c3p0的配置方式分为三种，分别是   
1.setters一个个地设置各个配置项   
2.类路径下提供一个c3p0.properties文件   
3.类路径下提供一个c3p0-config.xml文件   
  
1.setters一个个地设置各个配置项   
这种方式最繁琐，形式一般是这样：   
  
Properties props = new Properties();   
InputStream in = ConnectionManager.class.getResourceAsStream("/c3p0.properties");   
props.load(in);   
in.close();   
  
ComboPooledDataSource cpds = new ComboPooledDataSource();   
cpds.setDriverClass(props.getProperty("driverClass"));   
cpds.setJdbcUrl(props.getProperty("jdbcUrl"));   
cpds.setUser(props.getProperty("user"));   
cpds.setPassword(props.getProperty("password"));   
因为繁琐，所以很不适合采用，于是文档提供了另外另种方式。   
  
2. 类路径下提供一个c3p0.properties文件   
文件的命名必须是c3p0.properties，里面配置项的格式为：   
  
c3p0.driverClass=com.mysql.jdbc.Driver   
c3p0.jdbcUrl=jdbc:mysql://localhost:3306/jdbc   
c3p0.user=root   
c3p0.password=java   
上面只提供了最基本的配置项，其他配置项参照 文档配置，记得是c3p0.后面加属性名就是了，最后初始化数据源的方式就是这样简单：   
private static ComboPooledDataSource ds = new ComboPooledDataSource();   
public static Connection getConnection() {   
    try {   
        return ds.getConnection();   
    } catch (SQLException e) {   
        throw new RuntimeException(e);   
    }   
}   
3.类路径下提供一个c3p0-config.xml文件    
这种方式使用方式与第二种差不多，但是有更多的优点   
(1).更直观明显，很类似hibernate和spring的配置   
(2).可以为多个数据源服务，提供default-config和named-config两种配置方式   
下面是一个配置模板：   
<c3p0-config>   
  <default-config>    
    <property name="user">root</property>   
    <property name="password">java</property>   
    <property name="driverClass">com.mysql.jdbc.Driver</property>   
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/jdbc</property>   
    <property name="initialPoolSize">10</property>   
    <property name="maxIdleTime">30</property>   
    <property name="maxPoolSize">100</property>   
    <property name="minPoolSize">10</property>   
    </default-config>   
  
    <named-config name="myApp">   
    <property name="user">root</property>   
    <property name="password">java</property>   
    <property name="driverClass">com.mysql.jdbc.Driver</property>   
    <property name="jdbcUrl">jdbc:mysql://localhost:3306/jdbc</property>   
       
    <property name="initialPoolSize">10</property>   
    <property name="maxIdleTime">30</property>   
    <property name="maxPoolSize">100</property>   
    <property name="minPoolSize">10</property>   
  </named-config>   
</c3p0-config>   
如果要使用default-config则初始化数据源的方式与第二种一样，如果要使用named-config里面配置初始化数据源，则只要使用一个带参数的ComboPooledDataSource构造器就可以了   
private static ComboPooledDataSource ds = new ComboPooledDataSource("myApp");   
下面整理一下从文档和网上学习到的c3p0配置的理解 (user,password,driverClass,jdbcUrl没有说的必要)   
1.基本配置项   
  
acquireIncrement   
default : 3   
连接池在无空闲连接可用时一次性创建的新数据库连接数   
  
initialPoolSize   
default : 3   
连接池初始化时创建的连接数   
  
maxPoolSize   
default : 15   
连接池中拥有的最大连接数，如果获得新连接时会使连接总数超过这个值则不会再获取新连接，而是等待   
其他连接释放，所以这个值有可能会设计地很大   
  
maxIdleTime   
default : 0 单位 s   
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   
  
minPoolSize   
default : 3   
连接池保持的最小连接数，后面的maxIdleTimeExcessConnections跟这个配合使用来减轻连接池的负载   
2.管理连接池的大小和连接的生存时间   
  
maxConnectionAge   
default : 0 单位 s   
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掉。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。   
  
maxIdleTimeExcessConnections   
default : 0 单位 s   
这个配置主要是为了减轻连接池的负载，比如连接池中连接数因为某次数据访问高峰导致创建了很多数据连接   
但是后面的时间段需要的数据库连接数很少，则此时连接池完全没有必要维护那么多的连接，所以有必要将   
断开丢弃掉一些连接来减轻负载，必须小于maxIdleTime。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。   
maxIdleTime也可以归属到这一类，前面已经写出来了。   
  
