

连接池_JdbcTemplate

回顾：

使用JDBC连接的主要四个参数：

1. `USER`：用户名
2. `PASSWORD`：密码
3. `URL`：连接数据库的字符串
4. `DRIVER_CLASS`:连接数据库的驱动

JDBC核心API

1. `DriverManager`实现类：注册和加载驱动，得到连接对象
2. `Connection`：连接对象，创建Statement对象
3. `Statement`接口：表示SQL语句对象，得到结果集
4. `ResultSet`接口:表示查询的数据，对数据进行遍历

DriverManager中的方法

提供如下方法册驱动

```
static void registerDriver(Driver driver)
向 DriverManager 注册给定驱动程序。
```

如下方法获取数据库连接

```
static Connection getConnection(String url, String user, String password)
连接到给定数据库 URL，并返回连接。
```

Connection接口中的方法

有如下方法获取到 `Statement` 对象

```
Statement createStatement()
创建一个 Statement 对象来将 SQL 语句发送到数据库
```

有获取 `PreparedStatement` 对象的方法

```
PreparedStatement prepareStatement(String sql)
会先将SQL语句发送给数据库预编译。PreparedStatement对象会引用着预编译后的结果。
```

使用JDBC开发步骤：

1. 注册驱动
2. 获取连接对象
3. 得到语句执行对象
4. 执行sql

5. 处理结果集
6. 释放资源

ResultSet接口的方法

`boolean next()`

- 1) 游标向下移动一个
- 2) 判断是否有记录，是否指向最后一条记录的后面。

数据类型 `getXxx(参数)`

- 1) 通过字段名得到参数
- 2) 通过列号得到参数

PreparedStatement中的方法

设置SQL语句参数，和执行参数化的SQL语句的方法

1. `void setXxx(int parameterIndex, Xxx x)`
将指定参数设置为给定 Java Xxx 值。

2. `void setObject(int parameterIndex, Object x)`
使用给定对象设置指定参数的值。

3. `void setString(int parameterIndex, String x)`
将指定参数设置为给定 Java String 值。

4. `ResultSet executeQuery()`
在此 PreparedStatement 对象中执行 SQL 查询，并返回该查询生成的ResultSet对象。

5. `int executeUpdate()`
在此 PreparedStatement 对象中执行 SQL 语句，该语句必须是一个 SQL 数据操作语言 (Data Manipulation Language, DML) 语句，比如 INSERT、UPDATE 或 DELETE 语句；或者是无返回内容的 SQL 语句，比如 DDL 语句。

JDBC中操作事务的方法

Connection 接口中与事务有关的方法

1. `void setAutoCommit(boolean autoCommit) throws SQLException;`
false：开启事务，ture：关闭事务

2. `void commit() throws SQLException;`
提交事务

3. `void rollback() throws SQLException;`
回滚事务

学习目标

1. 能够理解连接池解决现状问题的原理
2. 能够使用C3P0连接池
3. 能够编写C3P0连接池工具类
4. 能够使用DRUID连接池
5. 能够使用JdbcTemplate执行SQL语句
6. 能够理解JdbcTemplate的原理

第1章 C3P0连接池

1.1 准备数据

```
CREATE TABLE student (  
    id INT PRIMARY KEY AUTO_INCREMENT,  
    NAME VARCHAR(20),  
    age INT,  
    score DOUBLE DEFAULT 0.0  
);
```

1.2 没有连接池的现状

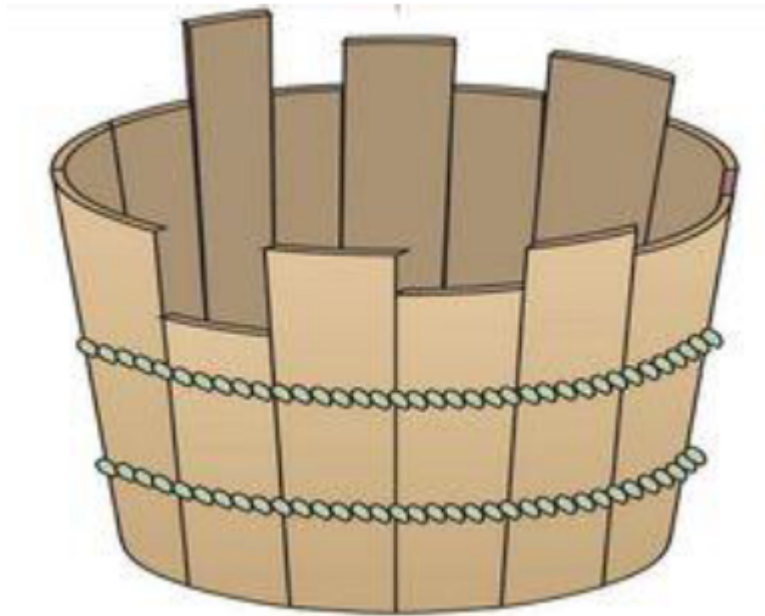
1. 之前JDBC访问数据库的步骤：**创建数据库连接**→**运行SQL语句**→**关闭连接** 每次数据库访问执行这样重复的动作



2. 每次创建数据库连接的问题

- 获取数据库连接需要消耗比较多的资源，而每次操作都要重新获取新的连接对象，执行一次操作就把连接关闭，而数据库创建连接通常需要消耗相对较多的资源，创建时间也较长。这样数据库连接对象的使用率低。
- 假设网站一天10万访问量，数据库服务器就需要创建10万次连接，极大的浪费数据库的资源，并且极易造成数据库服务器内存溢出

