数据库连接池

概念

> *其实就是一个容器(集合)，存放数据库连接的容器。*

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

好处：

- 节约资源
- 用户访问高效

实现：

标准接口：DataSource有两个包有，但是这个应该是javax.sql包下的Datasource

方法：

- 获取连接：getConnection()
  - 归还连接：Connection.close()。以前代表的是关闭连接，现在是归还链接。就是说，如果连接对象Connection是从连接池中获取的，那么调用Connection.close()方法，则不会再关闭连接了。而是归还连接

一般来说，数据库连接池我们不去实现它，有数据库厂商来实现

- C3P0：数据库连接池技术
- Druid：数据库连接池实现技术，由阿里巴巴提供的

C3P0：数据库连接池技术

- 步骤：

  ### 导入jar包 (两个) c3p0-0.9.5.2.jar，mchange-commons-java-0.2.12.jar，

  - 不要忘记导入数据库驱动jar包

  ### 定义配置文件：

  - 名称： c3p0.properties 或者 c3p0-config.xml（必须叫这两个名字）

    XML

```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/wang</property>
    <property name="user">root</property>
    <property name="password">root</property>

    <!-- 连接池参数 -->
      <!--初始化申请的连接数量-->
    <property name="initialPoolSize">5</property>
      <!--最大的连接数量-->
    <property name="maxPoolSize">10</property>
      <!--超时时间-->
    <property name="checkoutTimeout">3000</property>
  </default-config>

  <named-config name="otherc3p0">
    <!--  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">8</property>
    <property name="checkoutTimeout">1000</property>
  </named-config>
</c3p0-config>
```

- 路径：直接将文件放在src目录下即可。

创建核心对象 数据库连接池对象 ComboPooledDataSource

获取连接： getConnection

- 代码：

```
package datasource.c3p0;


import com.mchange.v2.c3p0.ComboPooledDataSource;

import javax.sql.DataSource;
import java.sql.Connection;
import java.sql.SQLException;

public class C3P0Demo1 {

    public static void main(String[] args) throws SQLException {

//        导入包  c3p0-0.9.5.2.jar, mchange-commons-java-0.2.12.jar, mysql-connector-java-5.1.19-bin.jar, 定义好配置文件
```

```
//            创建数据库连接池对象
        DataSource ds = new ComboPooledDataSource();

//            获取连接对象
        Connection connection = ds.getConnection();
    }
}
```

> *注意，在创建数据库连接池对象的时候，如果不传递参数，读取的XML就会是默认配置，也就是*
>
> *</>里面的值，假如向里面传递参数，比如传递：“otherc3po”，那么就会读取里面的配置*

