

连接池

学习目标

1. 能够通过PreparedStatement完成增、删、改、查
2. 能够完成PreparedStatement改造登录案例
3. 能够理解连接池解决现状问题的原理
4. 能够说出动态代理的好处
5. 能够使用动态代理
6. 能够使用C3P0连接池
7. 能够使用DRUID连接池
8. 能够编写C3P0连接池工具类

第1章 PreparedStatement预编译对象

1.1 SQL注入问题

在我们前一天JDBC实现登录案例中，当我们输入以下密码，我们发现我们账号和密码都不对竟然登录成功了

```
请输入用户名：
hehe
请输入密码：
a' or '1'='1
```

问题分析：

```
// 代码中的SQL语句
"SELECT * FROM user WHERE name='" + name + "' AND password='" + password + "';";
// 将用户输入的账号密码拼接后
"SELECT * FROM user WHERE name='hehe' AND password='a' or '1'='1';"
```

我们让用户输入的密码和SQL语句进行字符串拼接。用户输入的内容作为了SQL语句语法的一部分，改变了原有SQL真正的意义，以上问题称为SQL注入。

要解决SQL注入就不能让用户输入的密码和我们的SQL语句进行简单的字符串拼接。需要使用PreparedStatement类解决SQL注入。

1.2 PreparedStatement的执行原理

继承结构：



java.sql 接口 PreparedStatement

所有超级接口：

[Statement](#), [Wrapper](#)

所有已知子接口：

[CallableStatement](#)

```
public interface PreparedStatement  
extends Statement
```

表示预编译的 SQL 语句的对象。

我们写的SQL语句让数据库执行，数据库不是直接执行SQL语句字符串。和Java一样，数据库需要执行编译后的SQL语句（类似Java编译后的字节码文件）。

1. `Statement` 对象每执行一条SQL语句都会先将这条SQL语句发送给数据库编译，数据库再执行。

```
Statement stmt = conn.createStatement();  
stmt.executeUpdate("INSERT INTO users VALUES (1, '张三', '123456');");  
stmt.executeUpdate("INSERT INTO users VALUES (2, '李四', '666666');");
```

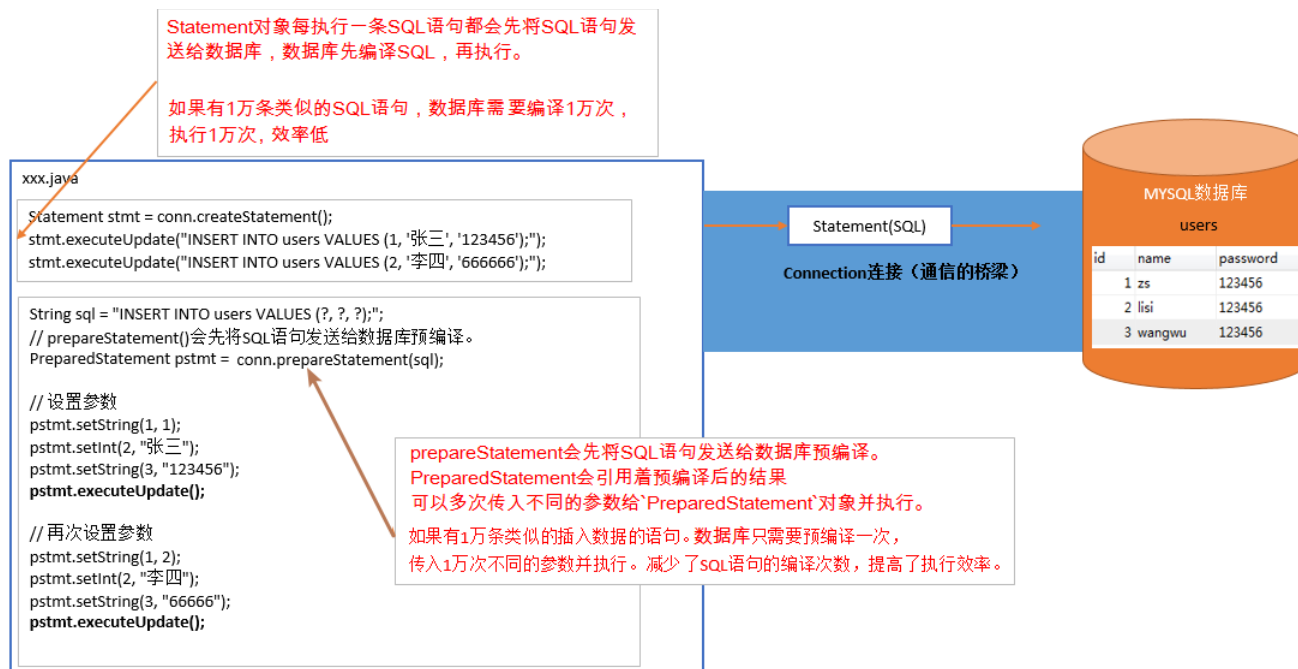
上面2条SQL语句我们可以看到大部分内容是相同的，只是数据略有不一样。数据库每次执行都编译一次。如果有1万条类似的SQL语句，数据库需要编译1万次，执行1万次，显然效率就低了。