解决思路：木桶理论



根据木桶理论，我们提升数据库的访问效率，就应该提高数据库的连接速度

解决两个问题

1. 提升数据库连接速度
2. 提升连接对象使用率，连接对象多次重复使用

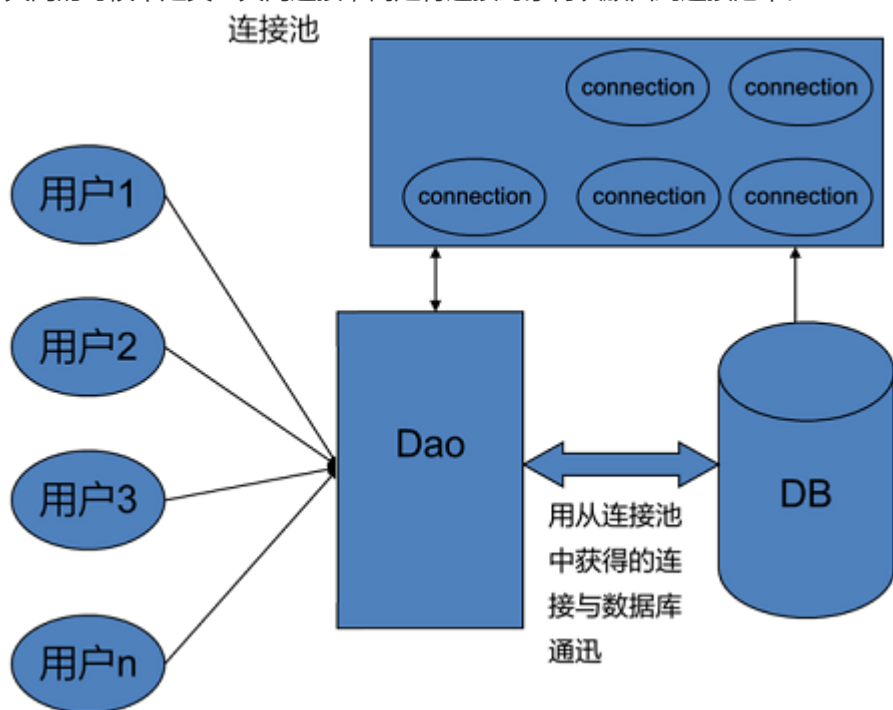
1.3 连接池解决现状问题的原理

我们现实生活中每日三餐,我们并不会吃一餐饭就将碗丢掉,而是吃完饭后将碗放到碗柜中，下一餐接着使用。目的是重复利用碗，我们的数据库连接也可以重复使用，可以减少数据库连接的创建次数。提高数据库连接对象的使用率。

连接池的原理：

1. 程序一开始就创建一定数量的连接，放在一个容器中，这个容器称为连接池(相当于碗柜/容器)。
2. 使用的时候直接从连接池中取一个已经创建好的连接对象。

3. 关闭的时候不是真正关闭连接，而是将连接对象再次放回到连接池中。



生活中的连接池：



打普通电话

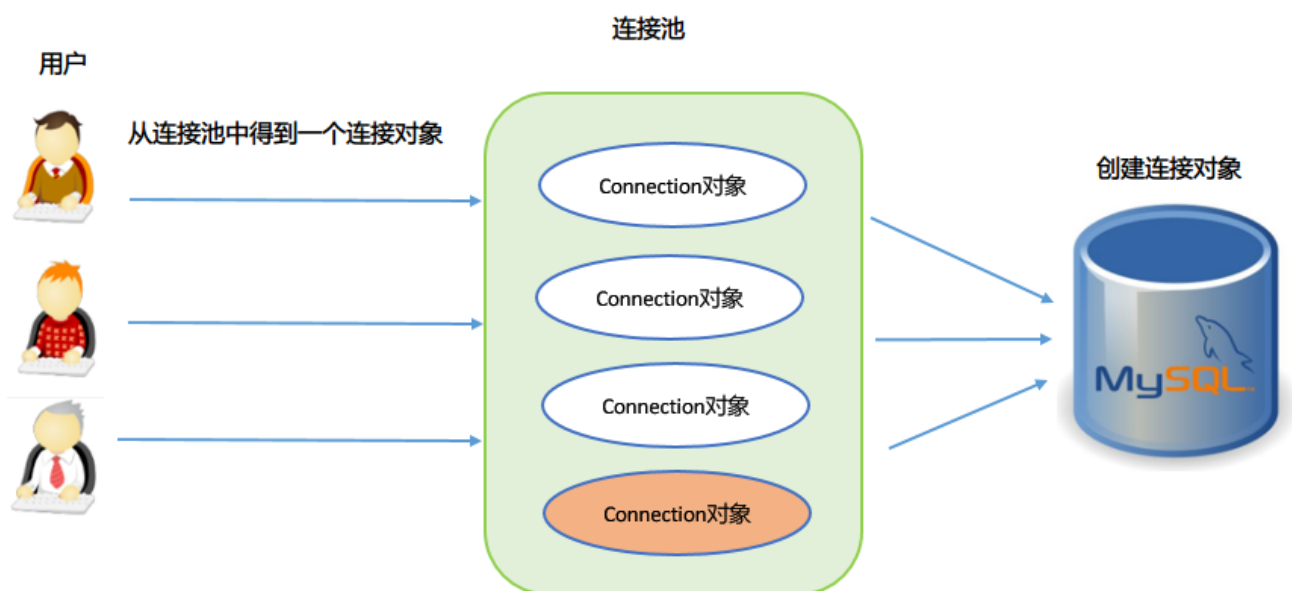




特点

创建时	由服务器一开始就创建了大量的连接对象放在内存中
使用时	不用自己再创建连接，而是从内存(连接池)中取出一个空闲的连接对象直接使用。效率提高
关闭时	连接池不是关闭连接对象，而是将连接对象再次放回到连接池中。

破天版原理图：



数据库连接池相关API 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。

数据源(连接池)接口: `javax.sql.DataSource` 中的方法

```
Connection getConnection()
```

从连接池中取出一个连接。

常见的连接池: C3P0、 Druid、 DBCP 等

扩展阅读：

DataSource本身只是Oracle公司提供的接口，没有具体的实现，它的实现由连接池的数据库厂商去实现。我们只需要学习这个工具如何使用即可。常用的连接池实现组件有这些：

！ 阿里巴巴-德鲁伊druid连接池：Druid是阿里巴巴开源平台上的一个项目，整个项目由数据库连接池、插件框架和SQL解析器组成。该项目主要是为了扩展JDBC的一些限制，可以让程序员实现一些特殊的需求。





！ DBCP(DataBase Connection Pool)数据库连接池，是Apache上的一个Java连接池项目，也是Tomcat使用的连接池组件。dbcp没有自动回收空闲连接的功能。

！ C3P0是一个开源的JDBC连接池，它实现了数据源和JNDI绑定，支持JDBC3规范和JDBC2的标准扩展。C3P0是异步操作的，所以一些操作时间过长的JDBC通过其它的辅助线程完成。目前使用它的开源项目有Hibernate，Spring等。C3P0有自动回收空闲连接功能。

Proxool数据库连接池技术，它是sourceforge下的一个开源项目，这个项目提供一个健壮、易用的连接池，最为关键的是这个连接池提供监控的功能，方便易用，便于发现连接泄漏的情况。

1.4 C3P0连接池简介

C3P0地址：<https://sourceforge.net/projects/c3p0/?source=navbar> C3P0是一个开源的连接池。Hibernate框架，默认推荐使用C3P0作为连接池实现。C3P0的jar包：`c3p0-0.9.5.2.jar`

 <code>c3p0-0.9.5.2.jar</code>	497,865
 <code>c3p0-0.9.5.2-sources.jar</code>	359,189
 <code>c3p0-config.xml</code>	1,078
 <code>mchange-commons-java-0.2.12.jar</code>	618,093

使用到的jar包

1.5 常用的配置参数解释

常用的配置参数：

参数	说明
<code>initialPoolSize</code>	初始连接数 创建连接池对象的时候 创建的连接数
<code>maxPoolSize</code>	最大连接数 允许最多多少个连接对象产生
<code>checkoutTimeout</code>	最大等待时间 连接池中无空闲连接时，外界最长等待时间。超过就抛异常
<code>maxIdleTime</code>	最大空闲回收时间 连接池中空闲连接多久就回收

`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`：连接池中无空闲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

