

day05-jdbc预处理对象&连接池&JdbcTemplate

- ☐ 能够描述SQL注入原理和解决方案
- ☐ 能够通过PreparedStatement完成CURD代码
- ☐ 能够理解连接池的原理
- ☐ 能够使用C3P0连接池
- ☐ 能够使用DRUID连接池
- ☐ 能够编写连接池工具类
- ☐ 掌握JdbcTemplate实现增删改
- ☐ 掌握JdbcTemplate实现查询

第一章 PreparedStatement

1.1 SQL注入问题

SQL注入：用户输入的内容作为SQL语句语法的一部分，改变了原有SQL真正的意义。假设有登录案例SQL语句如下：

```
SELECT * FROM 用户表 WHERE NAME = 用户输入的用户名 AND PASSWORD = 用户输的密码；
```

此时，当用户输入正确的账号与密码后，查询到了信息则让用户登录。但是当用户输入的账号为XXX 密码为：XXX' OR 'a'='a 时，则真正执行的代码变为：

```
SELECT * FROM 用户表 WHERE NAME = 'XXX' AND PASSWORD = ' XXX' OR 'a'='a'；
```

此时，上述查询语句时永远可以查询出结果的。那么用户就直接登录成功了，显然我们不希望看到这样的结果，这便是SQL注入问题。为此，我们使用PreparedStatement来解决对应的问题。

1.2 API详解：预处理对象

preparedStatement：预编译对象，是Statement对象的子类。

特点：

- 性能高
- 会把sql语句先编译
- 能过滤掉用户输入的关键字。

PreparedStatement预处理对象，处理的每条sql语句中所有的实际参数，都必须使用占位符?替换。

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

PreparedStatement使用，需要通过以下3步骤完成：



1. PreparedStatement预处理对象代码:

```
// 获得预处理对象，需要提供已经使用占位符处理后的SQL语句
PreparedStatement psmt = conn.prepareStatement(sql)
```

2. 设置实际参数

`void setXxx(int index, Xxx xx)` 将指定参数设置指定类型的值

参数1: index 实际参数序列号，从1开始。

参数2: xxx 实际参数值，xxx表示具体的类型。

例如:

`setString(2, "1234")` 把SQL语句中第2个位置的占位符?替换成实际参数 "1234"

3. 执行SQL语句:

```
int executeUpdate(); --执行insert update delete语句.
ResultSet executeQuery(); --执行select语句.
boolean execute(); --执行select返回true 执行其他的语句返回false.
```

1.3 插入

```
@Test
public void demo01(){
    //添加: 向分类表中添加数据
    Connection conn = null;
    PreparedStatement psmt = null;
    ResultSet rs = null;

    try {
        //1 获得连接
        conn = JdbcUtils.getConnection();
        //2 处理sql语句
        String sql = "insert into category(cname) values(? )";
        //3获得预处理对象
        psmt = conn.prepareStatement(sql);
        //4设置实际参数
        psmt.setString(1, "预处理");
        //5执行
        int r = psmt.executeUpdate();

        System.out.println(r);

    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{
        //6释放资源
        JdbcUtils.closeResource(conn, psmt, rs);
    }
}
```



1.4 更新

```
@Test
public void demo02(){
    //修改
    Connection conn = null;
    PreparedStatement psmt = null;
    ResultSet rs = null;

    try {
        conn = JdbcUtils.getConnection();

        //1 sql语句
        String sql = "update category set cname = ? where cid = ?";
        //2 获得预处理对象
        psmt = conn.prepareStatement(sql);
        //3设置实际参数
        psmt.setString(1, "测试数据");
        psmt.setInt(2, 4);
        //4执行
        int r = psmt.executeUpdate();
        System.out.println(r);

    } catch (Exception e) {
        throw new RuntimeException(e);
    } finally{
        JdbcUtils.closeResource(conn, psmt, rs);
    }
}
```