1. `prepareStatement()` 会先将SQL语句发送给数据库预编译。`PreparedStatement` 会引用着预编译后的结果。可以多次传入不同的参数给 `PreparedStatement` 对象并执行。相当于调用方法多次传入不同的参数。

```
String sql = "INSERT INTO users VALUES (?, ?, ?);";  
// 会先将SQL语句发送给数据库预编译。PreparedStatement会引用着预编译后的结果。  
PreparedStatement pstmt = conn.prepareStatement(sql);  
  
// 设置参数  
pstmt.setString(1, 1);  
pstmt.setInt(2, "张三");  
pstmt.setString(3, "123456");  
pstmt.executeUpdate();  
  
// 再次设置参数  
pstmt.setString(1, 2);  
pstmt.setInt(2, "李四");  
pstmt.setString(3, "666666");  
pstmt.executeUpdate();
```

上面预编译好一条SQL，2次传入了不同的参数并执行。如果有1万条类似的插入数据的语句。数据库只需要预编译一次，传入1万次不同的参数并执行。减少了SQL语句的编译次数，提高了执行效率。

示意图



1.3 PreparedStatement的好处

1. `prepareStatement()` 会先将SQL语句发送给数据库预编译。`PreparedStatement` 会引用着预编译后的结果。可以多次传入不同的参数给 `PreparedStatement` 对象并执行。减少SQL编译次数，提高效率。
2. 安全性更高，没有SQL注入的隐患。
3. 提高了程序的可读性

1.4 PreparedStatement的基本使用

1.4.1 API介绍

1.4.1.1 获取PreparedStatement的API介绍

在 `java.sql.Connection` 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。

1.4.1.2 PreparedStatement的API介绍

在 `java.sql.PreparedStatement` 中有设置SQL语句参数，和执行参数化的SQL语句的方法

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

2. `void setFloat(int parameterIndex, float x)`
将指定参数设置为给定 Java `REAL` 值。



3. `void setInt(int parameterIndex, int x)`
将指定参数设置为给定 Java `int` 值。
4. `void setLong(int parameterIndex, long x)`
将指定参数设置为给定 Java `long` 值。
5. `void setObject(int parameterIndex, Object x)`
使用给定对象设置指定参数的值。
6. `void setString(int parameterIndex, String x)`
将指定参数设置为给定 Java `String` 值。
7. `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。
8. `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。

1.4.2 PreparedStatement使用步骤

1. 编写SQL语句，未知内容使用?占位：`"SELECT * FROM user WHERE name=? AND password=?;"`
2. 获得PreparedStatement对象
3. 设置实际参数
4. 执行参数化SQL语句
5. 关闭资源

1.4.3 案例代码

```
public class Demo08 {  
  
    public static void main(String[] args) throws Exception {  
        // 获取连接  
        Connection conn = JDBCUtils.getConnection();  
  
        // 编写SQL语句，未知内容使用?占位  
        String sql = "SELECT * FROM user WHERE name=? AND password=?";  
  
        // preparedStatement()会先将SQL语句发送给数据库预编译。  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
  
        // 指定?的值  
        // parameterIndex: 第几个?, 从1开始算  
        // x: 具体的值  
        pstmt.setString(1, "admin");  
  
        pstmt.setString(2, "123"); // 正确的密码
```

```
// pstmt.setString(2, "6666"); // 错误的密码

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {
    String name = rs.getString("name");
    System.out.println("name: " + name);
} else {
    System.out.println("没有找到数据...");
}

JDBCUtils.close(conn, pstmt, rs);
}
```

1.4.4 案例效果

1. 输入正确的账号密码:

输入正确的账号密码

```
24      pstmt.setString( parameterIndex: 1, x: "admin");
25      pstmt.setString( parameterIndex: 2, x: "123");
```

Run Demo08

▶

↑

↓

⏸

🔄

📷

📄

C:\develop\Java\jdk-9.0.1\bin\java "-javaager
查询到数据:
id: 1 , name: admin , password: 123 查询到数据
Process finished with exit code 0

2. 输入错误的密码:

输入错误的密码

```
24      pstmt.setString( parameterIndex: 1, x: "admin");
25      pstmt.setString( parameterIndex: 2, x: "6666");
```

Run Demo08

▶

↑

↓

⏸

🔄

📷

📄

C:\develop\Java\jdk-9.0.1\bin\java "-javaager
没有找到数据...
错误的密码找不到数据
Process finished with exit code 0

1.5 PreparedStatement实现增删查改

1.5.1 添加数据

向Employee表添加3条记录



```
// 添加数据：向Employee表添加3条记录
public static void addEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();
    String sql = "INSERT INTO employee VALUES (NULL, ?, ?, ?)";
    // preparedStatement()会先将SQL语句发送给数据库预编译。
    PreparedStatement pstmt = conn.prepareStatement(sql);

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

    // 再次设置参数
    pstmt.setString(1, "张学友");
    pstmt.setInt(2, 55);
    pstmt.setString(3, "澳门");
    i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    // 再次设置参数
    pstmt.setString(1, "黎明");
    pstmt.setInt(2, 52);
    pstmt.setString(3, "香港");
    i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    JDBCUtils.close(conn, pstmt);
}
```

效果：

id	name	age	address
1	刘德华	57	香港
2	张学友	55	澳门
3	黎明	52	香港

1.5.2 修改数据

将id为2的学生地址改成台湾



// 修改数据：将id为2的学生地址改成台湾

```
public static void updateEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();

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

    JDBCUtils.close(conn, pstmt);
}
```