完整参数

分类	属性	描述
必须项	user	用户名
	password	密码
	driverClass	驱动类名 mysql驱动, com.mysql.jdbc.Driver
	jdbcUrl	数据库URL路径 mysql路径, jdbc:mysql://localhost:3306/数据库
基本配置	acquireIncrement	连接池无空闲连接可用时, 一次性创建的新连接数 默认值: 3
	initialPoolSize	连接池初始化时创建的连接数 默认值: 3
	maxPoolSize	连接池中拥有的最大连接数 默认值: 15
	minPoolSize	连接池保持的最小连接数。
	maxIdleTime	连接的最大空闲时间。如果超过这个时间, 某个数据库连接还没有被使用, 则会断开掉这个连接, 如果为0, 则永远不会断开连接。 默认值: 0
管理连接池的大小和连接的生存时间 (扩展)	maxConnectionAge	配置连接的生存时间, 超过这个时间的连接将由连接池自动断开丢弃。当然正在使用的连接不会马上断开, 而是等待它close再断开。配置为0的时候则不会对连接的生存时间进行限制。默认值0
	maxIdleTimeExcessConnections	这个配置主要是为了减轻连接池的负载, 配置不为0, 则会将连接池中的连接数量保持到minPoolSize, 为0则不处理。
配置 PreparedStatement缓存 (扩展)	maxStatements	连接池为数据源缓存的 PreparedStatement 的总数。由于 PreparedStatement 属于单个 Connection, 所以这个数量应该根据应用中平均连接数乘以每个连接的平均 PreparedStatement 来计算。为0的时候不缓存, 同时 maxStatementsPerConnection 的配置无效。
	maxStatementsPerConnection	连接池为数据源单个 Connection 缓存的 PreparedStatement 数, 这个配置比 maxStatements 更有意义, 因为它缓存的服务对象是单个数据连接, 如果设置的好, 肯定是可以提高性能的。为0的时候不缓存。

1.6 C3P0连接池基本使用

1.6.1 C3P0配置文件

我们看到要使用C3P0连接池, 需要设置一些参数。那么这些参数怎么设置最为方便呢? 使用配置文件方式。

配置文件的要求:

1. 文件名一定是: c3p0-config.xml
2. 放在源代码即src目录下
3. 配置方式一: 使用默认配置 (default-config)

如果连接池是使用无参的构造方法实例化连接池对象的时候, 使用的是默认配置

4. 配置方式二：使用命名配置 (named-config)

如果连接池使用指定的命名，构造 指定指定的命名配置

可以指定多个命名配置，可以选择其中的任何一个配置

多配置有什么好处呢？

- a) 可以连接不同的数据库，如：day25,day23
- b) 可以连接不同厂商的数据库，如：mysql，oracle
- c) 可以指定不同的连接池参数

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/day25</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>

  <named-config name="itheimac3p0">
    <!-- 连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day25</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
    <property name="initialPoolSize">5</property>
    <property name="maxPoolSize">15</property>
    <property name="checkoutTimeout">2000</property>
    <property name="maxIdleTime">1000</property>
  </named-config>
</c3p0-config>
```

1.6.2 API介绍

`com.mchange.v2.c3p0.ComboPooledDataSource` 类表示C3P0的连接池对象，常用2种创建连接池的方式：
1. 无参构造，使用默认配置，
2. 有参构造，使用命名配置

1. `public ComboPooledDataSource()`
无参构造使用默认配置（使用xml中default-config标签中对应的参数）
2. `public ComboPooledDataSource(String configName)`
有参构造使用命名配置（configName：xml中配置的名称，使用xml中named-config标签中对应的参数）
3. `public Connection getConnection() throws SQLException`
从连接池中取出一个连接

1.6.3 使用步骤

1. 导入jar包 `c3p0-0.9.5.2.jar`
2. 编写 `c3p0-config.xml` 配置文件，配置对应参数
3. 将配置文件放在src目录下
4. 创建连接池对象 `ComboPooledDataSource`，使用默认配置或命名配置
5. 从连接池中获取连接对象
6. 使用连接对象操作数据库
7. 关闭资源

1.6.4 注意事项

C3P0配置文件名称必须为 `c3p0-config.xml` C3P0命名配置可以有多个

1.6.5 案例代码

1. 配置文件

```
<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/day25</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>

  <named-config name="itheimac3p0">
    <!-- 连接参数 -->
    <property name="driverClass">com.mysql.jdbc.Driver</property>
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/day25</property>
    <property name="user">root</property>
    <property name="password">root</property>
  </named-config>
</c3p0-config>
```

```

<!-- 连接池参数 -->
<property name="initialPoolSize">5</property>
<property name="maxPoolSize">15</property>
<property name="checkoutTimeout">2000</property>
<property name="maxIdleTime">1000</property>
</named-config>
</c3p0-config>

```

2. java代码

```

public class Demo01 {

    public static void main(String[] args) throws Exception {
        // 方式一：使用默认配置 ( default-config )
        // new ComboPooledDataSource();
        //      ComboPooledDataSource ds = new ComboPooledDataSource();

        // 方式二：使用命名配置 ( named-config : 配置名 )
        // new ComboPooledDataSource("配置名");
        ComboPooledDataSource ds = new ComboPooledDataSource("otherc3p0");

        //      for (int i = 0; i < 10; i++) {
        //          Connection conn = ds.getConnection();
        //          System.out.println(conn);
        //      }

        // 从连接池中取出连接
        Connection conn = ds.getConnection();

        // 执行SQL语句
        String sql = "INSERT INTO student VALUES (NULL, ?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, "张三");
        pstmt.setInt(2, 25);
        pstmt.setDouble(3, 99.5);

        int i = pstmt.executeUpdate();
        System.out.println("影响的行数： " + i);
        pstmt.close();
        conn.close(); // 将连接还回连接池中
    }
}

```

1.6.6 案例效果

1. 正常获取连接池中连接

获取连接池中连接

```
7 public class Demo01 {
8
9 public static void main(String[] args) throws Exception {
10 // 方式一：使用默认配置 (default-config)
11 new ComboPooledDataSource();
12 ComboPooledDataSource ds = new ComboPooledDataSource();
13 for (int i = 0; i < 9; i++) {
14 Connection conn = ds.getConnection();
15 System.out.println(conn);
16 }
17 }
```

```
<default-config>
<!-- 连接参数 -->
<property name="driverClass">com.mysql.jdbc.Driver</property>
<property name="jdbcUrl">jdbc:mysql://localhost:3306/day25</property>
<property name="user">root</property>
<property name="password">root</property>

<!-- 连接池参数 -->
<property name="initialPoolSize">5</property>
<property name="maxPoolSize">9</property>
<property name="checkoutTimeout">3000</property>
</default-config>
```

4. 控制台输出9个连接

