

连接池_JdbcTemplate

学习目标

1. 能够说出什么是数据库元数据
2. 掌握自定义数据库框架，实现增加、删除、更新方法
3. 掌握JdbcTemplate实现增删改
4. 掌握JdbcTemplate实现增查询

第1章 数据库元数据

1.1 元数据的基本概述

元数据：数据库、表、列的定义信息。

创表语句

```
CREATE TABLE `product` (  
  `pid` INT(11) NOT NULL AUTO_INCREMENT,  
  `pname` VARCHAR(20) DEFAULT NULL,  
  `price` DOUBLE DEFAULT NULL,  
  PRIMARY KEY (`pid`)  
)
```

表

pid	pname	price
1	iPhone3GS	3333
2	iPhone4	5000
3	iPhone4S	5001
4	iPhone5	5555
5	iPhone5C	3888
6	iPhone5S	5666
8	iPhone6S	7000
9	iPhone6SP	7777

元数据：数据库、表、列的定义信息。

元数据

数据

1.2 ParameterMetaData

ParameterMetaData 可用于获取有关 PreparedStatement 对象中每个参数标记的类型和属性。

```
select * from user where name=? and password=?  
// ParameterMetaData可以用来获取 ? 的个数和类型
```

1.2.1 如何获取ParameterMetaData

Interface PreparedStatement

`ParameterMetaData` `getParameterMetaData()`

通过 `PreparedStatement` 的 `getParameterMetaData()`

方法来获取到 `ParameterMetaData` 对象

1.2.2 API介绍

1.

`int getParameterCount()` 获取PreparedStatement的SQL语句参数?的个数

2.

`int getParameterType(int param)` 获取指定参数的SQL类型。

1.2.3 使用步骤

1. 获取 `ParameterMetaData` 对象
2. 使用对象调用方法

1.2.4 注意事项

1. 不是所有的数据库驱动都能后去到参数类型 (MySQL会出异常)

1.2.5 案例代码

```
public class Demo01 {  
    public static void main(String[] args) throws Exception {  
        Connection conn = DataSourceUtils.getConnection();  
        String sql = "INSERT INTO student (name, age, score) VALUES (?, ?, ?)";  
        PreparedStatement stmt = conn.prepareStatement(sql);  
        ParameterMetaData md = stmt.getParameterMetaData();  
        System.out.println("参数个数: " + md.getParameterCount());  
  
        // Parameter metadata not available for the given statement  
        // MySQL不支持获取参数类型  
        // System.out.println("参数类型: " + md.getParameterType(1));  
    }  
}
```

1.3 ResultSetMetaData

`ResultSetMetaData` 可用于获取有关 `ResultSet` 对象中列的类型和属性的信息。

ResultSet 结果集

id	NAME	age	score
1	张三	25	99.5
2	王五	35	88.5
3	张三	25	99.5
4	王五	35	88.5



`ResultSetMetaData` 获取结果集中的列名和列的类型

1.3.1 如何获取ResultSetMetaData

Interface ResultSet

`ResultSetMetaData` `getMetaData()`

通过 `ResultSet` 的 `getMetaData()` 方法来获取到

`ResultSetMetaData` 对象

1.3.2 API介绍

1.

`int getColumnCount()` 返回此 `ResultSet` 对象中的列数

2.

`String getColumnName(int column)` 获取指定列的名称

3.

`String getColumnType(int column)` 获取指定列的数据库特定类型名称

1.3.3 使用步骤

1. 获取 `ResultSetMetaData` 对象
2. 使用对象调用方法

1.3.4 案例代码

```
// ResultSetMetaData
public static void test02() throws SQLException {
    Connection conn = DataSourceUtils.getConnection();
    String sql = "SELECT * FROM student WHERE id=1";
    PreparedStatement stmt = conn.prepareStatement(sql);
    ResultSet rs = stmt.executeQuery();
    // 获取结果集元数据
    ResultSetMetaData md = rs.getMetaData();
    int num = md.getColumnCount();
    System.out.println("列数：" + num);
    for (int i = 0; i < num; i++) {
```

```

        System.out.println("列名：" + md.getColumnName(i + 1)); // 获取列名
        System.out.println("列类型：" + md.getColumnTypeName(i + 1)); // 获取类的类型
        System.out.println("-----");
    }
}