Druid：数据库连接池实现技术，由阿里巴巴提供的

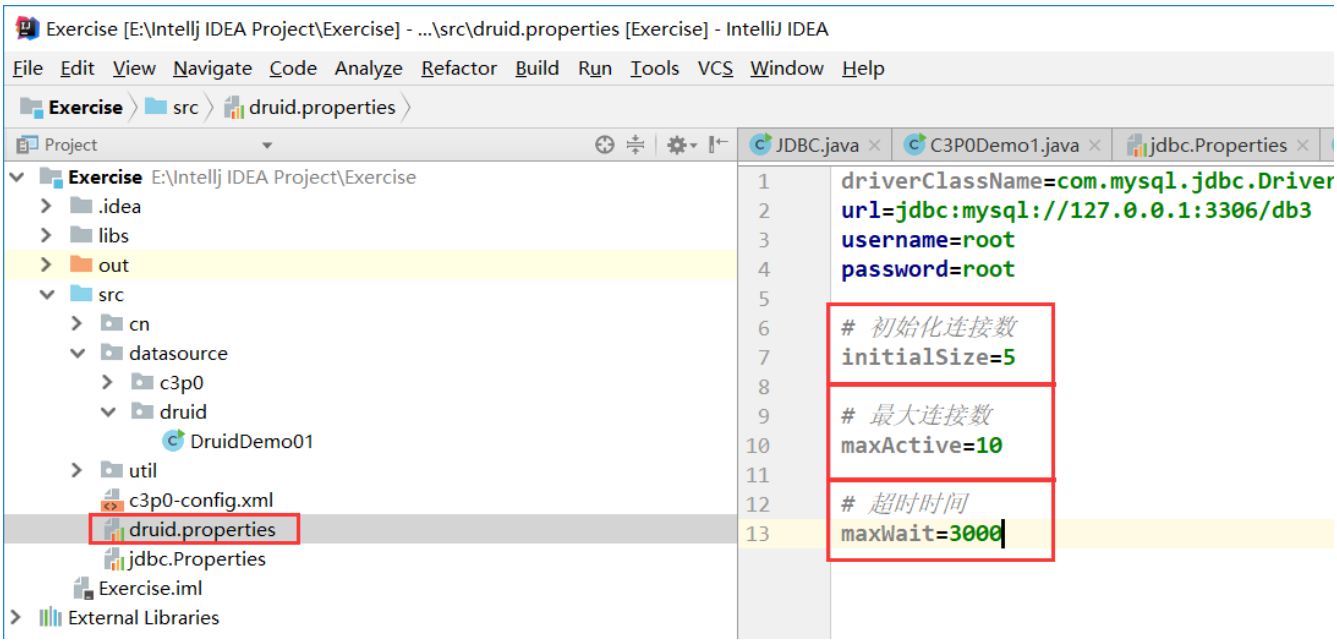
步骤：

导入jar包 druid-1.0.9.jar

定义配置文件：

- 是properties形式的
- 可以叫任意名称，可以放在任意目录下(这个就说明了不会自动加载，需要手动指定名字和路径)

加载配置文件。Properties



获取数据库连接池对象：通过工厂来来获取 DruidDataSourceFactory

获取连接：getConnection

```
```

```java
package datasource.druid;

import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.util.Properties;

public class DruidDemo01 {

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

//        导入jar包 druid-1.0.9.jar, mysql-connector-java-5.1.37-bin.jar

//        定义配置文件,druid.properties

//        加载配置文件

        /*获取Properties对象*/
        Properties properties = new Properties();

        /*获取类加载器*/
        ClassLoader classLoader = DruidDemo01.class.getClassLoader();

        /*获取输入流*/
        InputStream resourceAsStream =
classLoader.getResourceAsStream("druid.properties");

        /*加载druid.properties文件*/
        properties.load(resourceAsStream);

//        获取连接池对象
        DataSource dataSource =
DruidDataSourceFactory.createDataSource(properties);

//        获取连接
        Connection connection = dataSource.getConnection();

    }

}
```

定义工具类

1. 定义一个类 JDBCUtils
2. 提供静态代码块加载配置文件，初始化连接池对象 3. 提供方法 1. 获取连接方法：通过数据库连接池获取连接

2. 释放资源 3. 获取连接池的方法

```
package util;


import com.alibaba.druid.pool.DruidDataSourceFactory;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Properties;



/*Druid工具类*/
public class JDBCUtils {



    private static DataSource dataSource=null;

    /*在静态代码块中加载配置文件*/
    static {

        try {
            /*获取Properties对象*/
            Properties properties = new Properties();

            /*使用class字节码文件获取类加载器*/
            ClassLoader classLoader = JDBCUtils.class.getClassLoader();

            /*使用类加载器获取输入流*/
            InputStream resourceAsStream =
classLoader.getResourceAsStream("druid.properties");

            /*载入配置文件*/
            properties.load(resourceAsStream);

            /*获取Datasource*/
            dataSource = DruidDataSourceFactory.createDataSource(properties);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }

    /*获取连接方法*/
    public static Connection connection(){

        Connection connection=null;
        try {
            connection = dataSource.getConnection();
        } catch (SQLException e) {
            e.printStackTrace();
        }

        return connection;
    }

    /*关闭的方法*/
    public static void close(Statement statement,Connection connection){

        if (statement!=null){

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

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

    /*关闭的方法重载*/
    public static void close(ResultSet resultSet,Statement statement,Connection
connection){
        if (resultSet!=null){
            try {
                resultSet.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }

        if (statement!=null){

            try {
```

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

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

    }

    /*获取连接池的方法
    * 为什么要这个方法呢?
    * 因为在有的小框架里面，只需一个连接池，其他的操作全都帮你做了
    * */
    public static DataSource getDataSource(){
        return dataSource;
    }
}
```

案例：使用JDBCUtils来实现查询操作