效果:

id	name	age	address
1	刘德华	57	香港
2	张学友	55	台湾
3	黎明	52	香港

1.5.3 删除数据

删除id为2的员工

// 删除数据：删除id为2的员工

```
public static void deleteEmployee() throws Exception {
    Connection conn = JDBCUtils.getConnection();

    String sql = "DELETE FROM employee WHERE id=?";
    PreparedStatement pstmt = conn.prepareStatement(sql);
    pstmt.setInt(1, 2);
    int i = pstmt.executeUpdate();
    System.out.println("影响的行数:" + i);

    JDBCUtils.close(conn, pstmt);
}
```

效果:

id	name	age	address
1	刘德华	57	香港
3	黎明	52	香港

1.5.4 查询数据

查询id小于8的员工信息,并保存到员工类中

```
public class Employee {
    private int id;
    private String name;
    private int age;
    private String address;
```



```
public Employee() {  
}  
  
public Employee(int id, String name, int age, String address) {  
    this.id = id;  
    this.name = name;  
    this.age = age;  
    this.address = address;  
}  
  
public int getId() {  
    return id;  
}  
  
public void setId(int id) {  
    this.id = id;  
}  
  
public String getName() {  
    return name;  
}  
  
public void setName(String name) {  
    this.name = name;  
}  
  
public int getAge() {  
    return age;  
}  
  
public void setAge(int age) {  
    this.age = age;  
}  
  
public String getAddress() {  
    return address;  
}  
  
public void setAddress(String address) {  
    this.address = address;  
}  
  
@Override  
public String toString() {  
    return "Employee2 [id=" + id + ", name=" + name + ", age=" + age + ", address=" +  
address + "];"  
}  
}
```

```
// 查询数据: 查询id小于8的员工信息,并保存到员工类中  
public static void queryEmployee() throws Exception {  
    Connection conn = JDBCUtils.getConnection();  
    String sql = "SELECT * FROM employee WHERE id<?";  
}
```




```
PreparedStatement pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, 26);
ResultSet rs = pstmt.executeQuery();

// 创建集合存放多个Employee2对象
ArrayList<Employee> list = new ArrayList<>();

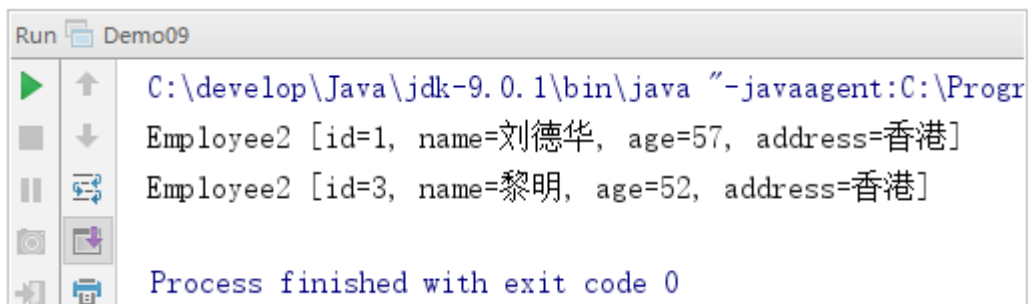
while (rs.next()) {
    // 移动到下一行有数据,取出这行数据
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int age = rs.getInt("age");
    String address = rs.getString("address");

    // 创建Employee2对象
    Employee e = new Employee(id, name, age, address);
    // 将创建好的员工添加到集合中
    list.add(e);
}

// 输出对象
for (Employee e : list) {
    System.out.println(e);
}

JDBCUtils.close(conn, pstmt, rs);
}
```

效果:



1.6 案例：PreparedStatement改造登录案例

1.6.1 案例需求

模拟用户输入账号和密码登录网站，防止SQL注入

1.6.2 案例效果



1. 输入正确的账号，密码，显示登录成功

请输入账号：

admin

输入正确的账号，密码

请输入密码：

123

登录成功

欢迎您,admin

2. 输入错误的账号，密码，显示登录失败

请输入账号：

admin

输入错误的账号或密码

请输入密码：

显示登录失败

aaa

账号或密码错误...

3. 输入 "a' or '1'='1'" 作为密码，解决SQL注入：

```

24      pstmt.setString( parameterIndex: 1, x: "admin");
25      pstmt.setString( parameterIndex: 2, x: "a' or '1'='1'");
Run Demo08
C:\develop\Java\jdk-9.0.1\bin\java "-javaager
没有找到数据...
没有找到数据，解决了SQL注入的问题
Process finished with exit code 0
    
```

1.6.3 案例分析

1. 使用数据库保存用户的账号和密码
2. 让用户输入账号和密码
3. 编写SQL语句，账号和密码部分使用？占位
4. 使用PreparedStatement给？设置参数
5. 使用PreparedStatement执行预编译的SQL语句
6. 如果查询到数据，说明登录成功
7. 如果查询不到数据，说明登录失败

1.6.4 实现步骤

1. 编写代码让用户输入账号和密码

```

public class Demo07 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("请输入账号：");
        String name = sc.nextLine();
        System.out.println("请输入密码：");
        String password = sc.nextLine();
    }
}
    
```



2. 编写SQL语句，账号和密码部分使用？占位，使用PreparedStatement给？设置参数，使用PreparedStatement执行预编译的SQL语句

```
public class Demo11 {  
    public static void main(String[] args) throws Exception {  
        // 让用户输入账号和密码  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
  
        // 获取连接  
        Connection conn = JDBCUtils.getConnection();  
        // 编写SQL语句，账号和密码使用？占位  
        String sql = "SELECT * FROM user WHERE name=? AND password=?";  
        // 获取到PreparedStatement对象  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        // 设置参数  
        pstmt.setString(1, name);  
        pstmt.setString(2, password);  
        // pstmt.setString(2, "a' or '1'='1");  
    }  
}
```