```

1.3.5 案例效果

```

列数：4
列名：id
列类型：INT
-----
列名：NAME
列类型：VARCHAR
-----
列名：age
列类型：INT
-----
列名：score
列类型：DOUBLE
-----

```

1.4 案例自定义数据库框架,实现增加、删除、更新方法

目前使用连接池工具类操作数据库的代码

```

// 增加数据
public static void testInsert() throws SQLException {
    // 获取连接
    Connection conn = DataSourceUtils.getConnection();

    String sql = "INSERT INTO student (name, age, score) VALUES (?, ?, ?)";
    PreparedStatement pstmt = conn.prepareStatement(sql);

    // 设置参数
    pstmt.setString(1, "刘德华");
    pstmt.setInt(2, 57);
    pstmt.setInt(3, 99);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数：" + i);

    // 关闭连接
    DataSourceUtils.close(conn, pstmt);
}

// 修改数据
public static void testUpdate() throws SQLException {

```

```

        Connection conn = DataSourceUtils.getConnection();

        String sql = "UPDATE student SET name=? WHERE id=?";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, "张学友");
        pstmt.setInt(2, 2);
        int i = pstmt.executeUpdate();
        System.out.println("影响的行数:" + i);

        DataSourceUtils.close(conn, pstmt);
    }

    // 删除数据
    public static void testDelete() throws SQLException {
        Connection conn = DataSourceUtils.getConnection();

        String sql = "DELETE FROM student WHERE id=?";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setInt(1, 3);
        int i = pstmt.executeUpdate();
        System.out.println("影响的行数:" + i);

        DataSourceUtils.close(conn, pstmt);
    }
}

```

```

// 增加数据
public static void testInsert() throws SQLException {
    // 获取连接
    Connection conn = DataSourceUtils.getConnection();

    String sql = "INSERT INTO student (name, age, score) VALUES (?, ?, ?)";
    PreparedStatement pstmt = conn.prepareStatement(sql);

    // 设置参数
    pstmt.setString(parameterIndex: 1, x: "刘德华");
    pstmt.setInt(parameterIndex: 2, x: 57);
    pstmt.setInt(parameterIndex: 3, x: 99);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    // 关闭连接
    DataSourceUtils.close(conn, pstmt);
}

```

我们发现增删改都需要获取连接，关闭连接，只是SQL语句不同，参数不同。

```

// 修改数据
public static void testUpdate() throws SQLException {
    Connection conn = DataSourceUtils.getConnection();

    String sql = "UPDATE student SET name=? WHERE id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setString(parameterIndex: 1, x: "张学友");
    pstmt.setInt(parameterIndex: 2, x: 2);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    DataSourceUtils.close(conn, pstmt);
}

// 删除数据
public static void testDelete() throws SQLException {
    Connection conn = DataSourceUtils.getConnection();

    String sql = "DELETE FROM student WHERE id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(parameterIndex: 1, x: 3);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    DataSourceUtils.close(conn, pstmt);
}
}

```

存在的问题：我们发现，增删改都需要获取连接，关闭连接，只是SQL语句不同。我们可以使用元数据来自动给SQL设置参数，合并成一个方法。

在 `DataSourceUtils` 中增加一个方法 `update`

```

/*
    String sql: sql语句
    Object[] params: 参数数组
*/
public static int update(String sql, Object[] params) {

```

```

Connection conn = null;
PreparedStatement st = null;

try {
    conn = getConnection();
    st = conn.prepareStatement(sql);
    // 获取到参数个数
    ParameterMetaData md = st.getParameterMetaData();
    int num = md.getParameterCount(); // 参数的个数
    for (int i = 0; i < num; i++) {
        // 将参数赋值给对应的 ? 号
        st.setObject(i + 1, params[i]);
    }
    return st.executeUpdate();
} catch (SQLException e) {
    e.printStackTrace();
} finally {
    close(conn, st);
}
return 0;
}