1.5 通过id查询详情

```
@Test
public void demo05(){
    //通过id查询
    Connection conn = null;
    PreparedStatement psmt = null;
    ResultSet rs = null;

    try {
        conn = JdbcUtils.getConnection();

        String sql = "select * from category where cid = ?";
        psmt = conn.prepareStatement(sql);
        psmt.setInt(1, 2);
        rs = psmt.executeQuery();
        if(rs.next()){
            System.out.println("查询到");
        } else {

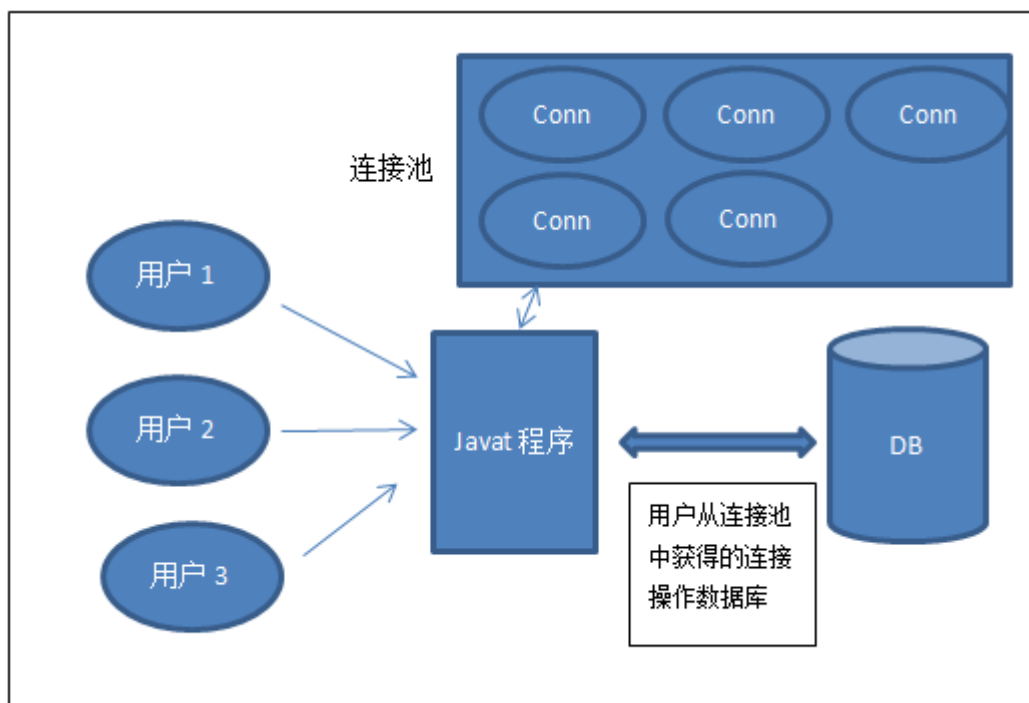
```

```
System.out.println("查询不到");  
}  
  
} catch (Exception e) {  
    throw new RuntimeException(e);  
} finally{  
    JdbcUtils.closeResource(conn, pstmt, rs);  
}  
}
```

第二章 使用连接池重写工具类

2.1 连接池原理

连接池理解为存放多个连接的集合。



使用连接池技术的目的：解决建立数据库连接耗费资源和时间很多的问题，提高性能。

2.2 编写标准的数据源(规范)

Java为数据库连接池提供了公共的接口：**javax.sql.DataSource**，各个厂商需要让自己的连接池实现这个接口。这样应用程序可以方便的切换不同厂商的连接池！

常见的连接池：C3P0、DRUID。

2.3 常用的数据源配置

2.3.1 C3P0连接池

C3P0开源免费的连接池！目前使用它的开源项目有：Spring、Hibernate等。使用C3P0连接池需要导入jar包，c3p0使用时还需要添加配置文件“c3p0-config.xml”

使用步骤

1. 添加jar包
 2. 编写配置文件 c3p0-config.xml，放在src中（注：文件名一定不要写错）
 3. 编写工具类
- 编写配置文件 c3p0-config.xml

```
<c3p0-config>
  <!-- 使用默认的配置读取连接池对象 -->
  <default-config>
    <!-- 连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day05</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">10</property>
    <property name="checkoutTimeout">2000</property>
    <property name="maxIdleTime">1000</property>
  </default-config>
</c3p0-config>
```

c3p0连接池常用的配置参数：

参数	说明
initialPoolSize	初始连接数
maxPoolSize	最大连接数
checkoutTimeout	最大等待时间
maxIdleTime	最大空闲回收时间

初始连接数：刚创建好连接池的时候准备的连接数量 最大连接数：连接池中最多可以放多少个连接 最大等待时间：连接池中如果没有连接时最长等待时间 最大空闲回收时间：连接池中的空闲连接多久没有使用就会回收

- 编写C3P0工具类

```
public class JdbcUtils {
    //创建一个C3P0的连接池对象（使用c3p0-config.xml中default-config标签中对应的参数）
    public static DataSource ds = new ComboPooledDataSource();
}
```



```
//从池中获得一个连接
public static Connection getConnection() throws SQLException {
    return ds.getConnection();
}

//释放资源
public static void closeAll(ResultSet rs, Statement stmt, Connection conn){
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        rs = null;
    }
    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        stmt = null;
    }
    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
        conn = null;
    }
}
}
```

2.3.2 DRUID连接池

Druid是阿里巴巴开发的号称为监控而生的数据库连接池，Druid（阿里自称）是目前最好的数据库连接池。在功能、性能、扩展性方面，都超过其他数据库连接池，同时加入了日志监控，可以很好的监控DB池连接和SQL的执行情况。Druid已经在阿里巴巴部署了超过600个应用，经过一年多生产环境大规模部署的严苛考验。Druid地址：

<https://github.com/alibaba/druid> DRUID连接池使用的jar包： `druid-1.0.9.jar`

使用步骤

1. 添加jar包
 2. 编写配置文件 druid.properties，放在src中（注：文件名一定不要写错）
 3. 编写工具类
- druid.properties，内容如下：



```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/day05
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000
```

