向下移动一行

2.判断是否有数据

3.获取数据

5.PreparedStatement

概念

执行sql的对象

1.SQL注入问题

在拼接sql时,有一些sql的特殊关键字参与字符串的拼接.会造成安全性问题

```
String sql = "select * from user where username = '" + username + "' and password = '" + password + "'";
```

2.解决SQL注入问题

使用PreparedStatement对象来解决

3.预编译的SQL

参数使用?作为占位符

不用sql的特殊关键字参与字符串的拼接,使用?代替值,然后使用PreparedStatement的setXxx方法进行赋值

```
String sql = "select * from user where username = ? and password = ?";
```

4.步骤

1.导入驱动jar包

1.复制mysql-connector-java-5.1.49-bin.jar到项目的libs目录下

2.右键复制过的文件 --> Add As Library

2.注册驱动

```
Class.forName("com.mysql.jdbc.Driver");
```

3.获取数据库来连接对象 *Connection*

```
Connection conn = DriverManager.getConnection("jdbc:mysql:///db3", "root", "root");
```

4.定义sql

注意

sql的参数使用?作为占位符

```
select * from user where  username  =  ? and  password  = ?;
```

5.获取执行sql语句的对象 *PreparedStatement*

```
conn.poreparedStatement(String sql);
```

有参数

6.给?赋值

方法

setXxx(参数1,参数2)

参数1

?的位置编号 从1开始

参数2

?的值

Xxx

代表数据类型

int getInt()

String getString()

7.执行sql,接受返回结果

```
int count = stmt.executeUpdate();
```

无参数

8.处理结果

9.释放资源

5.注意

后期都会使用PreparedStatement来完成增删改查的所有操作

1.可以防止SQL注入

2.效率更高

代码

```
public boolean login(String username, String password) {
        if (username == null || password == null) {
            return false;
        }
        Connection conn = null;
        PreparedStatement pstmt = null;
        ResultSet res = null;
        try {
            // 1.获取连接
            conn = JDBCUtils.getConnection();
            // 2.定义sql
            String sql = "select * from user where username = ? and password = ?";
            // 3.获取执行sql对象
            // stmt = conn.createStatement();        与Statement对象的不同
            pstmt = conn.prepareStatement(sql);
            // 新增.给?赋值
            pstmt.setString(1, username);
            pstmt.setString(2, password);
            // 4.执行查询
            // res = stmt.executeQuery(sql);     与Statement对象的不同
            res = pstmt.executeQuery();
            // 5.判断
            return res.next();
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {
            JDBCUtils.close(res, pstmt, conn);
        }
        return false;
    }
```

JDBC工具类

目的:

简化书写

分析:

1.注册驱动也抽取

2.抽取一个方法获取连接对象

1.需求:不想传递参数,还得保证工具类的通用性

2.解决:配置文件

jdbc.properties

3.抽取一个方法释放资源

```
url=jdbc:mysql://localhost:3306/db3?characterEncoding=utf8
user=root
password=root
driver=com.mysql.jdbc.Driver
```

```java
public class JDBCUtils {
    private static String url;
    private static String user;
    private static String password;
    private static String driver;

    // 文件的读取,只需要读取一次即可拿到这些值.使用静态代码快
    static {
        // 读取资源文件,获取值
        try {
            // 1.创建Properties集合类
            Properties pro = new Properties();

            // 获取src路径下的文件的方式  --> ClassLoader 类加载器
            ClassLoader classLoader = JDBCUtils.class.getClassLoader();
            URL resource = classLoader.getResource("jdbc.properties");
            String path = resource.getPath();

            // 2.加载文件
            // pro.load(new FileReader("src/jdbc.properties"));     //路径找不到
            pro.load(new FileReader(path));

            // 3.获取数据
            url = pro.getProperty("url");
            user = pro.getProperty("user");
            password = pro.getProperty("password");
            driver = pro.getProperty("driver");

            // 4.注册驱动
            //Class.forName("com.mysql.jdbc.Driver");
            Class.forName(driver);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
    }

    /**
     * @return
     */
    public static Connection getConnection() throws SQLException {

        // DriverManager.getConnection("jdbc:mysql://localhost/db3?
characterEncoding=utf8", "root", "root");
        return DriverManager.getConnection(url, user, password);
    }

    /**
     * 适用于Statementex的ecuteUpdate方法
     *
     * @param stmt
     * @param conn
     */
    public static void close(Statement stmt, Connection conn) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
```

```
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * 适用于Statementex的executeQuery方法,有结果集
     *
     * @param re
     * @param stmt
     * @param conn
     */
    public static void close(ResultSet re, Statement stmt, Connection conn) {
        if (re != null) {
            try {
                re.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }
}
```