```
Run Demo01
信息: Initializing c3p0 pool... com.mchange.v2.c3p0.Combo
com.mchange.v2.c3p0.impl.NewProxyConnection@4944252c
com.mchange.v2.c3p0.impl.NewProxyConnection@a3d8174
com.mchange.v2.c3p0.impl.NewProxyConnection@732c2a62
com.mchange.v2.c3p0.impl.NewProxyConnection@41fecb8b
com.mchange.v2.c3p0.impl.NewProxyConnection@625732
com.mchange.v2.c3p0.impl.NewProxyConnection@66498326
com.mchange.v2.c3p0.impl.NewProxyConnection@1e6454ec
com.mchange.v2.c3p0.impl.NewProxyConnection@b62d79
com.mchange.v2.c3p0.impl.NewProxyConnection@5aceed4
Process finished with exit code 0
```

2. 获取连接池中连接超时

获取连接池中连接超时

```
7 public class Demo01 {
8
9 public static void main(String[] args) throws Exception {
10 // 方式一：使用默认配置 (default-config)
11 new ComboPooledDataSource();
12 ComboPooledDataSource ds = new ComboPooledDataSource();
13 for (int i = 0; i < 10; i++) {
14 Connection conn = ds.getConnection();
15 System.out.println(conn);
16 }
17 }
```

```
<default-config>
<!-- 连接参数 -->
<property name="driverClass">com.mysql.jdbc.Driver</property>
<property name="jdbcUrl">jdbc:mysql://localhost:3306/day25</property>
<property name="user">root</property>
<property name="password">root</property>

<!-- 连接池参数 -->
<property name="initialPoolSize">5</property>
<property name="maxPoolSize">9</property>
<property name="checkoutTimeout">3000</property>
```

5. 获取到连接池中的9个连接

```
Run Demo01
信息: Initializing c3p0 pool... com.mchange.v2.c3p0.Combo
com.mchange.v2.c3p0.impl.NewProxyConnection@4944252c
com.mchange.v2.c3p0.impl.NewProxyConnection@a3d8174
com.mchange.v2.c3p0.impl.NewProxyConnection@732c2a62
com.mchange.v2.c3p0.impl.NewProxyConnection@41fecb8b
com.mchange.v2.c3p0.impl.NewProxyConnection@625732
com.mchange.v2.c3p0.impl.NewProxyConnection@66498326
com.mchange.v2.c3p0.impl.NewProxyConnection@1e6454ec
com.mchange.v2.c3p0.impl.NewProxyConnection@b62d79
com.mchange.v2.c3p0.impl.NewProxyConnection@5aceed4
Exception in thread "main" java.sql.SQLException: An
attempt by a client to checkout a Connection has timed out.
Process finished with exit code 0
```

6. 因为连接池中没有连接了，获取不到，3秒后报错

3. 使用连接池中的连接往数据库添加数据

```

28 // 从连接池中取出连接
29 Connection conn = ds.getConnection();
30
31 // 执行SQL语句
32 String sql = "INSERT INTO student VALUES (NULL, ?, ?, ?);";
33 PreparedStatement pstmt = conn.prepareStatement(sql);
34 pstmt.setString( parameterIndex: 1, x: "张三");
35 pstmt.setInt( parameterIndex: 2, x: 25);
36 pstmt.setDouble( parameterIndex: 3, x: 99.5);
37
38 int i = pstmt.executeUpdate();
39 System.out.println("影响的行数: " + i);
40 pstmt.close();
41 conn.close(); // 将连接还回连接池中

```

往数据库中添加一条数据

id	NAME	age	score
1	张三	25	99.5

1.6.7 总结

配置文件名称必须为：c3p0-config.xml，将配置文件放在src目录下 使用配置文件方式好处：只需要单独修改配置文件，不用修改代码 多个配置的好处：

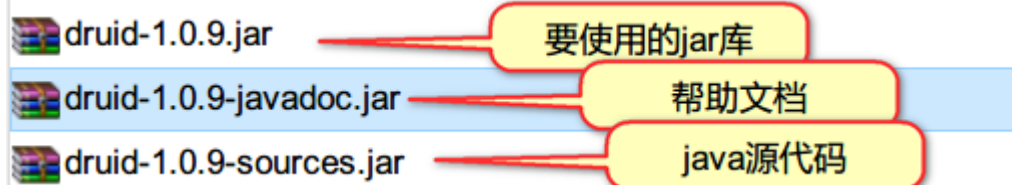
1. 可以连接不同的数据库：db1,db2
2. 可以使用不同的连接池参数：maxPoolSize
3. 可以连接不同厂商的数据库：Oracle或MySQL

第2章 DRUID连接池

2.1 DRUID简介

Druid是阿里巴巴开发的号称为监控而生的数据库连接池，Druid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



2.2 DRUID常用的配置参数

常用的配置参数：

参数	说明
jdbcUrl	连接数据库的url：mysql：jdbc:mysql://localhost:3306/druid2
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
maxIdle	已经不再使用，配置了也没效果
minIdle	最小连接池数量
maxWait	获取连接时最大等待时间，单位毫秒。

2.3 DRUID连接池基本使用

2.3.1 API介绍

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

```
public static DataSource createDataSource(Properties properties)
创建一个连接池，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。DRUID连接池的配置文件名称随便，建议放到src目录下面方便加载。`druid.properties` 文件内容：

```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/day25
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000
maxIdle=6
minIdle=3
```

2.3.2 使用步骤

1. 在src目录下创建一个properties文件，并设置对应参数
2. 加载properties文件的内容到Properties对象中

3. 创建DRUID连接池，使用配置文件中的参数
4. 从DRUID连接池中取出连接
5. 执行SQL语句
6. 关闭资源

2.3.3 案例代码

1. 在src目录下新建一个DRUID配置文件，命名为：druid.properties，内容如下

```
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/day25
username=root
password=root
initialSize=5
maxActive=10
maxWait=3000
maxIdle=6
minIdle=3
```

java代码

```
public class Demo02 {
    public static void main(String[] args) throws Exception {
        // 加载配置文件中的配置参数
        InputStream is = Demo03.class.getResourceAsStream("/druid.properties");
        Properties pp = new Properties();
        pp.load(is);

        // 创建连接池，使用配置文件中的参数
        DataSource ds = DruidDataSourceFactory.createDataSource(pp);

        // for (int i = 0; i < 10; i++) {
        //     Connection conn = ds.getConnection();
        //     System.out.println(conn);
        // }

        // 从连接池中取出连接
        Connection conn = ds.getConnection();

        // 执行SQL语句
        String sql = "INSERT INTO student VALUES (NULL, ?, ?, ?)";
        PreparedStatement pstmt = conn.prepareStatement(sql);
        pstmt.setString(1, "王五");
        pstmt.setInt(2, 35);
        pstmt.setDouble(3, 88.5);

        int i = pstmt.executeUpdate();
        System.out.println("影响的行数： " + i);

        // 执行查询
        sql = "SELECT * FROM student";
        ResultSet rs = pstmt.executeQuery(sql);
```



```

        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            int age = rs.getInt("age");
            double score = rs.getDouble("score");
            System.out.println("id: " + id + " ,name: " + name + " ,age = " + age + " ,score = " + score);
        }

        pstmt.close();
        conn.close(); // 将连接还回连接池中
    }
}