3.配置连接测试：因为连接池中的数据库连接很有可能是维持数小时的连接，很有可能因为数据库服务器的问题，网络问题等导致实际连接已经无效，但是连接池里面的连接还是有效的，如果此时获得连接肯定会发生异常，所以有必要通过测试连接来确认连接的有效性。   
下面的前三项用来配置如何对连接进行测试，后三项配置对连接进行测试的时机。   
  
automaticTestTable   
default : null   
用来配置测试连接的一种方式。配置一个表名，连接池根据这个表名创建一个空表，   
并且用自己的测试sql语句在这个空表上测试数据库连接   
这个表只能由c3p0来使用，用户不能操作，同时用户配置的preferredTestQuery 将会被忽略。   
  
preferredTestQuery   
default : null   
用来配置测试连接的另一种方式。与上面的automaticTestTable二者只能选一。   
如果要用它测试连接，千万不要设为null，否则测试过程会很耗时，同时要保证sql语句中的表在数据库中一定存在。   
  
connectionTesterClassName   
default :  com.mchange.v2.c3p0.impl.DefaultConnectionTester   
连接池用来支持automaticTestTable和preferredTestQuery测试的类，必须是全类名，就像默认的那样，   
可以通过实现UnifiedConnectionTester接口或者继承AbstractConnectionTester来定制自己的测试方法   
idleConnectionTestPeriod   
default : 0   
用来配置测试空闲连接的间隔时间。测试方式还是上面的两种之一，可以用来解决MySQL8小时断开连接的问题。因为它   
保证连接池会每隔一定时间对空闲连接进行一次测试，从而保证有效的空闲连接能每隔一定时间访问一次数据库，将于MySQL   
8小时无会话的状态打破。为0则不测试。   
  
testConnectionOnCheckin   
default : false   
如果为true，则在close的时候测试连接的有效性。为了提高测试性能，可以与idleConnectionTestPeriod搭配使用，   
配置preferredTestQuery或automaticTestTable也可以加快测试速度。   
testConnectionOnCheckout   
default : false   
性能消耗大。如果为true，在每次getConnection的时候都会测试，为了提高性能，   
可以与idleConnectionTestPeriod搭配使用，   
配置preferredTestQuery或automaticTestTable也可以加快测试速度。   
4.配置PreparedStatement缓存   
maxStatements   
default : 0   
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   
default : 0   
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。   
5.重连相关配置   
acquireRetryAttempts   
default : 30   
连接池在获得新连接失败时重试的次数，如果小于等于0则无限重试直至连接获得成功   
acquireRetryDelay   
default : 1000 单位ms   
连接池在获得新连接时的间隔时间   
breakAfterAcquireFailure   
default : false   
如果为true，则当连接获取失败时自动关闭数据源，除非重新启动应用程序。所以一般不用。   
个人觉得上述三个没有更改的必要，但可以将acquireRetryDelay配置地更短一些   
6.定制管理Connection的生命周期   
connectionCustomizerClassName   
default : null   
用来定制Connection的管理，比如在Connection acquire 的时候设定Connection的隔离级别，或者在   
Connection丢弃的时候进行资源关闭，就可以通过继承一个AbstractConnectionCustomizer来实现相关   
方法，配置的时候使用全类名。有点类似监听器的作用。   
例如：   
import java.sql.Connection;   
import com.mchange.v2.c3p0.AbstractConnectionCustomizer;   
public class ConnectionCustomizer extends AbstractConnectionCustomizer{   
    @Override   
    public void onAcquire(Connection c, String parentDataSourceIdentityToken)   
            throws Exception {   
        System.out.println("acquire : " + c);   
    }   
    @Override   
    public void onCheckIn(Connection c, String parentDataSourceIdentityToken)   
            throws Exception {   
        System.out.println("checkin : " + c);   
    }   
    @Override   
    public void onCheckOut(Connection c, String parentDataSourceIdentityToken)   
            throws Exception {   
       System.out.println("checkout : " + c);   
    }   
    @Override   
    public void onDestroy(Connection c, String parentDataSourceIdentityToken)   
            throws Exception {   
        System.out.println("destroy : " + c);   
    }   
}   
<property name="connectionCustomizerClassName">liuyun.zhuge.db.ConnectionCustomizer</property>   
7.配置未提交的事务处理   
autoCommitOnClose   
default : false   
连接池在回收数据库连接时是否自动提交事务   
如果为false，则会回滚未提交的事务   
如果为true，则会自动提交事务   
forceIgnoreUnresolvedTransactions   
default : false   
这个配置强烈不建议为true。   
一般来说事务当然由自己关闭了，为什么要让连接池来处理这种不细心问题呢？   
8.配置debug和回收Connection   
unreturnedConnectionTimeout   
default : 0 单位 s   
为0的时候要求所有的Connection在应用程序中必须关闭。如果不为0，则强制在设定的时间到达后回收   
Connection，所以必须小心设置，保证在回收之前所有数据库操作都能够完成。这种限制减少Connection未关闭   
情况的不是很适用。为0不对connection进行回收，即使它并没有关闭。   
debugUnreturnedConnectionStackTraces   
default : false   
如果为true并且unreturnedConnectionTimeout设为大于0的值，当所有被getConnection出去的连接   
unreturnedConnectionTimeout时间到的时候，就会打印出堆栈信息。只能在debug模式下适用，因为   
打印堆栈信息会减慢getConnection的速度   
同第七项一样的，连接用完当然得close了，不要通过unreturnedConnectionTimeout让连接池来回收未关闭的连接。   
9.其他配置项：因为有些配置项几乎没有自己配置的必要，使用默认值就好，所以没有再写出来   
checkoutTimeout   
default : 0   
配置当连接池所有连接用完时应用程序getConnection的等待时间。为0则无限等待直至有其他连接释放   
或者创建新的连接，不为0则当时间到的时候如果仍没有获得连接，则会抛出SQLException