3. 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败

```
public class Demo11 {  
    public static void main(String[] args) throws Exception {  
        // 让用户输入账号和密码  
        Scanner sc = new Scanner(System.in);  
        System.out.println("请输入账号: ");  
        String name = sc.nextLine();  
        System.out.println("请输入密码: ");  
        String password = sc.nextLine();  
  
        // 获取连接  
        Connection conn = JDBCUtils.getConnection();  
        // 编写SQL语句，账号和密码使用？占位  
        String sql = "SELECT * FROM user WHERE name=? AND password=?";  
        // 获取到PreparedStatement对象  
        PreparedStatement pstmt = conn.prepareStatement(sql);  
        // 设置参数  
        pstmt.setString(1, name);  
        pstmt.setString(2, password);  
        // pstmt.setString(2, "a' or '1'='1");  
  
        // 如果查询到数据，说明登录成功，如果查询不到数据，说明登录失败  
        ResultSet rs = pstmt.executeQuery();  
  
        if (rs.next()) {  
            // 能进来查询到了数据。  
        }  
    }  
}
```

```
String name2 = rs.getString("name");
System.out.println("欢迎您," + name2);
} else {
    // 查询不到数据, 说明登录失败
    System.out.println("账号或密码错误...");
}

JDBCUtils.close(conn, pstmt, rs);
}
}
```

第2章 自定义连接池和动态代理

2.1 连接池概念

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。

连接池的概念: 连接池是创建和管理数据库连接的缓冲池技术。连接池就是一个容器，连接池中保存了一些数据库连接，这些连接是可以重复使用的。

2.2 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

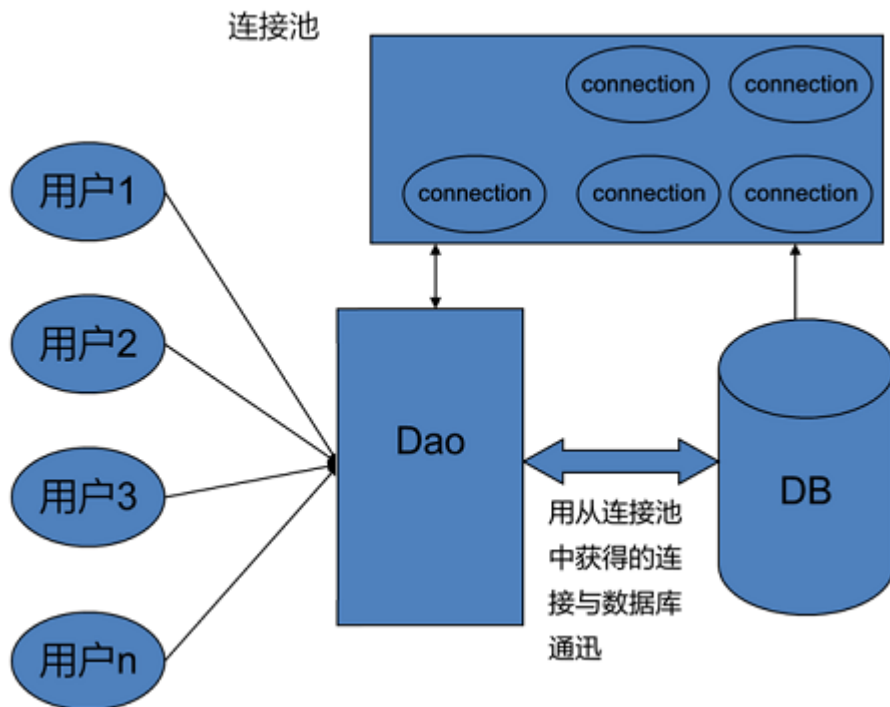


2. 每次创建数据库连接的问题

- 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

2.3 连接池解决现状问题的原理

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。



2.4 自定义连接池

2.4.1 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。

2.4.2 自定义连接池步骤

1. 定义一个类实现 `javax.sql.DataSource` 接口
2. 实现接DataSource口中的抽象方法
3. 定义连接池相关参数
4. 创建容器保存连接
5. 提供获取连接方法
6. 提供关闭连接方法

2.4.3 实现步骤

1. 定义一个类实现 `javax.sql.DataSource` 接口

```
import javax.sql.DataSource;
public class MyPool implements DataSource {

}
```

2. 实现接DataSource口中的抽象方法



```
public class MyPool implements DataSource {  
    @Override  
    public Connection getConnection() throws SQLException {  
        return null;  
    }  
  
    @Override  
    public Connection getConnection(String username, String password) throws SQLException {  
        return null;  
    }  
  
    @Override  
    public PrintWriter getLogWriter() throws SQLException {  
        return null;  
    }  
  
    @Override  
    public void setLogWriter(PrintWriter out) throws SQLException {  
  
    }  
  
    @Override  
    public void setLoginTimeout(int seconds) throws SQLException {  
  
    }  
  
    @Override  
    public int getLoginTimeout() throws SQLException {  
        return 0;  
    }  
  
    @Override  
    public Logger getParentLogger() throws SQLFeatureNotSupportedException {  
        return null;  
    }  
  
    @Override  
    public <T> T unwrap(Class<T> iface) throws SQLException {  
        return null;  
    }  
  
    @Override  
    public boolean isWrapperFor(Class<?> iface) throws SQLException {  
        return false;  
    }  
}
```