```

2.3.4 案例效果

1. 正常获取连接池中的连接

正常获取连接池中的连接

```

13 public static void main(String[] args) throws Exception {
14     // 加载配置文件中的数据
15     InputStream is = Demo03.class.getResourceAsStream( name: "/druid.properties");
16     Properties pp = new Properties();
17     pp.load(is); // 2.加载配置文件中的数据到Properties对象中
18
19     // 创建连接池，使用配置文件中的参数 // 3.使用配置文件中的参数创建DRUID连接池
20     DataSource ds = DruidDataSourceFactory.createDataSource(pp);
21
22     for (int i = 0; i < 10; i++) { // 4.获取连接池中的10个连接
23         Connection conn = ds.getConnection();
24         System.out.println(conn);
25     }

```

1.在配置文件中配置对应参数

```

druid.properties
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/day25
username=root
password=root
initialSize=5
maxActive=10 最大连接数10
maxWait=3000
maxIdle=6
minIdle=3

```

5.运行效果，10个连接都打印出来

```

Run Demo03
com.mysql.jdbc.JDBC4Connection@117e949d
com.mysql.jdbc.JDBC4Connection@6db9f5a4
com.mysql.jdbc.JDBC4Connection@5f8edcc5
com.mysql.jdbc.JDBC4Connection@7b02881e
com.mysql.jdbc.JDBC4Connection@1ebd319f
com.mysql.jdbc.JDBC4Connection@3c0be339
com.mysql.jdbc.JDBC4Connection@15ca7889
com.mysql.jdbc.JDBC4Connection@7a675056
com.mysql.jdbc.JDBC4Connection@d21a74c
com.mysql.jdbc.JDBC4Connection@6e509ffa
Process finished with exit code 0

```

2. 获取连接池中的连接超时

获取连接池中的连接超时

```
13 public static void main(String[] args) throws Exception {
14     // 加载配置文件中的数据
15     InputStream is = Demo03.class.getResourceAsStream( name: "/druid.properties");
16     Properties pp = new Properties();
17     pp.load(is); 2.加载配置文件中的数据到Properties对象中
18
19     // 创建连接池，使用配置文件中的参数 3.使用配置文件中的参数创建DRUID连接池
20     DataSource ds = DruidDataSourceFactory.createDataSource(pp);
21
22     for (int i = 0; i < 10; i++) { 4.从连接池中获取11个连接
23         Connection conn = ds.getConnection();
24         System.out.println(conn);
25     }
```

1.在配置文件中配置对应参数

```
druid.properties
driverClassName=com.mysql.jdbc.Driver
url=jdbc:mysql://127.0.0.1:3306/day25
username=root
password=root
initialSize=5
maxActive=10 最大连接数10
maxWait=3000 最大超时等待时间3秒
maxIdle=6
minIdle=3
```

Run Demo03 5.运行效果，10个连接都打印出来

```
com.mysql.jdbc.JDBC4Connection@117e949d
com.mysql.jdbc.JDBC4Connection@6db9f5a4
com.mysql.jdbc.JDBC4Connection@5f8edcc5
com.mysql.jdbc.JDBC4Connection@7b02881e
com.mysql.jdbc.JDBC4Connection@1ebd319f
com.mysql.jdbc.JDBC4Connection@3c0be339
com.mysql.jdbc.JDBC4Connection@15ca7889
com.mysql.jdbc.JDBC4Connection@7a675056
com.mysql.jdbc.JDBC4Connection@d21a74c
com.mysql.jdbc.JDBC4Connection@6e509ffa
Exception in thread "main" com.alibaba.druid.pool.
GetConnectionTimeoutException: wait millis 3000, active 10
Process finished with exit code 0
```

5.运行效果，前面10个连接取出来并打印，
取第11个连接的时候，连接池中没有连接了
过了3秒出现获取连接超时异常

3. 使用DRUID连接池中的连接操作数据库

```
12 public class Demo03 {
13     public static void main(String[] args) throws Exception {
14         // 加载配置文件中的数据
15         InputStream is = Demo03.class.getResourceAsStream( name: "/druid.properties");
16         Properties pp = new Properties();
17         pp.load(is);
18         // 创建连接池，使用配置文件中的参数
19         DataSource ds = DruidDataSourceFactory.createDataSource(pp);
20
21         // 从连接池中取出连接
22         Connection conn = ds.getConnection();
23         // 执行SQL语句
24         String sql = "INSERT INTO student VALUES (NULL, ?, ?, ?)";
25         PreparedStatement pstmt = conn.prepareStatement(sql);
26         pstmt.setString( parameterIndex: 1, x: "王五");
27         pstmt.setInt( parameterIndex: 2, x: 35);
28         pstmt.setDouble( parameterIndex: 3, x: 88.5);
29         int i = pstmt.executeUpdate();
30         System.out.println("影响的行数: " + i);
31
32         // 执行查询
33         sql = "SELECT * FROM student;";
34         ResultSet rs = pstmt.executeQuery(sql);
35         while (rs.next()) {
36             int id = rs.getInt( columnLabel: "id");
37             String name = rs.getString( columnLabel: "name");
38             int age = rs.getInt( columnLabel: "age");
39             double score = rs.getDouble( columnLabel: "score");
40             System.out.println("id: " + id + ", name: " + name + ", age = " + age + s
41         }
42
43         pstmt.close();
44         conn.close(); // 将连接归还连接池中
45     }
46 }
```

操作SQL语句运行效果：

操作之前数据库数据

id	NAME	age	score
1	张三	25	99.5

操作之后数据库数据

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

代码查询出的数据

Run Demo03

```
id: 1, name: 张三, age = 25, score = 99.5
id: 2, name: 王五, age = 35, score = 88.5
```

2.3.5 总结

DRUID连接池根据Properties对象中的数据作为连接池参数去创建连接池，我们自己定义properties类型的配置文件，名称自己取，也可以放到其他路径，建议放到src目录下方便加载。不管是C3P0连接池，还是DRUID连接池，配置大致都可以分为2种：1.连接数据库的参数，2.连接池的参数，这2种配置大致参数作用都相同，只是参数名称可能不一样。

2.4 Jdbc工具类

我们每次操作数据库都需要创建连接池，获取连接，关闭资源，都是重复的代码。我们可以将创建连接池和获取连接池的代码放到一个工具类中，简化代码。

Jdbc工具类步骤：

1. 声明静态数据源成员变量
2. 创建连接池对象
3. 定义公有的得到数据源的方法
4. 定义得到连接对象的方法
5. 定义关闭资源的方法

案例代码 JdbcUtils.java

```
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);
}
}