```

增删改都调用 `update` 方法，只需要编写SQL语句和设置参数即可，可以减少代码。

```

// 增加数据
public static void testInsert2() throws SQLException {
    String sql = "INSERT INTO student (name, age, score) VALUES (?, ?, ?);";
    Object[] params = {"郭富城", 55, 88};
    int i = DataSourceUtils.update(sql, params);
    System.out.println("影响的行数:" + i);
}

// 修改数据
public static void testUpdate2() throws SQLException {
    String sql = "UPDATE student SET name=? WHERE id=?;";
    Object[] params = {"黎明", 1};
    int i = DataSourceUtils.update(sql, params);
    System.out.println("影响的行数:" + i);
}

// 删除数据
public static void testDelete2() throws SQLException {
    String sql = "DELETE FROM student WHERE id=?;";
    Object[] params = {7};
    int i = DataSourceUtils.update(sql, params);
    System.out.println("影响的行数:" + i);
}

```

第2章 JdbcTemplate

2.1 JdbcTemplate概念

JDBC已经能够满足大部分用户最基本的需求，但是在使用JDBC时，必须自己来管理数据库资源如：获取PreparedStatement，设置SQL语句参数，关闭连接等步骤。JdbcTemplate就是Spring对JDBC的封装，目的是使JDBC更加易于使用。JdbcTemplate是Spring的一部分。JdbcTemplate处理了资源的建立和释放。他帮助我们避免一些常见的错误，比如忘了总要关闭连接。他运行核心的JDBC工作流，如Statement的建立和执行，而我们只需要提供SQL语句和提取结果。Spring源码地址：<https://github.com/spring-projects/spring-framework> 在JdbcTemplate中执行SQL语句的方法大致分为3类：

1. `execute`：可以执行所有SQL语句，一般用于执行DDL语句。
2. `update`：用于执行 `INSERT`、`UPDATE`、`DELETE` 等DML语句。
3. `queryXxx`：用于DQL数据查询语句。

2.2 JdbcTemplate使用过程

2.2.1 Druid基于配置文件实现连接池

2.2.1.1 API介绍

`org.springframework.jdbc.core.JdbcTemplate` 类方便执行SQL语句

1.

```
public JdbcTemplate(DataSource dataSource)
创建JdbcTemplate对象，方便执行SQL语句
```

2.

```
public void execute(final String sql)
execute可以执行所有SQL语句，因为没有返回值，一般用于执行DML语句。
```

2.2.1.2 使用步骤

1. 准备DruidDataSource连接池

2. 导入依赖的jar包

- `spring-beans-4.1.2.RELEASE.jar`
 - `spring-core-4.1.2.RELEASE.jar`
 - `spring-jdbc-4.1.2.RELEASE.jar`
 - `spring-tx-4.1.2.RELEASE.jar`
 - `com.springsource.org.apache.commons.logging-1.1.1.jar`
- ```
> com.springsource.org.apache.commons.logging-1.1.1.jar
> spring-beans-4.1.2.RELEASE.jar
> spring-core-4.1.2.RELEASE.jar
> spring-jdbc-4.1.2.RELEASE.jar
> spring-tx-4.1.2.RELEASE.jar
```