JDBC控制事务

1.事务

一个包含多个步骤的业务操作.如果这个业务操作被事务管理,则着多个步骤要么同时成功,要么同时失败

2.操作

1.开启事务,在执行操作之前
2.提交事务,操作都执行完了
3.回滚事务,出现异常,在catch中

3.使用Connection对象来管理事务

开启事务 **setAutoCommit(boolean autoCommit)**:调用该方法设置参数为false,即开启事务
提交事务 **commit()**
回滚事务 **rollback()**

```
public class JDBCTransaction {
    public static void main(String[] args) {
        Connection conn = null;
        PreparedStatement pstmt1 = null;
        PreparedStatement pstmt2 = null;
        try {
```

```
            // 1.获取连接
            conn = JDBCUtils.getConnection();

            //开启事务
            conn.setAutoCommit(false);

            // 2.定义sql
            // 2.1 张三 - 500
            String sql1 = "update account set balance = balance - ? where id = ?";
            // 2.2 李四 + 500
            String sql2 = "update account set balance = balance + ? where id = ?";

            // 3.获取执行sql对象
            pstmt1 = conn.prepareStatement(sql1);
            pstmt2 = conn.prepareStatement(sql2);

            // 4.设置参数
            pstmt1.setDouble(1, 500);
            pstmt1.setInt(2, 1);
            pstmt2.setDouble(1, 500);
            pstmt2.setInt(2, 2);

            // 5.执行sql
            pstmt1.executeUpdate();
            pstmt2.executeUpdate();

            // 提交事务
            conn.commit();

        } catch (Exception e) {
            try {
                if (conn != null) {
                    // 回滚事务
                    conn.rollback();
                }
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
            e.printStackTrace();
        } finally {
            JDBCUtils.close(pstmt1, conn);
            JDBCUtils.close(pstmt2, null);
        }
    }
}
```

数据库连接池

1.概念

其实就是一个容器(集合),存放数据库连接的容器

当系统初始化好后,容器被创建,容器中会申请一些连接对象,当用户来访问数据库时,从容器中获取连接对象,用户访问完之后,会将连接对象归还给容器

2.好处

1.节约资源

2,用户访问高效

3.实现

1.标准接口 DataSource javax.sql包下的

1,方法

获取连接

getConnection()

归还连接

Connection.close()

如果连接对象Connection是从连接池中获取的,那么调用Connection.close()方法则不会再关闭连接了,而是 归还连接

2.一般我们不去实现它,由数据库厂商来实现

1.C3P0

数据库连接池技术

2.Druid

数据库连接池实现技术,阿里巴巴提供的

4.C3P0

数据库连接池技术

步骤:

1.导入jar包(三个【一个mysql驱动jar包】【两个c3p0包】)