3. 定义连接池相关参数 连接池必要的因素：

- 初始化连接个数
- 最大的连接个数
- 当前已经创建的连接个数



```
public class MyPool implements DataSource {  
    private int initCount = 5; //初始化的个数  
    private int maxCount = 10; //最大的连接个数  
    private int curCount = 0; //当前已经创建的连接个数  
    // 其他方法省略  
}
```

4. 创建容器保存连接

```
public class MyPool implements DataSource {  
    private int initCount = 5; //初始化的个数  
    private int maxCount = 10; //最大的连接个数  
    private int curCount = 0; //当前已经创建的连接个数  
  
    // 存储连接的容器  
    private LinkedList<Connection> list = new LinkedList<Connection>();  
  
    // 构造方法,初始化连接池的, 连接池一旦创建那么连接池中就应该要有指定个数的连接。  
    public MyPool() throws SQLException {  
        for (int i = 0; i < initCount; i++) {  
            Connection connection = createConnection();  
            // 把连接存储到容器中  
            list.add(connection);  
        }  
    }  
  
    // 创建Connection的方法  
    public Connection createConnection() throws SQLException {  
        Connection connection = JDBCUtils.getConnection();  
        curCount++;  
        return connection;  
    }  
}
```

5. 提供获取连接方法,提供关闭连接方法

```
public class MyPool implements DataSource {  
    private int initCount = 5; //初始化的个数  
    private int maxCount = 10; //最大的连接个数  
    private int curCount = 0; //当前已经创建的连接个数  
  
    // 存储连接的容器  
    private LinkedList<Connection> list = new LinkedList<Connection>();  
  
    // 构造方法,初始化连接池的, 连接池一旦创建那么连接池中就应该要有指定个数的连接。  
    public MyPool() throws SQLException {  
        for (int i = 0; i < initCount; i++) {  
            Connection connection = createConnection();  
            // 把连接存储到容器中  
            list.add(connection);  
        }  
    }  
}
```



```
// 创建Connection的方法
public Connection createConnection() throws SQLException {
    Connection connection = JDBCUtils.getConnection();
    curCount++;
    return connection;
}

// 别人问你的连接池要连接
@Override
public Connection getConnection() throws SQLException {
    // 情况一：先判断连接池是否有连接
    if (list.size() > 0) {
        //如果连接池有连接，那么直接取出连接，返回即可
        Connection connection = list.removeFirst();
        return connection;
    }

    // 情况二：连接池中无连接，然后先判断目前的连接个数是否已经超过了最大的连接个数
    if (curCount < maxCount) {
        // 创建连接
        Connection connection = createConnection();
        return connection;
    } else {
        // 没有连接，并且已经超过了最大的连接个数
        throw new RuntimeException("已经达到了最大的连接个数，请稍后");
    }
}

// 回收Connection
public void close(Connection connection) {
    list.addLast(connection);
}
}
```

6. 编写测试类

```
public class Demo04 {
    public static void main(String[] args) {
        // 创建自定义的连接池
        MyPool myPool = new MyPool();
        System.out.println("连接1: " + myPool.getConnection());
        System.out.println("连接2: " + myPool.getConnection());
        System.out.println("连接3: " + myPool.getConnection());
        System.out.println("连接4: " + myPool.getConnection());
        System.out.println("连接5: " + myPool.getConnection());
        System.out.println("连接6: " + myPool.getConnection());
        System.out.println("连接7: " + myPool.getConnection());
        System.out.println("连接8: " + myPool.getConnection());
        System.out.println("连接9: " + myPool.getConnection());
        Connection connection = myPool.getConnection();
        System.out.println("连接10: " + connection);
    }
}
```




```
/*
    目前的问题： 由于我们编程习惯的问题，我们都习惯关闭资源的时候直接调用close方法，不习惯传递
    进去。

    目标： 如果连接需要归还给连接池，那么直接调用close方法即可。
*/

// 把连接还回给连接池
myPool.close(connection);

System.out.println("连接11: " + myPool.getConnection());
}
}
```

7. 运行效果

```
public class Demo04 {
    public static void main(String[] args) {
        // 创建自定义的连接池
        MyPool myPool = new MyPool();
        System.out.println("连接1: " + myPool.getConnection());
        System.out.println("连接2: " + myPool.getConnection());
        System.out.println("连接3: " + myPool.getConnection());
        System.out.println("连接4: " + myPool.getConnection());
        System.out.println("连接5: " + myPool.getConnection());
        System.out.println("连接6: " + myPool.getConnection());
        System.out.println("连接7: " + myPool.getConnection());
        System.out.println("连接8: " + myPool.getConnection());
        System.out.println("连接9: " + myPool.getConnection());
        Connection connection = myPool.getConnection();
        System.out.println("连接10: " + connection);
        // 把连接还回给连接池
        myPool.close(connection);
        System.out.println("连接11: " + myPool.getConnection());
    }
}
```

连接1: com.mysql.jdbc.JDBC4Connection@7b9a4292
 连接2: com.mysql.jdbc.JDBC4Connection@4a94ee4
 连接3: com.mysql.jdbc.JDBC4Connection@4cc451f2
 连接4: com.mysql.jdbc.JDBC4Connection@6379eb
 连接5: com.mysql.jdbc.JDBC4Connection@294425a7
 连接6: com.mysql.jdbc.JDBC4Connection@6572421
 连接7: com.mysql.jdbc.JDBC4Connection@6c64cb25
 连接8: com.mysql.jdbc.JDBC4Connection@3a93b025
 连接9: com.mysql.jdbc.JDBC4Connection@36902638
 连接10: com.mysql.jdbc.JDBC4Connection@77847718
 连接11: com.mysql.jdbc.JDBC4Connection@77847718