3. 创建 `JdbcTemplate` 对象，传入 `Druid` 连接池

4. 调用 `execute`、`update`、`queryXxx` 等方法

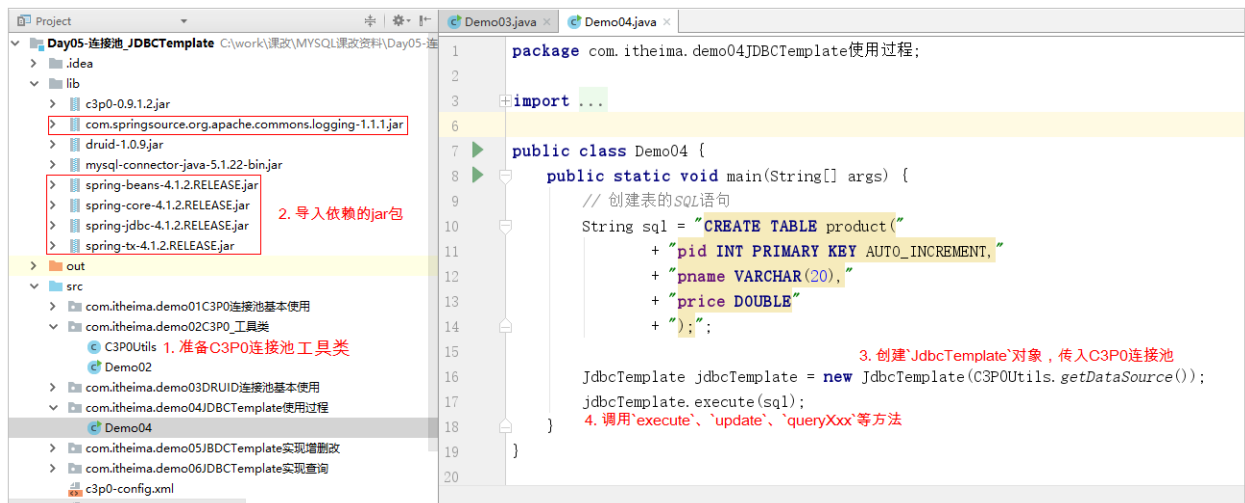
### 2.2.1.3 案例代码

```
public class Demo04 {
 public static void main(String[] args) {
 // 创建表的SQL语句
 String sql = "CREATE TABLE product("
 + "pid INT PRIMARY KEY AUTO_INCREMENT,"
 + "pname VARCHAR(20),"
 + "price DOUBLE"
 + ");";

 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 jdbcTemplate.execute(sql);
 }
}
```

### 2.2.1.4 案例效果

#### 1. 代码效果



#### 2. 执行SQL后创建数据库效果

| pid | pname  | price  |
|-----|--------|--------|
| *   | (Auto) | (NULL) |

## 2.3 JdbcTemplate实现增删改

### 2.3.1 API介绍

org.springframework.jdbc.core.JdbcTemplate 类方便执行SQL语句

#### 1.

```
public int update(final String sql)
 用于执行INSERT、UPDATE、DELETE等DML语句。
```

### 2.3.2 使用步骤

1.创建JdbcTemplate对象 2.编写SQL语句 3.使用JdbcTemplate对象的update方法进行增删改



### 2.3.3 案例代码

```
public class Demo05 {
 public static void main(String[] args) throws Exception {
 // test01();
 // test02();
 // test03();
 }

 // JDBCTemplate添加数据
 public static void test01() throws Exception {
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());

 String sql = "INSERT INTO product VALUES (NULL, ?, ?)";

 jdbcTemplate.update(sql, "iPhone3GS", 3333);
 jdbcTemplate.update(sql, "iPhone4", 5000);
 jdbcTemplate.update(sql, "iPhone4S", 5001);
 jdbcTemplate.update(sql, "iPhone5", 5555);
 jdbcTemplate.update(sql, "iPhone5C", 3888);
 jdbcTemplate.update(sql, "iPhone5S", 5666);
 jdbcTemplate.update(sql, "iPhone6", 6666);
 jdbcTemplate.update(sql, "iPhone6S", 7000);
 jdbcTemplate.update(sql, "iPhone6SP", 7777);
 jdbcTemplate.update(sql, "iPhoneX", 8888);
 }

 // JDBCTemplate修改数据
 public static void test02() throws Exception {
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());

 String sql = "UPDATE product SET pname=?, price=? WHERE pid=?";

 int i = jdbcTemplate.update(sql, "XVIII", 18888, 10);
 System.out.println("影响的行数: " + i);
 }

 // JDBCTemplate删除数据
 public static void test03() throws Exception {
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 String sql = "DELETE FROM product WHERE pid=?";
 int i = jdbcTemplate.update(sql, 7);
 System.out.println("影响的行数: " + i);
 }
}
```

### 2.3.4 案例效果

## 1. 增加数据效果

### 添加数据前

| pid    | pname  | price  |
|--------|--------|--------|
| (Auto) | (NULL) | (NULL) |

### 添加数据后

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 7   | iPhone6   | 6666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | iPhoneX   | 8888  |

## 2. 修改数据效果

### 修改数据前

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 7   | iPhone6   | 6666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | iPhoneX   | 8888  |

### 修改数据后

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 7   | iPhone6   | 6666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

## 3. 删除数据效果

### 删除数据前

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 7   | iPhone6   | 6666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