```
package datasource.druid;

import com.alibaba.druid.pool.DruidDataSourceFactory;
import util.JDBCUtils;

import javax.sql.DataSource;
import java.io.IOException;
import java.io.InputStream;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Properties;

public class DruidDemo01 {

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

        /*获取连接*/
        Connection connection = JDBCUtils.connection();


        /*定义sql*/
```

```
        String sql = "SELECT * from student";

        /*获取preparedStatement*/
        PreparedStatement preparedStatement = connection.prepareStatement(sql);

        /*执行sql，返回的是影响的行数*/
        ResultSet resultSet = preparedStatement.executeQuery();

        /*处理结果*/
        while (resultSet.next()){
            System.out.println(resultSet.getString("name"));
        }

        /*关闭流*/
        JDBCUtils.close(resultSet,preparedStatement,connection);
    }

}
```

Spring JDBC

- Spring框架对JDBC的简单封装。提供了一个JDBCTemplate对象简化JDBC的开发

步骤：

导入jar包

> *commons-logging-1.2.jar，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*

创建JdbcTemplate对象。依赖于数据源DataSource

- JdbcTemplate template = new JdbcTemplate(new Datasource());

调用JdbcTemplate的方法来完成CRUD的操作

- update():执行DML语句。增、删、改语句

```
package jdbcTemplate;

import org.springframework.jdbc.core.JdbcTemplate;
import util.JDBCUtils;

public class jdbcTemplateDemo1 {

    public static void main(String[] args) {

//        导入jar包（五个）
```

```
//          创建JDBCEmplate对象

            /*注意了，这个JDBCUtils是刚才讲解druid写的工具类
          * 其中getDatasource的用处就是返回一个DataSource
            * 因为JdbcTemplate需要一个Datasource
            * */
            JdbcTemplate jdbcTemplate = new
    JdbcTemplate(JDBCUtils.getDataSource());

    //          调用方法
            /*sql语句*/
            String sql="update student set age = ? where id=?";

            /*给?赋值，这个方法要比PreparedStatement.setXXX(索引(1开始),值)要好得多
            * jdbcTemplate.update(sql,40,1);
            * 其中sql就是sql语句，然后按照位置从左到右进行赋值，那么age就是40，id就是1
            * 返回值是影响的行数
            * */
            int count = jdbcTemplate.update(sql, 40, 1);

    //          处理结果
            System.out.println("共有"+count+"受到影响");

    //          归还链接与释放资源
            /*不需要了，内部已经做完了*/
        }

    }
```

- queryForMap(sql,sql语句中给?赋值):查询结果将结果集封装为map集合，将列名作为key，将值作为value 将这条记录封装为一个map集合
- 注意：这个方法查询的结果集长度只能是1
- queryForList(sql,sql语句中给?赋值):查询结果将结果集封装为list集合
    - 注意：将每一条记录封装为一个Map集合，再将Map集合装载到List集合中
- query():查询结果，将结果封装为JavaBean对象
    - query的参数：sql,new RowMapper<查询的表的对象>(){实现对象}
    - 一般我们使用BeanPropertyRowMapper实现类。可以完成数据到JavaBean的自动封装
    - new RowMapper<查询的表的对象>(){实现对象}可以使用：new BeanPropertyRowMapper<类型>(类型.class);来替代

```
package cn.domain;

public class User {

    private int id,
                age;

    private String  name,
```

```
                    gender,
                    address,
                    qq,
                    email;

    public User() {
    }

    public User(int id, int age, String name, String gender, String address,
String qq, String email) {
        this.id = id;
        this.age = age;
        this.name = name;
        this.gender = gender;
        this.address = address;
        this.qq = qq;
        this.email = email;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getGender() {
        return gender;
    }

    public void setGender(String gender) {
        this.gender = gender;
    }
```

```
    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getQq() {
        return qq;
    }

    public void setQq(String qq) {
        this.qq = qq;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    @Override
    public String toString() {
        return "User{" +
                "id=" + id +
                ", age=" + age +
                ", name='" + name + '\'' +
                ", gender='" + gender + '\'' +
                ", address='" + address + '\'' +
                ", qq='" + qq + '\'' +
                ", email='" + email + '\'' +
                '}';
    }
}
```