将第10个连接放回连接池
 从连接池中取出连接，取到的是刚刚放回的那个连接

8. 自定义连接池存在的问题

由于我们编程习惯的问题，我们都习惯关闭资源的时候直接调用close方法，不习惯传作为参数递进去。

目标：如果连接需要归还给连接池，那么直接调用close方法即可。 connection.close() 方法用于将连接还回连接池，而不是关闭连接。我们可以使用动态代理来增强Connection的close方法，变成将连接还回连接池。

2.5 代理模式

2.5.1 现实生活中的代理

假如我们要去买电脑，我们通常需要去找电脑代理商购买，电脑代理商去电脑工厂提货，电脑代理商可以赚取中间的差价。假如我们想买火车票，我们可以直接去12306网站买票。可是太难抢到票了，于是我们去找火车票的黄牛，让黄牛帮我们去12306买票，黄牛买到票再加价卖给我们，赚取中间的差价。

你买电脑	->	电脑代理商	->	电脑工厂
你买火车票	->	黄牛	->	12306
调用者		代理对象		真正干活的目标对象

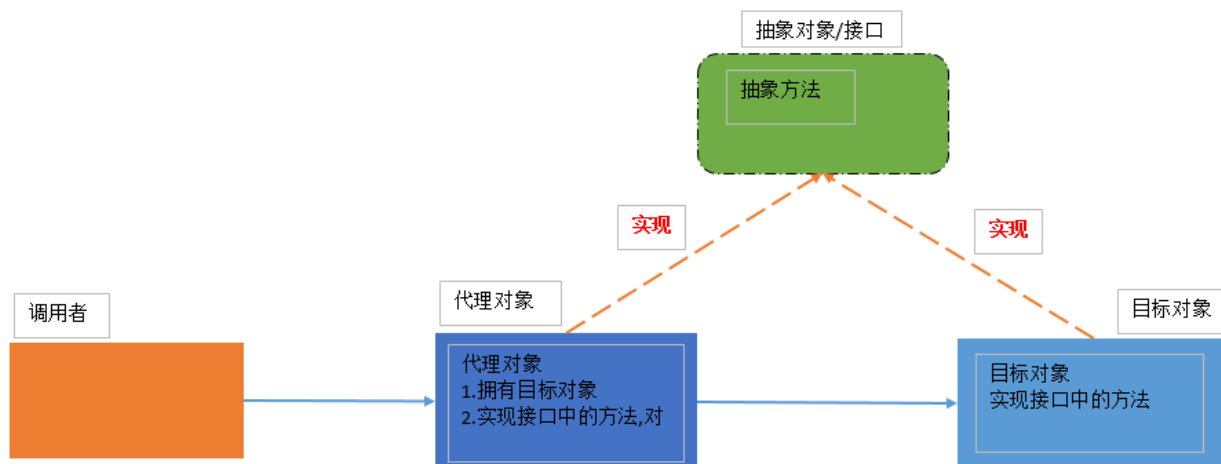
我们发现 代理对象 和 真正干活的目标 都具有相同的功能（卖电脑/卖票），代理可以在中间赚取差价（增强功能）。

2.5.2 代理模式的作用

代理对象可以在 调用者 和 目标对象 之间起到中介的作用。代理对象可以对 目标对象 的功能进行 增强。

2.5.3 代理模式涉及到四个要素

1. 调用者: 你。
2. 代理对象: 联想电脑代理商/黄牛。
3. 目标对象: 电脑工厂/12306。
4. 抽象对象: 代理对象和目标对象都共有的接口,保证代理对象和真实对象都有相应的方法, 如电脑代理商和电脑工厂都需要有卖电脑的功能。



2.5.4 动态代理

2.5.4.1 什么是动态代理

在程序运行的过程中, 动态创建出代理对象。

2.5.4.2 动态代理类相应的API:

1. Proxy类

```
static Object newProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler h)
```

作用: 生成实现指定接口的代理对象

参数说明:

loader参数: 目标对象的类加载器

interfaces: 代理对象实现的接口数组

h: 具体的代理操作, InvocationHandler是一个接口, 需要传入一个实现了此接口的实现类。

返回值: 实现指定接口的代理对象。

2. InvocationHandler接口



`Object invoke(Object proxy, Method method, Object[] args)`

作用：在这个方法中实现对真实方法的增强

参数说明：

`proxy`：即方法`newProxyInstance()`方法返回的代理对象，该对象一般不要在`invoke`方法中使用。

`method`：代理对象调用的方法对象。

`args`：代理对象调用方法时传递的参数。

返回值：是真实对象方法的返回值。

2.5.4.3 动态代理模式的开发步骤

1. 直接创建真实对象
2. 通过Proxy类，创建代理对象
3. 调用代理方法
4. 在InvocationHandler的invoke进行处理

2.5.4.4 动态代理模式使用

The screenshot shows a Java IDE with the following code:

```
public class Demo05 {
    public static void main(String[] args) {
        // 1. 直接创建真实对象
        ComputerFactory computerFactory = new ComputerFactory();
        // 2. 通过Proxy类，创建代理对象
        Providable proxy = (Providable) Proxy.newProxyInstance(
            ComputerFactory.class.getClassLoader(),
            new Class[] { Providable.class },
            new MyInvocationHandler(computerFactory)
        );
        // 3. 调用代理方法
        // 1.调用代理对象的方法，会执行InvocationHandler的invoke方法
        proxy.sellComputer( price: 5000);
    }

    // InvocationHandler实现类
    class MyInvocationHandler implements InvocationHandler {
        private ComputerFactory computerFactory;

        public MyInvocationHandler(ComputerFactory computerFactory) {
            this.computerFactory = computerFactory;
        }

        // 4. 在InvocationHandler的 invoke 进行处理
        @Override
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
            System.out.println("method = " + method);
            System.out.println("args = " + Arrays.toString(args));
            return null;
        }
    }
}
```