### 删除数据后

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

## 2.3.5 总结

JdbcTemplate的 `update` 方法用于执行DML语句。同时还可以在SQL语句中使用 `?` 占位，在update方法的 `Object... args` 可变参数中传入对应的参数。

## 2.4 JdbcTemplate实现查询

`org.springframework.jdbc.core.JdbcTemplate` 类方便执行SQL语句

## 2.4.1 queryForInt返回一个整数

### 2.4.1.1 API介绍

1.

```
public int queryForInt(String sql)
 执行查询语句，返回一个int类型的值。
```

### 2.4.1.2 使用步骤

1. 创建JdbcTemplate对象
2. 编写查询的SQL语句
3. 使用JdbcTemplate对象的queryForInt方法
4. 输出结果

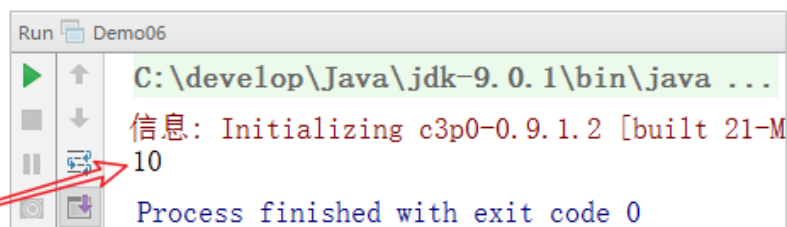
### 2.4.1.3 案例代码

```
// queryForInt返回一个整数
public static void test01() throws Exception {
 // String sql = "SELECT COUNT(*) FROM product;";
 String sql = "SELECT pid FROM product WHERE price=18888;";
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 int forInt = jdbcTemplate.queryForInt(sql);
 System.out.println(forInt);
}
```

### 2.4.1.4 案例效果

```
public static void test02() throws Exception { 查询价格为18888商品的pid,返回int类型值
 String sql = "SELECT pid FROM product WHERE price=18888;";
 JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());
 int forInt = jdbcTemplate.queryForInt(sql);
 System.out.println(forInt);
}
```

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 7   | iPhone6S  | 7000  |
| 8   | iPhone6SP | 7777  |
| 9   | iPhone6SP | 7777  |
| 10  | XVII1     | 18888 |



## 2.4.2 queryForLong返回一个long类型整数

### 2.6.2.1 API介绍

1.

```
public long queryForLong(String sql)
执行查询语句，返回一个long类型的数据。
```

### 2.4.2.2 使用步骤

1. 创建JdbcTemplate对象
2. 编写查询的SQL语句
3. 使用JdbcTemplate对象的queryForLong方法
4. 输出结果

### 2.4.2.3 案例代码

```
// queryForLong 返回一个long类型整数
public static void test02() throws Exception {
 String sql = "SELECT COUNT(*) FROM product;";
 // String sql = "SELECT pid FROM product WHERE price=18888;";
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 long forLong = jdbcTemplate.queryForLong(sql);
 System.out.println(forLong);
}
```

### 2.4.2.4 案例效果

```
57 public static void test03() throws Exception {
58 String sql = "SELECT COUNT(*) FROM product;";
59 // String sql = "SELECT pid FROM product WHERE price=18888;";
60 JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());
61 long forLong = jdbcTemplate.queryForLong(sql);
62 System.out.println(forLong);
63 }
```

查询product表中记录条数，返回long类型值

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

Run Demo06

C:\develop\Java\jdk-9.0.1\bin\java ...

2月 05, 2018 11:07:28 下午 com.mchange.v2...

信息: MLog clients using java 1.4+ standard

**9** 数据库中总共9条数据

Process finished with exit code 0

## 2.4.3 queryForObject返回String

### 2.4.3.1 API介绍

- 1.

```
public <T> T queryForObject(String sql, Class<T> requiredType)
执行查询语句，返回一个指定类型的数据。
```

### 2.4.3.2 使用步骤

1. 创建JdbcTemplate对象
2. 编写查询的SQL语句
3. 使用JdbcTemplate对象的queryForObject方法，并传入需要返回的数据的类型
4. 输出结果

### 2.4.3.3 案例代码

```
public static void test03() throws Exception {
 String sql = "SELECT pname FROM product WHERE price=7777;";
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 String str = jdbcTemplate.queryForObject(sql, String.class);
 System.out.println(str);
}
```