常用的配置参数：

参数	说明
url	连接的数据库的位置
username	数据库的用户名
password	数据库的密码
driverClassName	驱动类名。根据url自动识别，这一项可配可不配，如果不配置druid会根据url自动识别dbType，然后选择相应的driverClassName(建议配置下)
initialSize	初始化时建立物理连接的个数。初始化发生在显示调用init方法，或者第一次getConnection时
maxActive	最大连接池数量
maxWait	获取连接时最大等待时间，单位毫秒。

- 编写DRUID工具类

API介绍

`com.alibaba.druid.pool.DruidDataSourceFactory` 类有创建Druid连接池的方法

```
public static DataSource createDataSource(Properties properties)
创建一个Druid连接池，连接池的参数使用properties中的数据
```

我们可以看到DRUID连接池在创建的时候需要一个Properties对象来设置参数，所以我们使用properties文件来保存对应的参数。

```
public class JdbcUtils {
    // 1. 声明静态数据源成员变量
    private static DataSource ds;

    // 2. 创建连接池对象
    static {
        // 加载配置文件中的数据
        InputStream is = JdbcUtils.class.getResourceAsStream("/druid.properties");
        Properties pp = new Properties();

        try {
```



```
        pp.load(is);
        // 创建连接池，使用配置文件中的参数
        ds = DruidDataSourceFactory.createDataSource(pp);
    } catch (IOException e) {
        e.printStackTrace();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

// 3. 定义公有的得到数据源的方法
public static DataSource getDataSource() {
    return ds;
}

// 4. 定义得到连接对象的方法
public static Connection getConnection() throws SQLException {
    return ds.getConnection();
}

// 5. 定义关闭资源的方法
public static void close(Connection conn, Statement stmt, ResultSet rs) {
    if (rs != null) {
        try {
            rs.close();
        } catch (SQLException e) {}
    }

    if (stmt != null) {
        try {
            stmt.close();
        } catch (SQLException e) {}
    }

    if (conn != null) {
        try {
            conn.close();
        } catch (SQLException e) {}
    }
}

// 6. 重载关闭方法
public static void close(Connection conn, Statement stmt) {
    close(conn, stmt, null);
}
}
```

2.3.3 连接池工具类的使用

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

        // 拿到连接
```



```
Connection conn = JdbcUtils.getConnection();

// 执行sql语句
String sql = "INSERT INTO student VALUES (NULL, ?, ?, ?)";
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setString(1, "李四");
pstmt.setInt(2, 30);
pstmt.setDouble(3, 50);
int i = pstmt.executeUpdate();
System.out.println("影响的函数: " + i);

// 关闭资源
JdbcUtils.close(conn, pstmt);
}
```

第三章 JdbcTemplate

3.1 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JDBC的封装，目的是使JDBC更加易于使用。JdbcTemplate处理了资源的建立和释放。它帮助我们避免一些常见的错误，比如忘了总要关闭连接。它运行核心的JDBC workflow，如Statement的建立和执行，而我们只需要提供SQL语句和提取结果。

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。

3.2 JdbcTemplate使用过程

JdbcTemplate API介绍

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。

使用步骤

1. 准备DruidDataSource连接池
2. 导入依赖的jar包
 - o `spring-beans-5.0.0.RELEASE.jar`

- o spring-core-5.0.0.RELEASE.jar
- o spring-jdbc-5.0.0.RELEASE.jar
- o spring-tx-5.0.0.RELEASE.jar
- o commons-logging-1.2.jar

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

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

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);  
    }  
}
```

案例效果

执行代码后创建数据库效果

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

3.3 JdbcTemplate实现增删改

JdbcTemplate API介绍

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

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

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);
}
}
```

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

3.4 JdbcTemplate实现查询

3.4.1 queryForObject返回指定类型的数据

JdbcTemplate API介绍

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

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

案例代码

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

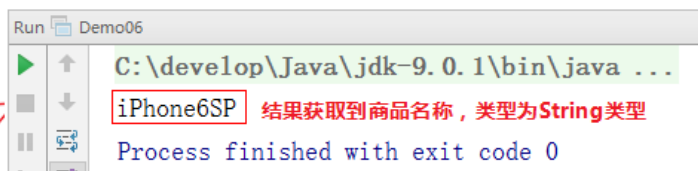
案例效果

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(JDBCUtils.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	iPhone6S	7000
8	iPhone6SP	7777
9	iPhone6SP	7777
10	XVIII	18888



3.4.2 query使用RowMapper做映射返回对象

API介绍

1. `public <T> List<T> query(String sql, RowMapper<T> rowMapper, Object... args)`
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。

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

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 rs, int rowNum) throws SQLException {
            Product p = new Product();
            p.setPid(rs.getInt("pid"));
            p.setName(rs.getString("pname"));
            p.setPrice(rs.getDouble("price"));
            return p;
        }
    });


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

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
```

3.4.3 query使用BeanPropertyRowMapper做映射返回对象

API介绍

1. `public <T> List<T> query(String sql, RowMapper<T> rowMapper, Object... args)`
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

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

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);
    }
}
```

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
```