```
package cn.dao.Impl;

import cn.dao.UserDao;
import cn.domain.User;
import cn.util.JDBCUtils;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;

import java.util.List;

public class UserDaoImpl implements UserDao {
```

```
    /*
    * 使用JDBC操作数据库
    * */

    private JdbcTemplate template = new JdbcTemplate(JDBCUtils.getDataSource());

    @Override
    public List<User> findAll() {
        String sql = "select * from user";

        List<User> query = template.query(sql, new BeanPropertyRowMapper<User>
(User.class));

        return query;
    }
}
```

> *所以参数就是sql,new BeanPropertyRowMapper<类型>(类型.class);*

> *但是注意，是用这种方式的时候基本类型是不可以使用的，必须使用包装类，也就是使用表的类不能用基本类型，而是用包装类，比如int改为Integer，double改为Double*

- queryForObject(sql,class字节码)：查询结果，将结果封装为对象
- 一般用于聚合函数的查询

练习：

id	ename	job_id	mgr	joindate	salary	bonus	dept_id
1001	孙悟空	4	1004	2000-12-17	8000.00	(NULL)	20
1002	卢俊义	3	1006	2001-02-20	16000.00	3000.00	30
1003	林冲	3	1006	2001-02-22	12500.00	5000.00	30
1004	唐僧	2	1009	2001-04-02	29750.00	(NULL)	20
1005	李逵	4	1006	2001-09-28	12500.00	14000.00	30
1006	宋江	2	1009	2001-05-01	28500.00	(NULL)	30
1007	刘备	2	1009	2001-09-01	24500.00	(NULL)	10
1008	猪八戒	4	1004	2007-04-19	30000.00	(NULL)	20
1009	罗贯中	1	(NULL)	2001-11-17	50000.00	(NULL)	10
1010	吴用	3	1006	2001-09-08	15000.00	0.00	30
1011	沙僧	4	1004	2007-05-23	11000.00	(NULL)	20
1012	李逵	4	1006	2001-12-03	9500.00	(NULL)	30
1013	小白龙	4	1004	2001-12-03	30000.00	(NULL)	20
1014	关羽	4	1007	2002-01-23	13000.00	(NULL)	10

- 需求：

1. 修改1001号数据的 salary 为 10000
2. 添加一条记录
3. 删除刚才添加的记录

4. 查询id为1001的记录，将其封装为Map集合
5. 查询所有记录，将其封装为List
6. 查询所有记录，将其封装为Emp对象的List集合
7. 查询总记录数

> *emp表所对应的类*

```
package jdbcTemplate;


import java.sql.Date;

/*要对emp表进行操作，那么就需要emp对象，就需要emp类*/
public class EMP {

    /*注意要使用包装类*/
    private Integer id, job_id, mgr, dept_id;

    private String  ename;

    /*注意要使用包装类*/
    private Date joindate;

    /*注意要使用包装类*/
    private Double salary, bonus;

    public EMP() {
    }

    public EMP(Integer id, Integer job_id, Integer mgr, Integer dept_id, String
ename, Date joindate, Double salary, Double bonus) {
        this.id = id;
        this.job_id = job_id;
        this.mgr = mgr;
        this.dept_id = dept_id;
        this.ename = ename;
        this.joindate = joindate;
        this.salary = salary;
        this.bonus = bonus;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public Integer getJob_id() {
```

```
    return job_id;
}

public void setJob_id(Integer job_id) {
    this.job_id = job_id;
}

public Integer getMgr() {
    return mgr;
}

public void setMgr(Integer mgr) {
    this.mgr = mgr;
}

public Integer getDept_id() {
    return dept_id;
}

public void setDept_id(Integer dept_id) {
    this.dept_id = dept_id;
}

public String getEname() {
    return ename;
}

public void setEname(String ename) {
    this.ename = ename;
}

public Date getJoindate() {
    return joindate;
}

public void setJoindate(Date joindate) {
    this.joindate = joindate;
}

public Double getSalary() {
    return salary;
}

public void setSalary(Double salary) {
    this.salary = salary;
}

public Double getBonus() {
    return bonus;
}
```

```
    public void setBonus(Double bonus) {
        this.bonus = bonus;
    }

    @Override
    public String toString() {
        return "EMP{" +
                "id=" + id +
                ", job_id=" + job_id +
                ", mgr=" + mgr +
                ", dept_id=" + dept_id +
                ", ename='" + ename + '\'' +
                ", joindate=" + joindate +
                ", salary=" + salary +
                ", bonus=" + bonus +
                '}';
    }
}
```

> *使用测试类来实现*