### 2.4.3.4 案例效果

查询价格为7777的商品的pname。返回值是字符串类型

```
65 public static void test04() throws Exception {
66 String sql = "SELECT pname FROM product WHERE price=7777;";
67 JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());
68 String str = jdbcTemplate.queryForObject(sql, String.class);
69 System.out.println(str);
70 }
```

指定返回值类型为String

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 7   | iPhone6   | 6666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

Run Demo06

```
C:\develop\Java\jdk-9.0.1\bin\java ...
iPhone6SP 结果获取到商品名称，类型为String类型
Process finished with exit code 0
```

## 2.4.4 queryForMap返回一个Map集合对象

### 2.4.4.1 API介绍

- 1.

```
public Map<String, Object> queryForMap(String sql)
执行查询语句，将一条记录放到一个Map中。
```

### 2.4.4.2 使用步骤

1. 创建JdbcTemplate对象
2. 编写查询的SQL语句
3. 使用JdbcTemplate对象的queryForMap方法
4. 处理结果

### 2.4.4.3 案例代码

```
public static void test04() throws Exception {
 String sql = "SELECT * FROM product WHERE pid=?";
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 Map<String, Object> map = jdbcTemplate.queryForMap(sql, 6);
 System.out.println(map);
}
```

### 2.4.4.4 案例效果

执行查询语句，将一条记录放到一个Map中

```
73 public static void test05() throws Exception {
74 String sql = "SELECT * FROM product WHERE pid=?";
75 JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());
76 Map<String, Object> map = jdbcTemplate.queryForMap(sql, ...args: 6);
77 System.out.println(map);
78 }
```

查询pid为6的这条记录

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

键

值

```
Run Demo06
C:\develop\Java\jdk-9.0.1\bin\java ...
{pid=6, pname=iPhone5S, price=5666.0}
Process finished with exit code 0
```

将这条记录放到一个Map中，key是字段名，value是该字段的值

## 2.4.5 queryForList返回一个List集合对象，集合对象存储Map类型数据

### 2.4.5.1 API介绍

1.

```
public List<Map<String, Object>> queryForList(String sql)
执行查询语句，返回一个List集合，List中存放的是Map类型的数据。
```

### 2.4.5.2 使用步骤

1. 创建JdbcTemplate对象
2. 编写查询的SQL语句
3. 使用JdbcTemplate对象的queryForList方法
4. 处理结果

### 2.4.5.3 案例代码

```

public static void test05() throws Exception {
 String sql = "SELECT * FROM product WHERE pid<?";
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());
 List<Map<String, Object>> list = jdbcTemplate.queryForList(sql, 8);
 for (Map<String, Object> map : list) {
 System.out.println(map);
 }
}

```

#### 2.4.5.4 案例效果

将一条记录放到一个Map中，多条记录对应多个Map，再放到一个List集合中

```

81 public static void test06() throws Exception {
82 String sql = "SELECT * FROM product WHERE pid<?";
83 JdbcTemplate jdbcTemplate = new JdbcTemplate(C3P0Utils.getDataSource());
84 List<Map<String, Object>> list = jdbcTemplate.queryForList(sql, ...args: 8);
85 for (Map<String, Object> map : list) {
86 System.out.println(map);
87 }
88 }

```

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

— 一条记录对应  
— 一个 Map

— 一条记录对应  
— 一个 Map

```

Run Demo06
{pid=1, pname=iPhone3GS, price=3333.0}
{pid=2, pname=iPhone4, price=5000.0}
{pid=3, pname=iPhone4S, price=5001.0}
{pid=4, pname=iPhone5, price=5555.0}
{pid=5, pname=iPhone5C, price=3888.0}
{pid=6, pname=iPhone5S, price=5666.0}
Process finished with exit code 0

```

#### 2.4.6 query使用RowMapper做映射返回对象

##### 2.4.6.1 API介绍

1.

```

public <T> List<T> query(String sql, RowMapper<T> rowMapper)

```

执行查询语句，返回一个List集合，List中存放的是RowMapper指定类型的数据。

##### 2.4.6.2 使用步骤

1. 定义Product类
2. 创建JdbcTemplate对象
3. 编写查询的SQL语句
4. 使用JdbcTemplate对象的query方法，并传入RowMapper匿名内部类
5. 在匿名内部类中将结果集中的一行记录转成一个Product对象