c3p0-0.9.5.2.jar

mchange-commons-java-0.2.12.jar

mysql-connector-java-5.1.49-bin.jar

2.定义配置文件

名称

c3p0.properties 或者 c3p0-config.xml

路径

直接将文件放在src目录下

3.创建核心对象

数据库连接池对象 ComboPooledDataSource

`DataSource ds = new ComboPooledDataSource();`

4.获取连接 getConnection

`Connection conn = ds.getConnection();`

配置文件:

```
<c3p0-config>
    <default-config>
        <!-- 数据库驱动名 -->
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <!-- 数据库的url -->
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/db4?
characterEncoding=utf8</property>
        <!--用户名。 -->
        <property name="user">root</property>
        <!--密码。-->
        <property name="password">root</property>
        <!--初始化申请的连接数量.取值应在minPoolSize与maxPoolSize之间。 -->
        <property name="initialPoolSize">5</property>
        <!--连接池中最大连接数.  -->
        <property name="maxPoolSize">10</property>
        <!--超时时间,单位毫秒。 -->
        <property name="checkoutTimeout">3000</property>
    </default-config>

    <named-config name="otherc3p0">
        <!-- 数据库驱动名 -->
```

```
        <property name="driverClass">com.mysql.jdbc.Driver</property>
        <!-- 数据库的url -->
        <property name="jdbcUrl">jdbc:mysql://localhost:3306/db4?characterEncoding=utf8</property>
        <!--用户名。 -->
        <property name="user">root</property>
        <!--密码。-->
        <property name="password">root</property>
        <!--初始化时获取三个连接，取值应在minPoolSize与maxPoolSize之间。 -->
        <property name="initialPoolSize">5</property>
        <!--连接池中最大连接数.  -->
        <property name="maxPoolSize">8</property>
        <!--当连接池用完时客户端调用getConnection()后等待获取新连接的时间，超时后将抛出SQLException,如设为0则无限期等待。单位毫秒。 -->
        <property name="checkoutTimeout">1000</property>
    </named-config>
</c3p0-config>
```

5.Druid

数据库连接池实现技术,阿里巴巴提供的

1.步骤:

1.导入jar包(两个【一个mysql驱动jar包】【一个druid包】)

druid-1.0.9.jar

mysql-connector-java-5.1.49-bin.jar

2.定义配置文件

是properties形式的

可以叫任意名称,可以放在任意目录下

3.加载配置文件Properties

```
Properties pro = new Properties();
ClassLoader classLoader = DruidDataSourceFactory.class.getClassLoader();
InputStream stream = classLoader.getResourceAsStream("druid.properties");
pro.load(stream);
```

4.获取数据库连接池对象:

通过工厂来获取 DruidDataSourceFactory

`DataSource ds = DruidDataSourceFactory.createDataSource(pro);`

5.获取连接 getConnection

`Connection conn = ds.getConnection();`

配置文件

```
# druid.properties文件的配置
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/db3?characterEncoding=utf8
username=root
password=root
# 初始化连接数量
initialSize=5
# 最大连接数
maxActive=10
# 最大超时时间
maxWait=3000
```

测试类

```
public class DruidDemo01 {
    public static void main(String[] args) throws Exception {

        // 1.导入jar包
        // 2.定义配置时间

        // 3.加载配置文件
        Properties pro = new Properties();
        ClassLoader classLoader = DruidDemo01.class.getClassLoader();
        InputStream stream = classLoader.getResourceAsStream("druid.properties");
        pro.load(stream);


        // 4.获取连接池对象
        DataSource ds = DruidDataSourceFactory.createDataSource(pro);

        // 5.获取连接
        Connection conn = ds.getConnection();
        System.out.println(conn);

    }
}
```

2.定义工具类

1.定义一个类 JDBCUtils

2.提供静态代码块加载配置文件,初始化连接池对象

1.加载配置文件

```
Properties pro = new Properties();
InputStream asStream =
JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");
pro.load(asStream);
```

2.获取 DataSource

`ds = DruidDataSourceFactory.createDataSource(pro);`

3.提供方法

1.获取连接方法

通过数据库连接池获取连接

```
public static Connection getConnection() throws SQLException {
    return ds.getConnection();
}
```

2.释放资源

3.获取连接池的方法

```
public static DataSource getDataSource() {
    return ds;
}
```

工具类JDBCUtils代码

```
/**
 * Druid连接池的工具类
 */
public class JDBCUtils {
```

```
    // 1.定义一个成员变量
    private static DataSource ds;

    static {
        try {
            // 1.加载配置文件
            Properties pro = new Properties();
            InputStream asStream =
JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties");
            pro.load(asStream);

            // 2.获取DataSource
            ds = DruidDataSourceFactory.createDataSource(pro);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取连接
     */
    public static Connection getConnection() throws SQLException {
        return ds.getConnection();
    }

    /**
     * 释放资源
     */
    public static void close(Statement stmt, Connection conn) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    public static void close(ResultSet re, Statement stmt, Connection conn) {
        if (re != null) {
            try {
                re.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
        if (conn != null) {
```

```
            try {
                conn.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

    /**
     * 获取连接池
     *
     * @return
     */
    public static DataSource getDataSource() {
        return ds;
    }
}
```