Annotations on the code:

- Red arrow from "1.调用代理对象的方法，会执行InvocationHandler的invoke方法" points to the `proxy.sellComputer` call.
- Red arrow from "2.method就是代理调用的方法对象" points to the `method` parameter in the `invoke` method.
- Red arrow from "3.args就是代理调用方法是传入的参数" points to the `args` parameter in the `invoke` method.

Run output (Demo05):

```
C:\develop\Java\jdk-9.0.1\bin\java "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA
method = public abstract void com.itheima.demo05动态代理.Providable.sellComputer(double)
args = [5000.0]
```

1. Providable接口

```
/*
    提供商品的接口
*/
public interface Providable {
    // 卖电脑
    public abstract void sellComputer(double price);
    // 维修电脑
    public abstract void repairComputer(double price);
}
```



2. ComputerFactory目标对象

```
/*  
    电脑工厂，真正生产电脑的厂商  
*/  
public class ComputerFactory implements Providable {  
    @Override  
    public void sellComputer(double price) {  
        System.out.println("电脑工厂卖出一台电脑，价格： " + price);  
    }  
  
    @Override  
    public void repairComputer(double price) {  
        System.out.println("电脑工厂修好一台电脑，价格： " + price);  
    }  
}
```

3. 调用者，动态生成代理对象

```
public class Demo05 {  
    public static void main(String[] args) {  
        // 1. 直接创建真实对象  
        ComputerFactory computerFactory = new ComputerFactory();  
        // 2. 通过Proxy类，创建代理对象  
        Providable proxy = (Providable) Proxy.newProxyInstance(  
            ComputerFactory.class.getClassLoader(),  
            new Class[] {Providable.class},  
            new MyInvocationHandler(computerFactory)  
        );  
  
        // 3. 调用代理方法  
        proxy.sellComputer(5000);  
        proxy.repairComputer(800);  
    }  
}  
  
// InvocationHandler实现类  
class MyInvocationHandler implements InvocationHandler {  
    private ComputerFactory computerFactory;  
  
    public MyInvocationHandler(ComputerFactory computerFactory) {  
        this.computerFactory = computerFactory;  
    }  
  
    // 4. 在InvocationHandler的invoke进行处理  
    @Override  
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {  
        if (method.getName().equals("sellComputer")) { // 对于是要增强的方法进行增强  
            // 代理对象处理卖电脑的方法。  
            double price = (double) args[0];  
            double realPrice = price * 0.75;  
  
            System.out.println("代理商收到： " + price + "元，实际使用" + realPrice + "去电脑工厂买
```



```

    电脑");
        computerFactory.sellComputer(realPrice);
        System.out.println("代理商赚到: " + (price - realPrice) + "元钱");
        return null;
    } else { // 不需要增强的方法直接调用目标对象的方法
        return method.invoke(computerFactory, args);
    }
}
}
}

```

代理商收到：5000.0元，实际使用3750.0去电脑工厂买电脑
电脑工厂卖出一台电脑，价格： 3750.0
代理商赚到：1250.0元钱

代理对象对需要增强的功能进行了增强

电脑工厂修好一台电脑，价格： 800.0

不需要增强的功能，直接调用目标对象的相应功能

2.5.4.5 动态代理模式小结

1. 什么是动态代理 在程序运行的过程中，动态创建出代理对象。
2. 动态代理使用到的API Proxy类的新ProxyInstance(ClassLoader loader, Class[] interfaces, InvocationHandler h)方法 InvocationHandler接口的invoke(Object proxy, Method method, Object[] args)方法
3. Proxy.newProxyInstance()方法的本质 创建了代理对象，代理对象实现了传入的接口。
4. InvocationHandler接口的作用 调用代理对象的方法就会执行到InvocationHandler接口的invoke方法。
5. 动态代理使用步骤
 1. 直接创建真实对象
 2. 通过Proxy类，创建代理对象
 3. 调用代理方法
 4. 在InvocationHandler的invoke进行处理
6. 代理模式的好处 在不改变目标类的代码情况下，代理对象可以对目标对象的功能进行增强。

2.5.5 动态代理解决close问题

在创建Connection连接时使用动态代理产生Connection代理对象，使用代理对象。修改 createConnection 方法，创建Connection代理对象。

1. 自定义连接池类：MyPool2.java

```

public class MyPool2 implements DataSource {
    private int initCount = 5; //初始化的个数
    private int maxCount = 10; //最大的连接个数
    private int curCount = 0; //当前已经创建的连接个数

    // 存储连接的容器
    private LinkedList<Connection> list = new LinkedList<Connection>();

    // 构造方法,初始化连接池的, 连接池一旦创建那么连接池中就应该要有指定个数的连接。
    public MyPool2() {
        for (int i = 0; i < initCount; i++) {