```

测试类代码

```

public class Demo03 {
    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);
    }
}

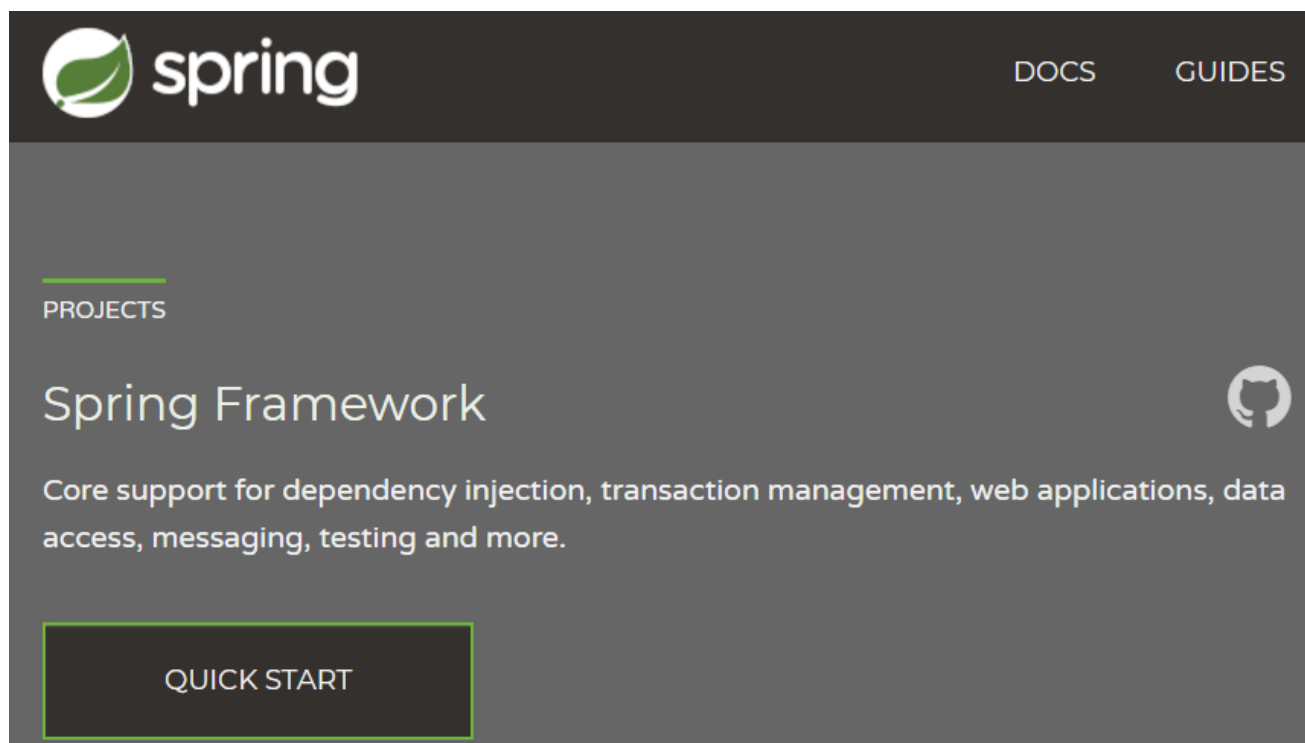
```

小结：使用Jdbc工具类后可以简化代码，我们只需要写SQL去执行。

第3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。

3.2 JdbcTemplate使用过程

3.2.1 C3P0基于配置文件实现连接池

3.2.1.1 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。

3.2.1.2 使用步骤

1. 准备C3P0DataSource连接池
2. 导入依赖的jar包

- `spring-beans-5.0.0.RELEASE.jar`
- `spring-core-5.0.0.RELEASE.jar`
- `spring-jdbc-5.0.0.RELEASE.jar`
- `spring-tx-5.0.0.RELEASE.jar`
- `com.springsource.org.apache.commons.logging-1.2.jar`



3. 创建 `JdbcTemplate` 对象，传入 `C3P0` 连接池
4. 调用 `execute`、`update`、`queryXxx` 等方法

3.2.1.3 案例代码

需求：创建一个学生表id,name,gender,birthday

- 1) id是主键，整数类型，自增长
- 2) name是varchar(20)，非空
- 3) 性别是boolean类型
- 4) 生日是date类型

开发步骤：

- 1) 创建JdbcTemplate对象
- 2) 编写建表的SQL语句
- 3) 使用JdbcTemplate对象的execute()方法执行DDL语句

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);
    }
}

```

3.2.1.4 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)

3.3 JdbcTemplate实现增删改

在JdbcTemplate中执行SQL语句的方法大致分为3类：

JdbcTemplate**中的方法**	功能说明****
execute()	用于执行DDL语句，如：建表
update()	用于执行DML语句，实现增删改操作
queryXxx()	用于执行DQL语句，实现各种查询的操作

3.3.1 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。

3.3.2 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

3.3.3 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);
    }
}
```

```
}
```

3.3.4 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

3.3.5 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。

3.6 JdbcTemplate实现查询

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

3.6.1 queryForInt返回一个整数

3.6.1.1 API介绍

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

3.6.1.2 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

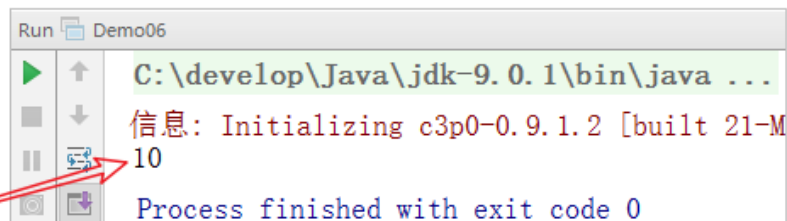
3.6.1.3 案例代码

```
// 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);
}
```

3.6.1.4 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	XH11	18888



```
Run Demo06
C:\develop\Java\jdk-9.0.1\bin\java ...
信息: Initializing c3p0-0.9.1.2 [built 21-M
10
Process finished with exit code 0
```

3.6.2 queryForLong返回一个long类型整数

3.6.2.1 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。

3.6.2.2 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

3.6.2.3 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);
}
```

3.6.2.4 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

3.6.3 queryForObject返回String

3.6.3.1 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。

3.6.3.2 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

3.6.3.3 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);  
}
```

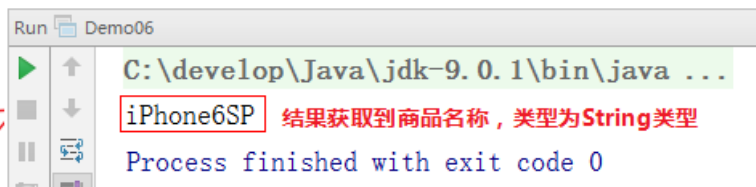
3.6.3.4 案例效果

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	iPhone6S	7000
8	iPhone6SP	7777
9	iPhone6SP	7777
10	XVIII	18888