测试类

```
public class DruidDemo02 {
    public static void main(String[] args) {

        Connection conn = null;
        PreparedStatement pstmt = null;

        try {
            // 1.获取连接
            conn = JDBCUtils.getConnection();

            // 2.定义sql
            String sql = "insert into account values(null,?,?)";

            // 3.获取pstmt对象
            pstmt = conn.prepareStatement(sql);

            // 4.给?赋值
            pstmt.setString(1,"王五");
            pstmt.setInt(2,30000);

            // 5.执行sql
            int count = pstmt.executeUpdate();
            System.out.println(count);
        } catch (SQLException e) {
            e.printStackTrace();
        } finally {

            // 6.释放资源
            JDBCUtils.close(pstmt,conn);
        }

    }
}

```

Spring JDBC

概念

Spring框架对JDBC的简单封装,提供了一个JDBCTMplate对象简化JDBC的开发

步骤

快速入门

```
// 2.创建JDBCTemplate对象
JdbcTemplate template = new JdbcTemplate(JDBCUtils.getDataSource());
// 定义sql
String sql = "update account set balance = 5000 where id = ?";
// 3.调用方法
int count = template.update(sql, 3);
System.out.println(count);
```

1.导入jar包

commons-logging-1.2.jar
spring-beans-5.1.10.RELEASE.jar
spring-core-5.1.10.RELEASE.jar
spring-jdbc-5.1.10.RELEASE.jar
spring-tx-5.1.10.RELEASE.jar

2.创建JdbcTemplate对象,依赖于数据源DataSource

`JdbcTemplate template = new JdbcTemplate(DataSource ds);`

3.调用JdbcTemplate的方法来完成CRUD的操作

1.执行DML语句 增删改语句

`int update(String sql,Object...args)`

参数
sql
sql语句
args
sql中?的值

2. 查询结果将结果集封装成 *map* 集合

`Map<String,Object> queryForMap(String sql,Object...args)`

将列名作为key,将值作为value
参数
sql
sql语句
args
sql中?的值
注意:
这个方法查询的结果集长度只能是1,只能查一条数据

```
@Test
public void test4(){
    String sql = "select * from emp where id = ?";
    Map<String, Object> map = template.queryForMap(sql,1001);
    System.out.println(map);
    Set<Map.Entry<String, Object>> entries = map.entrySet();
    for (Map.Entry<String, Object> entry : entries) {
        System.out.println(entry);
    }
}
```

```
{id=1001, ename=孙悟空, job_id=4, mgr=1004, joindate=2000-12-17 14:35:48.0, salary=10000.00,
bonus=null, dept_id=20}
id=1001
ename=孙悟空
job_id=4
mgr=1004
joindate=2000-12-17 14:35:48.0
salary=10000.00
bonus=null
dept_id=20
```