```
package jdbcTemplate;

import org.junit.Test;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import util.JDBCUtils;

import java.sql.Connection;
import java.sql.Date;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.Map;

public class jdbcTemplateDemo1 {

    @Test
    public void test1(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*修改1001号的数据为10000*/
        int count_update = jdbcTemplate.update("update emp set salary= ? where
id = ?", 10000, 1001);
        System.out.println(count_update);
    }
```

```
    @Test
    public void test2(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*添加一条记录*/
        int count_insert = jdbcTemplate.update("INSERT INTO emp VALUES (NULL,'王小二',2,1009,NULL ,2000,NULL ,10)");
        System.out.println(count_insert);
    }

    @Test
    public void test3(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*删除刚才添加的记录*/
        int count_delete = jdbcTemplate.update("DELETE FROM emp where ename=?", "王小二");
    }

    @Test
    public void test4(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*查询id为1的记录，将其封装为map集合*/
        Map<String, Object> map = jdbcTemplate.queryForMap("SELECT * FROM emp WHERE id=?", 1001);

        System.out.println(map);
    }

    @Test
    public void test5(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /* 查询所有记录，将其封装为List*/
        List<Map<String, Object>> list = jdbcTemplate.queryForList("SELECT * FROM emp");
        for (Map<String, Object> stringObjectMap : list) {
            System.out.println(stringObjectMap);
        }

    }
```

```
    @Test
    public void test6(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*查询所有记录，将其封装为EMP对象的List集合
        * 其中new RowMapper<EMP>()可以自己实现也可以交给他们来实现，先来一个自己实现的
        * */
        List<EMP> empQuery = jdbcTemplate.query("SELECT * FROM emp", new
RowMapper<EMP>() {
            @Override
            public EMP mapRow(ResultSet resultSet, int i) throws SQLException {

                EMP emp = new EMP();

                int id = resultSet.getInt("id"),
                    job_id = resultSet.getInt("job_id"),
                    mgr = resultSet.getInt("mgr"),
                    dept_id = resultSet.getInt("dept_id");

                String  ename = resultSet.getString("ename");

                Date joindate = resultSet.getDate("joindate");

                double  salary = resultSet.getDouble("salary"),
                        bonus = resultSet.getDouble("bonus");

                emp.setId(id);
                emp.setBonus(bonus);
                emp.setJoindate(joindate);
                emp.setDept_id(dept_id);
                emp.setEname(ename);
                emp.setJob_id(job_id);
                emp.setMgr(mgr);
                emp.setSalary(salary);

                return emp;
            }
        });

        for (EMP emp : empQuery) {
            System.out.println(emp);
        }
    }


    @Test
    public void test6_2(){
        Connection connection = JDBCUtils.connection();
```

```
        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*查询所有记录，将其封装为EMP对象的List集合
         * 其中new RowMapper<EMP>()可以自己实现也可以交给他们来实现，这个是交给它自己自动实现的，和自己实现的没有任何区别
         * */
        List<EMP> list = jdbcTemplate.query("SELECT * FROM emp", new BeanPropertyRowMapper<EMP>(EMP.class));

        for (EMP emp : list) {
            System.out.println(emp);
        }

    }



    @Test
    public void test7(){
        Connection connection = JDBCUtils.connection();

        JdbcTemplate jdbcTemplate = new JdbcTemplate(JDBCUtils.getDataSource());

        /*查询总记录数
        * 因为这个返回的是Long类型的，所以这个就是用Long.class
        * */
        Long total = jdbcTemplate.queryForObject("SELECT COUNT(id) FROM emp", Long.class);

        System.out.println(total);
    }

}
```