3.6.4 queryForMap返回一个Map集合对象

3.6.4.1 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。

3.6.4.2 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

3.6.4.3 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);
}

```

3.6.4.4 案例效果

执行查询语句，将一条记录放到一个Map中

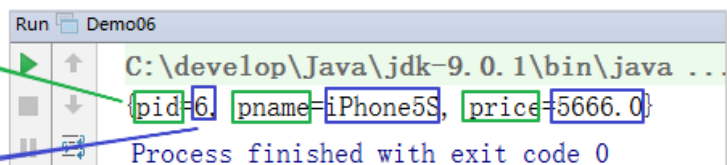
```

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



将这条记录放到一个Map中，key是字段名，value是该字段的值

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

3.6.5.1 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。

3.6.5.2 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

3.6.5.3 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);
    }
}

```

3.6.5.4 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

```

3.6.6 query使用RowMapper做映射返回对象

3.6.6.1 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。

3.6.6.2 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

3.6.6.3 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.setPname(arg0.getString("pname"));
            p.setPrice(arg0.getDouble("price"));
            return p;
        }
    });


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

3.6.6.4 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
```

3.6.7 query使用BeanPropertyRowMapper做映射返回对象

3.6.7.1 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

3.6.7.2 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

3.6.7.3 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));

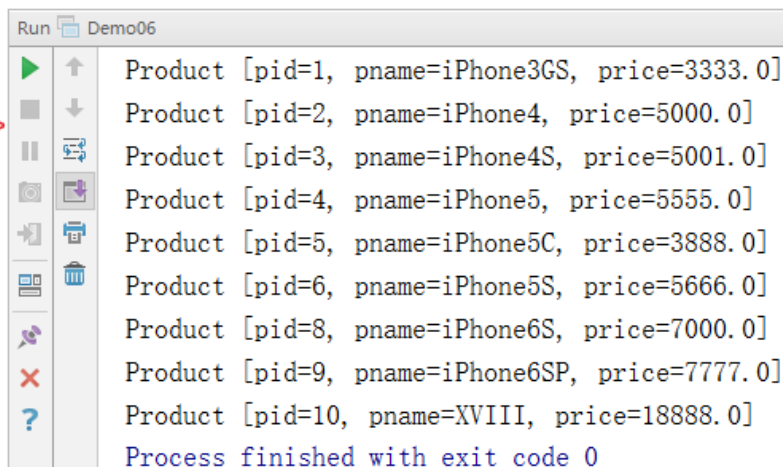
    for (Product product : list) {
        System.out.println(product);
    }
}
```

3.6.6.4 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
```

3.6.8 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。

3.7 JdbcTemplate原理分析

JdbcTemplate的public <T> List<T> query(String sql, RowMapper<T> rowMapper)方法用于返回对象集合，这个方法使用频率相对较多。我们来分析这个类的实现原理。

3.7.1 模拟JdbcTemplate

我们看到在使用 `JdbcTemplate` 的 `query` 方法返回对象，需要用到 `JdbcTemplate` 类、`RowMapper` 接口。我们使用代码来模拟 `JdbcTemplate` 的 `query` 方法。

3.7.1.1 步骤分析

1. 定义 `MyRowMapper` 接口，模拟 `RowMapper`，添加 `mapRow` 方法用于处理查询结果
2. 定义 `MyJdbcTemplate` 类，模拟 `JdbcTemplate` 类的功能
3. 在 `MyJdbcTemplate` 定义一个构造方法，参数为 `DataSource`
4. 在 `MyJdbcTemplate` 定义 `query` 方法，用于查询数据，返回对象集合
5. 定义测试类，编写SQL语句，使用 `MyJdbcTemplate` 的 `query` 方法来查询数据

3.7.1.2 案例代码

MyRowMapper.java

```
public interface MyRowMapper<T> {  
    /*  
    用于处理一行结果  
    */  
    T mapRow(ResultSet rs) throws SQLException;  
}
```

MyJdbcTemplate.java

```
public class MyJdbcTemplate {  
    /*  
    连接池，拿到里面的连接执行SQL语句  
    */  
    private DataSource dataSource;  
  
    public MyJdbcTemplate(DataSource dataSource) {  
        this.dataSource = dataSource;  
    }  
  
    /*  
    查询数据，返回一个List集合，集合中保存对象  
    */  
    public <T> List<T> query(String sql, MyRowMapper<T> rowMapper) throws Exception {  
        // 获取连接  
        Connection conn = dataSource.getConnection();  
        // 创建Statement对象  
        Statement stmt = conn.createStatement();  
        // 执行SQL语句，获取查询结果集  
        ResultSet rs = stmt.executeQuery(sql);  
  
        // 定义List集合，用于存储查询到的数据  
        ArrayList<T> list = new ArrayList<>();  
        // 循环判断结果集中是否还有数据  
        while (rs.next()) {  
            // 结果集中有数据，调用rowMapper的mapRow方法处理这行数据  
            T obj = rowMapper.mapRow(rs);  
  
            // 将mapRow处理后的对象添加到list集合中
```

```

        list.add(obj);
    }

    return list;
}
}

```

测试类

```

public class Demo07 {
    public static void main(String[] args) throws Exception {
        // 编写SQL语句
        String sql = "SELECT * FROM product WHERE pid<5;";

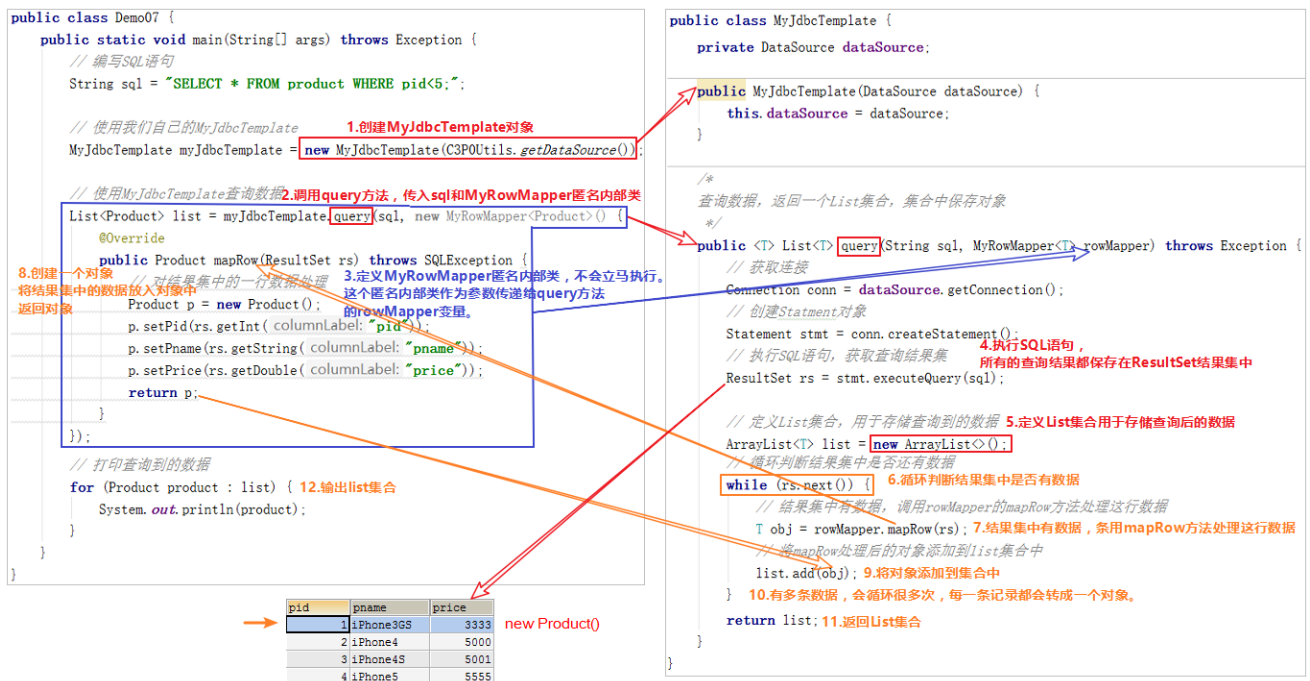
        // 使用我们自己的MyJdbcTemplate
        MyJdbcTemplate myJdbcTemplate = new MyJdbcTemplate(JdbcUtils.getDataSource());