3.查询结果将结果集封装为 *list* 集合

`List<Map<String,Object>>  queryForList(String sql)`

参数

sql

sql语句

注意

将每一条记录封装为一个Map集合,再将Map集合装载到List集合中

```java
@Test
public void test5(){
    String sql = "select * from emp";
    List< Map<String, Object> > list = template.queryForList(sql);
    System.out.println(list);
}
```

```
[
{id=1003, ename=林冲, job_id=3, mgr=1006, joindate=2001-02-22 14:35:48.0, salary=12500.00,
bonus=5000.00, dept_id=30}, {id=1004, ename=唐僧, job_id=2, mgr=1009, joindate=2001-04-02
14:35:48.0, salary=29750.00, bonus=null, dept_id=20}, {id=1005, ename=李逵, job_id=4, mgr=1006,
joindate=2001-09-28 14:35:48.0, salary=12500.00, bonus=14000.00, dept_id=30}, {id=1006, ename=宋江,
job_id=2, mgr=1009, joindate=2001-05-01 14:35:48.0, salary=28500.00, bonus=null, dept_id=30},
{id=1007, ename=刘备, job_id=2, mgr=1009, joindate=2001-09-01 14:35:48.0, salary=24500.00,
bonus=null, dept_id=10}, {id=1008, ename=猪八戒, job_id=4, mgr=1004, joindate=2007-04-19 14:35:48.0,
salary=30000.00, bonus=null, dept_id=20}, {id=1009, ename=罗贯中, job_id=1, mgr=null, joindate=2001-
11-17 14:35:48.0, salary=50000.00, bonus=null, dept_id=10}, {id=1010, ename=吴用, job_id=3,
mgr=1006, joindate=2001-09-08 14:35:48.0, salary=15000.00, bonus=0.00, dept_id=30}, {id=1011,
ename=沙僧, job_id=4, mgr=1004, joindate=2007-05-23 14:35:48.0, salary=11000.00, bonus=null,
dept_id=20}, {id=1012, ename=李逵, job_id=4, mgr=1006, joindate=2001-12-03 14:35:48.0,
salary=9500.00, bonus=null, dept_id=30}, {id=1013, ename=小白龙, job_id=4, mgr=1004, joindate=2001-
12-03 14:35:48.0, salary=30000.00, bonus=null, dept_id=20}, {id=1014, ename=关羽, job_id=4,
mgr=1007, joindate=2002-01-23 14:35:48.0, salary=13000.00, bonus=null, dept_id=10}
]
```

4.查询结果,将结果封装为 *JavaBean* 对象

`<T> List<T> query()`

1.自己实现接口: `query(String sql, RowMapper<T> rowMapper)`

```
List<Object> list1 = template.query(sql, new RowMapper<Object>() {
@Override
public Object mapRow(ResultSet resultSet, int i) throws SQLException {
   return null;
}
});
```

```java
@Test
public void test6(){
    String sql = "select * from emp";
    List<Emp> list = template.query(sql, new RowMapper<Emp>() {
```

```
        @Override
        public Emp mapRow(ResultSet re, int i) throws SQLException {
            int id = re.getInt("id");
            String ename = re.getString("ename");
            int job_id = re.getInt("job_id");
            int mgr = re.getInt("mgr");
            Date joindate = re.getDate("joindate");
            double salary = re.getDouble("salary");
            double bonus = re.getDouble("bonus");
            int dept_id = re.getInt("dept_id");
            Emp emp = new Emp(id,ename,job_id,mgr,joindate,salary,bonus,dept_id);
            return emp;
        }
    });
    for (Emp emp : list) {
        System.out.println(emp);
    }
}
```

```
Emp{id=1001, ename='孙悟空', job_id=4, mgr=1004, joindate=2000-12-17, salary=10000.0, bonus=0.0, dept_id=20}
Emp{id=1002, ename='卢俊义', job_id=3, mgr=1006, joindate=2001-02-20, salary=16000.0, bonus=3000.0, dept_id=30}
Emp{id=1003, ename='林冲', job_id=3, mgr=1006, joindate=2001-02-22, salary=12500.0, bonus=5000.0, dept_id=30}
Emp{id=1004, ename='唐僧', job_id=2, mgr=1009, joindate=2001-04-02, salary=29750.0, bonus=0.0, dept_id=20}
Emp{id=1005, ename='李逵', job_id=4, mgr=1006, joindate=2001-09-28, salary=12500.0, bonus=14000.0, dept_id=30}
Emp{id=1006, ename='宋江', job_id=2, mgr=1009, joindate=2001-05-01, salary=28500.0, bonus=0.0, dept_id=30}
Emp{id=1007, ename='刘备', job_id=2, mgr=1009, joindate=2001-09-01, salary=24500.0, bonus=0.0, dept_id=10}
Emp{id=1008, ename='猪八戒', job_id=4, mgr=1004, joindate=2007-04-19, salary=30000.0, bonus=0.0, dept_id=20}
Emp{id=1009, ename='罗贯中', job_id=1, mgr=0, joindate=2001-11-17, salary=50000.0, bonus=0.0, dept_id=10}
Emp{id=1010, ename='吴用', job_id=3, mgr=1006, joindate=2001-09-08, salary=15000.0, bonus=0.0, dept_id=30}
Emp{id=1011, ename='沙僧', job_id=4, mgr=1004, joindate=2007-05-23, salary=11000.0, bonus=0.0, dept_id=20}
Emp{id=1012, ename='李逵', job_id=4, mgr=1006, joindate=2001-12-03, salary=9500.0, bonus=0.0, dept_id=30}
Emp{id=1013, ename='小白龙', job_id=4, mgr=1004, joindate=2001-12-03, salary=30000.0, bonus=0.0, dept_id=20}
Emp{id=1014, ename='关羽', job_id=4, mgr=1007, joindate=2002-01-23, salary=13000.0, bonus=0.0, dept_id=10}
```

