**Jdbc连接池总结**

# 前言

    JDBC作为J2EE的规范之一，它定义了连接**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**的接口，封装了连接数据库的细节问题，给开发人员极大的方便。开发人员只需要实现这些接口，就可以连接不同的数据库，不需要针对不同数据库写不同的连接代码。

# JDBC连接数据库

    JDBC连接数据库，简单来说有下面四个步骤：

        1、加载数据库驱动

        2、建立连接

        3、访问数据库

        4、关闭数据库连接

    当然，JDBC连接数据库没有这么简单，它还包括Statement对象的创建，如果需要，还会创建ResultSet对象，关闭连接的时候，也要相应的先关闭其他对象，如《[看JDBC轻松连接数据库，你值得拥有](http://blog.csdn.net/fengyao1995/article/details/50757555" \t "_blank)》中，有详细介绍。如果采用单个连接的话，Web的每一次请求都会创建数据库连接，这样既浪费资源又浪费时间，所以就采用了连接池的技术。将数据库连接放入连接池中，访问数据库时去连接池中取连接，用完之后就将连接放回连接池。

# 连接池

    数据库连接池又称Resource Pool，基本思想是为数据库连接建立一个缓冲池，当创建连接池时，预先在连接池中配置一些数量的连接，当需要创建数据库连接时，就在连接池中取出连接，用完之后再放进去，否则当使用连接数超过最大连接数，接着再请求连接时，就会出现Timeout异常。当需要创建的连接数超过连接池中预先放入的连接，再创建新的连接。当然，连接池可以设置它的最大连接数，以及初始化时的连接数。

    JDBC并未提供连接池的方法，我们可以采用Tomcat，对连接池的一些参数进行设置。我们可以通过进入Tomcat的管理界面进行设置，或者直接设置Tomcat的server.xml文件，因为管理界面设置完成后，同样也会写入server.xml文件中。如果采用xml文件直接配置，就需要把下面的代码放入server.xml文件中，然后修改相应的参数就可以了。设置连接池的时候，有几个固定的参数：

1. **<Resource**
2. name="jdbc/drp"
3. type="javax.sql.DataSource"
4. maxActive="4"
5. maxIdle="2"
6. username="drp1"
7. maxWait="5000"
8. driverClassName="oracle.jdbc.driver.OracleDriver"
9. password="123456"
10. url="jdbc:oracle:thin:@localhost:1521:orcl"**/>**

    driverClassName：表示数据库驱动，上面代码中是**[Oracle](http://lib.csdn.net/base/oracle" \o "Oracle知识库" \t "_blank)**驱动，如果是**[MySQL](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**数据库，只需改成Mysql的驱动类就好了

    url：数据库路径，如果是连接服务器的数据库，只需要将localhost改成服务器IP地址就OK了

    username：username是数据库的用户名

    password：是该用户的密码

    maxActive：最大的可用的连接数量

    maxIdle：初始化时，连接池中的空闲连接数

    maxWait：最长等待时间，如果超过这个时间没有获得连接（单位为ms），则提示Timeout

    name：是随便起的名字

    type：指的是实现的javax.sql.DataSource的接口，是JDBC提供的

    配置完成之后，该连接池就对Tomcat服务器下的所有程序起作用。如果想让该连接池设置只对某一个应用程序起作用，只需要将上述代码拷贝到tomcat/conf路径下的context.xml文件中，然后将该文件剪切到应用程序的WebRoot/META-INF路径下即可。

    当需要创建连接池时，直接去连接池中取就好了：

1. Connection conn = **null**;
2. **try** {
3. Context ctx = **new** InitialContext();
4. //通过JNDI查找DataSource
5. DataSource ds = (DataSource)ctx.lookup("java:comp/env/jdbc/drp");
6. conn = ds.getConnection();
7. }**catch**(Exception e) {
8. e.printStackTrace();
9. **throw** **new** ApplicationException("系统错误，请联系系统管理员");
10. }
11. **return** conn;

# 线程并发问题

    数据库连接池并不是线程安全的，所以我们可以在创建Connection时可以采用synchronized关键字来保证线程同步，也可以采用ThreadLocal进行管理。当一个方法需要创建数据库连接时，可以先判断它所在的线程是否已经绑定了一个Connection，如果已经绑定了，那就直接用，如果没有，就创建一个Connection对象，放到ThreadLocal中，以备后面的方法使用。

1. **private** **static** ThreadLocal<Connection> connectionHolder = **new** ThreadLocal<Connection>();
2. /\*\*
3. \* 得到Connection
4. \* @return
5. \*/
6. **public** **static** Connection getConnection() {
7. Connection conn = connectionHolder.get();
8. //如果在当前线程中没有绑定相应的Connection
9. **if** (conn == **null**) {
10. //创建一个新连接
11. conn = DbUtil.getConnection();
12. //将连接放入ThreadLocal
13. connectionHolder.set(conn);
14. }
15. **return** conn;
16. }

当然，关闭连接时，也需要把Connection从ThreadLocal中移除出去：

1. **public** **static** **void** closeConnection() {
2. Connection conn = connectionHolder.get();
3. **if** (conn != **null**) {
4. **try** {
5. conn.close();
6. //从ThreadLocal中清除Connection
7. connectionHolder.remove();
8. } **catch** (SQLException e) {
9. e.printStackTrace();
10. }
11. }
12. }

# 总结

    JDBC作为一个规范，它将连接数据库的细节封装了起来，提供给我们各种接口，只要去实现这些接口，就可以实现数据库的连接。JDBC连接数据库有固定的步骤，而每创建一个连接都需要执行这个过程，对于访问人数较多的网站来说，如果不对数据库连接进行限制的话，系统会一直创建连接，这样就会造成数据库的内存泄露，对数据安全造成问题，所以就采用了连接池技术做改善，在连接池中限制最大连接数，只允许程序创建一定量的连接，用完之后就一定要放回连接池。即使这样，也会出现多线程并发问题，对于线程同步，可以采用synchronized关键字，也可以使用ThreadLocal来解决。