        // 使用MyJdbcTemplate查询数据
        List<Product> list = myJdbcTemplate.query(sql, new MyRowMapper<Product>() {
            @Override
            public Product mapRow(ResultSet rs) throws SQLException {
                // 对结果集中的一行数据处理
                Product p = new Product();
                p.setPid(rs.getInt("pid"));
                p.setPname(rs.getString("pname"));
                p.setPrice(rs.getDouble("price"));
                return p;
            }
        });

        // 打印查询到的数据
        for (Product product : list) {
            System.out.println(product);
        }
    }
}

```

3.7.1.3 执行流程分析



3.7.2 JdbcTemplate源码分析

由于版面有限，一些不重要的代码没有贴出来。主要涉及到 JdbcTemplate、RowMapper、RowMapperResultSetExtractor、QueryStatementCallback 类 源码流程：

1. 编写SQL语句
2. 调用 JdbcTemplate 类的 query 方法，传入SQL语句和 RowMapper的匿名内部类
3. query方法调用 JdbcTemplate 类中重载的query方法，并传入了一个 RowMapperResultSetExtractor 对象
4. RowMapperResultSetExtractor 类中保存了前面传入的 RowMapper的匿名内部类
5. JdbcTemplate 类的重载 query`方法里面
 - 定义了一个局部内部类 QueryStatementCallback，重写了 doInStatement 方法。
 - 调用了 execute 方法，传入了 QueryStatementCallback 对象
6. JdbcTemplate 类的 execute`方法内部
 - 获取连接
 - 创建Statement对象
 - 调用 doInStatement 方法
7. 回到 QueryStatementCallback 类的 doInStatement 方法，执行SQL语句，获取到结果集，调用 RowMapperResultSetExtractor 类的 extractData 方法
8. RowMapperResultSetExtractor 类的 extractData 方法中判断结果集中是否有数据，如果有数据就调用 RowMapper 匿名内部类的 mapRow 方法。
9. mapRow 方法创建一个Product对象，将结果集中的数据放到Product对象中，返回Product对象。
10. RowMapperResultSetExtractor 类的 extractData 拿到返回的数据放到list集合中，返回list集合
11. 最终query返回list集合

```
String sql = "SELECT * FROM product"; // 1. 编写SQL语句
List<Product> query = jdbcTemplate.query(sql, new RowMapper<Product>() {
    @Override
    public Product mapRow(ResultSet arg0, int arg1) throws SQLException {
        // 9.1 创建一个Product对象
        Product p = new Product();
        p.setPid(arg0.getInt(columnLabel: "pid"));
        p.setPname(arg0.getString(columnLabel: "pname"));
        p.setPrice(arg0.getDouble(columnLabel: "price"));
        // 9.2 将结果集中的数据放到Product对象中
        return p; // 9.3 返回Product对象
    }
});
```

```
public class RowMapperResultSetExtractor<T> implements ResultSetExtractor<List<T>> {
    private final RowMapper<T> rowMapper;
    // 4. RowMapperResultSetExtractor类中保存了前面传入的RowMapper的匿名内部类
    public RowMapperResultSetExtractor(RowMapper<T> rowMapper) {
        this(rowMapper, rowsExpected: 0);
    }

    public RowMapperResultSetExtractor(RowMapper<T> rowMapper, int rowsExpected) {
        Assert.notNull(rowMapper, message: "RowMapper is required");
        this.rowMapper = rowMapper;
        this.rowsExpected = rowsExpected;
    }

    @Override
    public List<T> extractData(ResultSet rs) throws SQLException {
        List<T> results = (this.rowsExpected > 0 ?
            new ArrayList<T>(this.rowsExpected) : new ArrayList<T>());

        int rowNum = 0; // 8.1 判断结果集中是否有数据
        while (rs.next()) {
            // 8.2 如果有数据就调用RowMapper匿名内部类的mapRow方法。
            results.add(this.rowMapper.mapRow(rs, rowNum++));
            // 10.1 拿到返回的数据放到list集合中
        }
        // 10.2 返回list集合
        return results;
    }
}
```

```
public class JdbcTemplate extends JdbcAccessor implements JdbcOperations {
    public <T> List<T> query(String sql, RowMapper<T> rowMapper) {
        return query(sql, new RowMapperResultSetExtractor<T>(rowMapper));
    }
    // 3. query方法调用JdbcTemplate类中重载的query方法，并传入了一个RowMapperResultSetExtractor对象

    public <T> T query(final String sql, final ResultSetExtractor<T> rse) {
        // 5.1 定义了一个局部内部类QueryStatementCallback，重写了doInStatement方法
        class QueryStatementCallback implements StatementCallback<T>, SqlProvider {
            public T doInStatement(Statement stmt) throws SQLException {
                // 7.1 返回QueryStatementCallback类的doInStatement方法
                ResultSet rs = null;
                try {
                    // 7.2 执行SQL语句，获取到结果集
                    rs = stmt.executeQuery(sql);
                    ResultSet rsToUse = rs;
                    // 7.3 调用RowMapperResultSetExtractor类的extractData方法
                    return rse.extractData(rsToUse);
                } catch (SQLException ex) {
                    // 5.2 调用了execute方法，传入了QueryStatementCallback对象
                    return execute(new QueryStatementCallback());
                }
            }
        }

        // 6.1 获取连接
        Connection con = DataSourceUtils.getConnection(getDataSource());
        Statement stmt = null;
        try {
            Connection conToUse = con; // 6.2 创建Statement对象
            stmt = conToUse.createStatement();
            Statement stmtToUse = stmt; // 6.3 调用doInStatement方法
            T result = action.doInStatement(stmtToUse);
            // 11.1 返回列表集合
            return result;
        } catch (SQLException ex) {
            JdbcUtils.closeStatement(stmt);
        } finally {
            JdbcUtils.closeStatement(stmt);
            DataSourceUtils.releaseConnection(con, getDataSource());
        }
    }
}
```