

数据库连接池

1. 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。

2. 好处：

1. 节约资源
2. 用户访问高效

3. 实现：

1. 标准接口：DataSource javax.sql包下的

1. 方法：

- * 获取连接：getConnection()

- * 归还连接：Connection.close()。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

2. 一般我们不去实现它，有数据库厂商来实现

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

4. C3P0：数据库连接池技术

- * 步骤：

1. 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

2. 定义配置文件：

- * 名称： c3p0.properties 或者 c3p0-config.xml
- * 路径：直接将文件放在src目录下即可。

3. 创建核心对象 数据库连接池对象 ComboPooledDataSource

4. 获取连接： getConnection

- * 代码：

```
//1.创建数据库连接池对象
```

```
DataSource ds = new ComboPooledDataSource();
```

```
//2. 获取连接对象
```

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

5. Druid：数据库连接池实现技术，由阿里巴巴提供的

1. 步骤：

1. 导入jar包 druid-1.0.9.jar

2. 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

3. 加载配置文件。Properties

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

5. 获取连接： getConnection

- * 代码：

```
//3.加载配置文件
```

```

        Properties pro = new Properties();
        InputStream is =
DruidDemo.class.getClassLoader().getResourceAsStream("druid.properties");
        pro.load(is);
        //4. 获取连接池对象
        DataSource ds = DruidDataSourceFactory.createDataSource(pro);
        //5. 获取连接
        Connection conn = ds.getConnection();

```

2. 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

* 代码：

```

public class JDBCUtils {

    //1.定义成员变量 DataSource
    private static DataSource ds ;

    static{
        try {
            //1.加载配置文件
            Properties pro = new Properties();

pro.load(JDBCUtils.class.getClassLoader().getResourceAsStream("druid.properties"));
            //2.获取DataSource
            ds = DruidDataSourceFactory.createDataSource(pro);
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    /**
     * 获取连接
     */
    public static Connection getConnection() throws SQLException {
        return ds.getConnection();
    }

    /**
     * 释放资源
     */
    public static void close(Statement stmt,Connection conn){
        /* if(stmt != null){
            try {
                stmt.close();
            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
    }

```

```

    }
}

if(conn != null){
    try {
        conn.close();//归还连接
    } catch (SQLException e) {
        e.printStackTrace();
    }
}*/

close(null,stmt,conn);
}

```

```

public static void close(ResultSet rs , Statement stmt, Connection conn){

```

```

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

```

```

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

    if(conn != null){
        try {
            conn.close();//归还连接
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

/**
 * 获取连接池方法
 */

public static DataSource getDataSource(){
    return ds;
}

}

```

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:

1. 导入jar包

2. 创建JdbcTemplate对象。依赖于数据源DataSource

```
* JdbcTemplate template = new JdbcTemplate(ds);
```

3. 调用JdbcTemplate的方法来完成CRUD的操作

* update():执行DML语句。增、删、改语句

* queryForMap():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():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: RowMapper

* 一般我们使用BeanPropertyRowMapper实现类。可以完成数据到JavaBean的自动封装

* new BeanPropertyRowMapper<类型>(类型.class)

* queryForObject: 查询结果, 将结果封装为对象

* 一般用于聚合函数的查询

4. 练习:

* 需求:

1. 修改1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1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

* 代码:

```
import cn.itcast.domain.Emp;
import cn.itcast.utils.JDBCUtils;
import org.junit.Test;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
```

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

```
public class JdbcTemplateDemo2 {
```

```
    //JUnit单元测试, 可以让方法独立执行
```

```

//1. 获取JdbcTemplate对象
private JdbcTemplate template = new JdbcTemplate(JDBCUtils.getDataSource());
/**
 * 1. 修改1号数据的 salary 为 10000
 */
@Test
public void test1(){

    //2. 定义sql
    String sql = "update emp set salary = 10000 where id = 1001";
    //3. 执行sql
    int count = template.update(sql);
    System.out.println(count);
}

/**
 * 2. 添加一条记录
 */
@Test
public void test2(){
    String sql = "insert into emp(id,ename,dept_id) values(?,?,?)";
    int count = template.update(sql, 1015, "郭靖", 10);
    System.out.println(count);
}

/**
 * 3. 删除刚才添加的记录
 */
@Test
public void test3(){
    String sql = "delete from emp where id = ?";
    int count = template.update(sql, 1015);
    System.out.println(count);
}

/**
 * 4. 查询id为1001的记录, 将其封装为Map集合
 * 注意: 这个方法查询的结果集长度只能是1
 */
@Test
public void test4(){
    String sql = "select * from emp where id = ? or id = ?";
    Map<String, Object> map = template.queryForMap(sql, 1001, 1002);
    System.out.println(map);
    //{id=1001, ename=孙悟空, job_id=4, mgr=1004, joindate=2000-12-17,
salary=10000.00, bonus=null, dept_id=20}

}

/**
 * 5. 查询所有记录, 将其封装为List
 */

```

```

@Test
public void test5(){
    String sql = "select * from emp";
    List<Map<String, Object>> list = template.queryForList(sql);

    for (Map<String, Object> stringObjectMap : list) {
        System.out.println(stringObjectMap);
    }
}

/**
 * 6. 查询所有记录, 将其封装为Emp对象的List集合
 */

@Test
public void test6(){
    String sql = "select * from emp";
    List<Emp> list = template.query(sql, new RowMapper<Emp>() {

        @Override
        public Emp mapRow(ResultSet rs, int i) throws SQLException {
            Emp emp = new Emp();
            int id = rs.getInt("id");
            String ename = rs.getString("ename");
            int job_id = rs.getInt("job_id");
            int mgr = rs.getInt("mgr");
            Date joindate = rs.getDate("joindate");
            double salary = rs.getDouble("salary");
            double bonus = rs.getDouble("bonus");
            int dept_id = rs.getInt("dept_id");

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

            return emp;
        }
    });
}

```

```

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

/**
 * 6. 查询所有记录, 将其封装为Emp对象的List集合
 */

```

```
        @Test
        public void test6_2(){
            String sql = "select * from emp";
            List<Emp> list = template.query(sql, new BeanPropertyRowMapper<Emp>
(Emp.class));

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

        /**
         * 7. 查询总记录数
         */

        @Test
        public void test7(){
            String sql = "select count(id) from emp";
            Long total = template.queryForObject(sql, Long.class);
            System.out.println(total);
        }
    }
}
```