2.用spring提供的实现类: `query(sql, new BeanPropertyRowMapper<Object>(Object.class))`

一般我们使用BeanPropertyRowMapper实现类,可以完成数据到JavaBean的自动封装

new BeanPropertyRowMapper<类型>(类型.class)

`List<Object> list = template.query(sql, new BeanPropertyRowMapper<Object>(Object.class));`

```java
@Test
public void test6_1(){
    String sql = "select * from emp";
    List<Emp> list = template.query(sql, new BeanPropertyRowMapper<Emp>(Emp.class));
    for (Emp emp : list) {
        System.out.println(emp);
    }
}
```

```
Emp{id=1001, ename='孙悟空', job_id=4, mgr=1004, joindate=2000-12-17, salary=10000.0, bonus=0.0, dept_id=20}
Emp{id=1002, ename='卢俊义', job_id=3, mgr=1006, joindate=2001-02-20, salary=16000.0, bonus=3000.0, dept_id=30}
Emp{id=1003, ename='林冲', job_id=3, mgr=1006, joindate=2001-02-22, salary=12500.0, bonus=5000.0, dept_id=30}
Emp{id=1004, ename='唐僧', job_id=2, mgr=1009, joindate=2001-04-02, salary=29750.0, bonus=0.0, dept_id=20}
Emp{id=1005, ename='李逵', job_id=4, mgr=1006, joindate=2001-09-28, salary=12500.0, bonus=14000.0, dept_id=30}
Emp{id=1006, ename='宋江', job_id=2, mgr=1009, joindate=2001-05-01, salary=28500.0, bonus=0.0, dept_id=30}
Emp{id=1007, ename='刘备', job_id=2, mgr=1009, joindate=2001-09-01, salary=24500.0, bonus=0.0, dept_id=10}
Emp{id=1008, ename='猪八戒', job_id=4, mgr=1004, joindate=2007-04-19, salary=30000.0, bonus=0.0, dept_id=20}
Emp{id=1009, ename='罗贯中', job_id=1, mgr=0, joindate=2001-11-17, salary=50000.0, bonus=0.0, dept_id=10}
Emp{id=1010, ename='吴用', job_id=3, mgr=1006, joindate=2001-09-08, salary=15000.0, bonus=0.0, dept_id=30}
Emp{id=1011, ename='沙僧', job_id=4, mgr=1004, joindate=2007-05-23, salary=11000.0, bonus=0.0, dept_id=20}
Emp{id=1012, ename='李逵', job_id=4, mgr=1006, joindate=2001-12-03, salary=9500.0, bonus=0.0, dept_id=30}
Emp{id=1013, ename='小白龙', job_id=4, mgr=1004, joindate=2001-12-03, salary=30000.0, bonus=0.0, dept_id=20}
Emp{id=1014, ename='关羽', job_id=4, mgr=1007, joindate=2002-01-23, salary=13000.0, bonus=0.0, dept_id=10}
```

5.查询结果,将结果封装为 *对象*

`<T> T  queryForObject (String sql, Class<T> requiredType)`

参数

sql

sql语句

requiredType

类型.class

一般用于聚合函数的查询

```java
@Test
public void test7(){
    String sql = "select count(id) from emp";
    Long total = template.queryForObject(sql,Long.class);
    System.out.println(total);
}
```

```
14
```