### 2.4.6.3 案例代码

```
// query使用rowMap做映射返回一个对象
public static void test06() throws Exception {
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());

 // 查询数据的SQL语句
 String sql = "SELECT * FROM product;";

 List<Product> query = jdbcTemplate.query(sql, new RowMapper<Product>() {
 @Override
 public Product mapRow(ResultSet arg0, int arg1) throws SQLException {
 Product p = new Product();
 p.setPid(arg0.getInt("pid"));
 p.setName(arg0.getString("pname"));
 p.setPrice(arg0.getDouble("price"));
 return p;
 }
 });


 for (Product product : query) {
 System.out.println(product);
 }
}
```

### 2.4.6.4 案例效果

数据库中的数据

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

程序查询到数据转成Product对象



```
Run Demo06
Product [pid=1, pname=iPhone3GS, price=3333.0]
Product [pid=2, pname=iPhone4, price=5000.0]
Product [pid=3, pname=iPhone4S, price=5001.0]
Product [pid=4, pname=iPhone5, price=5555.0]
Product [pid=5, pname=iPhone5C, price=3888.0]
Product [pid=6, pname=iPhone5S, price=5666.0]
Product [pid=8, pname=iPhone6S, price=7000.0]
Product [pid=9, pname=iPhone6SP, price=7777.0]
Product [pid=10, pname=XVIII, price=18888.0]
Process finished with exit code 0
```

## 2.4.7 query使用BeanPropertyRowMapper做映射返回对象

### 2.4.7.1 API介绍

1.

```
public <T> List<T> query(String sql, RowMapper<T> rowMapper)
 执行查询语句，返回一个List集合，List中存放的是RowMapper指定类型的数据。
```



2.

```
public class BeanPropertyRowMapper<T> implements RowMapper<T>
BeanPropertyRowMapper类实现了RowMapper接口
```

### 2.4.7.2 使用步骤

1. 定义Product类
2. 创建JdbcTemplate对象
3. 编写查询的SQL语句
4. 使用JdbcTemplate对象的query方法，并传入BeanPropertyRowMapper对象

### 2.4.7.3 案例代码

```
// query使用BeanPropertyRowMapper做映射返回对象
public static void test07() throws Exception {
 JdbcTemplate jdbcTemplate = new JdbcTemplate(JdbcUtils.getDataSource());

 // 查询数据的SQL语句
 String sql = "SELECT * FROM product;";
 List<Product> list = jdbcTemplate.query(sql, new BeanPropertyRowMapper<>
(Product.class));

 for (Product product : list) {
 System.out.println(product);
 }
}
```

### 2.4.6.4 案例效果

数据库中的数据

| pid | pname     | price |
|-----|-----------|-------|
| 1   | iPhone3GS | 3333  |
| 2   | iPhone4   | 5000  |
| 3   | iPhone4S  | 5001  |
| 4   | iPhone5   | 5555  |
| 5   | iPhone5C  | 3888  |
| 6   | iPhone5S  | 5666  |
| 8   | iPhone6S  | 7000  |
| 9   | iPhone6SP | 7777  |
| 10  | XVIII     | 18888 |

程序查询到数据转成Product对象

```
Run Demo06
Product [pid=1, pname=iPhone3GS, price=3333.0]
Product [pid=2, pname=iPhone4, price=5000.0]
Product [pid=3, pname=iPhone4S, price=5001.0]
Product [pid=4, pname=iPhone5, price=5555.0]
Product [pid=5, pname=iPhone5C, price=3888.0]
Product [pid=6, pname=iPhone5S, price=5666.0]
Product [pid=8, pname=iPhone6S, price=7000.0]
Product [pid=9, pname=iPhone6SP, price=7777.0]
Product [pid=10, pname=XVIII, price=18888.0]
Process finished with exit code 0
```

## 2.4.8 总结

JdbcTemplate的 `query` 方法用于执行SQL语句，简化JDBC的代码。同时还可以在SQL语句中使用 `?` 占位，在 `query` 方法的 `Object... args` 可变参数中传入对应的参数。