```



```
        Connection connection = createConnection();
        // 把连接存储到容器中
        list.add(connection);
    }
}

// 创建Connection的方法
public Connection createConnection() {
    Connection proxyConnection = null;
    try {
        Connection connection = JDBCUtils.getConnection();
        // 使用动态代理，产生代理对象
        proxyConnection = (Connection) Proxy.newProxyInstance(
            MyPool2.class.getClassLoader(),
            new Class[]{Connection.class},
            new ConnectionHandler(connection));
    } catch (SQLException e) {
        e.printStackTrace();
    }
    curCount++;
    return proxyConnection;
}

// 自定义处理器 ---内部类
class ConnectionHandler implements InvocationHandler {
    private Connection connection;    // 在内部维护一个需要被增强的对象（被代理对象）
    public ConnectionHandler(Connection connection) {
        this.connection = connection;
    }

    @Override
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        // 先得到当前调用的方法的方法名
        String methodName = method.getName();
        if ("close".equalsIgnoreCase(methodName)) {
            // 需要增强的方法，进行增强
            list.addLast((Connection) proxy);
        } else {
            // 其他方法正常调用
            return method.invoke(connection, args);
        }
        return null;
    }
}

// 别人问你的连接池要连接
@Override
public Connection getConnection() {
    // 情况一：先判断连接池是否有连接
    if (list.size() > 0) {
        //如果连接池有连接，那么直接取出连接，返回即可
        Connection connection = list.removeFirst();

        return connection;
    }
}
```



```
    }

    // 情况二：连接池中无连接，然后先判断目前的连接个数是否已经超过了最大的连接个数
    if (curCount < maxCount) {
        // 创建连接
        Connection connection = createConnection();
        return connection;
    } else {
        // 没有连接，并且已经超过了最大的连接个数
        throw new RuntimeException("已经达到了最大的连接个数，请稍后");
    }
}

// 回收Connection
public void close(Connection connection) {
    list.addLast(connection);
}

@Override
public Connection getConnection(String username, String password) throws SQLException {
    return null;
}

@Override
public PrintWriter getLogWriter() throws SQLException {
    return null;
}

@Override
public void setLogWriter(PrintWriter out) throws SQLException {
}

@Override
public void setLoginTimeout(int seconds) throws SQLException {
}

@Override
public int getLoginTimeout() throws SQLException {
    return 0;
}

@Override
public Logger getParentLogger() throws SQLFeatureNotSupportedException {
    return null;
}

@Override
public <T> T unwrap(Class<T> iface) throws SQLException {
    return null;
}
```



```
@Override
public boolean isWrapperFor(Class<?> iface) throws SQLException {
    return false;
}
}
```

2. 测试类: Demo05.java

```
public class Demo05 {
    public static void main(String[] args) throws SQLException {
        //创建自定义的连接池
        MyPool2 myPool = new MyPool2();
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        System.out.println("连接: "+ myPool.getConnection());
        Connection connection = myPool.getConnection() ;    ///代理Connection
        System.out.println("连接: "+ connection );

        /*
            目前的问题:  由于我们编程习惯的问题, 我们都习惯关闭资源的时候直接调用close方法, 不习惯传递
            进去。

            目标:   如果连接需要归还给连接池, 那么直接调用close方法即可。
            对connection的close方法不满意, 需要增强。
        */

        // 把连接还给连接池
        connection.close(); // 代理对象的Connection调用close, 那么还回给连接池的对象是原有connection
        // 还是代理对象?

        System.out.println("连接: "+ myPool.getConnection());
    }
}
```


3. 效果

```
连接1: com.mysql.jdbc.JDBC4Connection@2b662a77
连接2: com.mysql.jdbc.JDBC4Connection@7f0eb4b4
连接3: com.mysql.jdbc.JDBC4Connection@5c33f1a9
连接4: com.mysql.jdbc.JDBC4Connection@1623b78d
连接5: com.mysql.jdbc.JDBC4Connection@c8c12ac
连接6: com.mysql.jdbc.JDBC4Connection@1b7cc17c
连接7: com.mysql.jdbc.JDBC4Connection@35e2d654
连接8: com.mysql.jdbc.JDBC4Connection@3c19aaa5
连接9: com.mysql.jdbc.JDBC4Connection@689604d9
连接10: com.mysql.jdbc.JDBC4Connection@4135c3b
连接11: com.mysql.jdbc.JDBC4Connection@4135c3b
```

连接10又放回到了连接池中

第3章 C3P0连接池

3.1 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: `c3p0-0.9.1.2.jar`  `c3p0-0.9.1.2.jar`

3.2 常用的配置参数解释

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

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。

3.3 C3P0连接池基本使用

3.3.1 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: 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)
4. 配置方式二: 使用命名配置 (named-config)

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>
```

3.3.2 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

3.3.3 使用步骤

1. 导入jar包 `c3p0-0.9.1.2.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

3.3.4 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

3.3.5 案例代码

1. 准备数据

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

2. 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>  
  
        <!-- 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>
```

```
</named-config>
</c3p0-config>
```

3. 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
    }
}
```

3.3.6 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@5aceled4
Process finished with exit code 0
```

1. 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@5aceled4
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

1. 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

3.3.7 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4章 DRUID连接池

4.1 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`  `druid-1.0.9.jar`

4.2 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。

4.3 DRUID连接池基本使用

4.3.1 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
```

4.3.2 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

4.3.3 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
}
```

4.3.4 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 < 11; 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 + ", score = " + score);
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
```

4.3.5 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。

4.4 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 DataSourceUtils.java

```
public class DataSourceUtils {
    // 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 = DataSourceUtils.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
        DataSourceUtils.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。