首先创建DataSource，这个时候会读取配置，但不会创建连接对象

然后要调用getConnection，这里会进行数据库连接，似乎要获取一个叫token的东西，很明显这个token很像是c3p0自己生成的跟数据库没有关系，在getPoolManager里面也没有开始数据库连接

看源代码真的是一件很痛苦的事情

**1.看源代码一定要是带着目的去看，比如是要知道某个组件是如何实现的额，而不是从头看到尾。**

**2.关键是找到核心重要的方法或者代码块**

**3.不要太自信，觉得仅凭借自己的力量就可以去去读源代码，还是先看看网友们怎么看，看不懂就去查资料**

**4.人不会相信经验，他总是要去自己尝试一下**

**5.框架一定是基于最原始的东西，一定是最原始的东西有痛点**

**Q：数据库连接池保存的是连接还是数据连接对象，保存数据库连接对象是长连接么？**

**A：首先长连接的保持需要心跳的为此，也就是说没有维持机制的连接一般都为短连接。长连接一般都需要心跳维持机制，否则在第一次发送数据后，会被断开。**

**Q:数据库连接connect，statement，resultset之间的关系**

**A：一个数据库的连接是可以有多个statement的，如果按照传统的编程方式很容易出现一个连接只进行一个statement，这也是为什么要有这个框架**

**对于一个框架要从这借个方面：**

1. **为什么要有这个框架（比如数据连接池在于解决一个statement一个连接的情况）**
2. **框架主要思路**
3. **框架的整体架构（这个地方估计也是最难的了，也是突出专业性的部分，框架也分大小，大型框架一般在官网就可以看到整体架构，而小框架就比较麻烦了，有一个方法就是看包名）**



**关键问题在于：一个链接是可以有多少**

**target 1：**

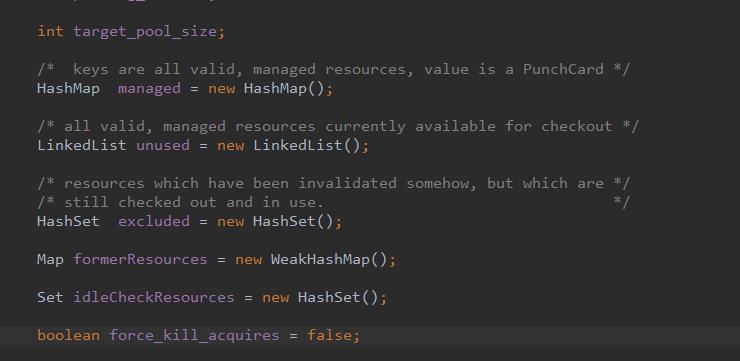
1. **想要知道这个池是用什么数据结构实现的，数组or队列**

**猜测应该使用的是**

**hashset:已经失效了，但还是还在被拿出来用**

**hashmap:key是所有有效的资源，值是PunchCard**

**LinkedList:当前可用的所有资源，可以用来被检出**



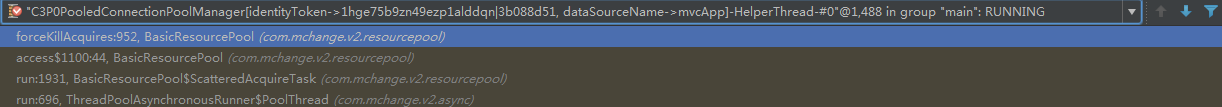
**target 2：**

1. **当进行一次查询开始到查询结束，如何取对象，如何使用对象去进行查询**
2. **这个地方进行第一次连接**
3. public Connection getConnection() throws SQLException  
   {  
    PooledConnection pc = getPoolManager().getPool().checkoutPooledConnection();  
    return pc.getConnection();  
   }
4. public PooledConnection checkoutPooledConnection() throws SQLException  
    {   
    //System.err.println(this + " -- CHECKOUT");  
    try   
    {   
   PooledConnection pc = (PooledConnection) this.checkoutAndMarkConnectionInUse(); //这里进行的连接操作  
   pc.addConnectionEventListener( cl );  
   return pc;  
    }  
    catch (TimeoutException e)  
    { throw SqlUtils.*toSQLException*("An attempt by a client to checkout a Connection has timed out.", e); }  
    catch (CannotAcquireResourceException e)  
    { throw SqlUtils.*toSQLException*("Connections could not be acquired from the underlying database!", "08001", e); }  
    catch (Exception e)  
    { throw SqlUtils.*toSQLException*(e); }  
    }

out = rp.checkoutResource( checkoutTimeout );

**这一句是真真的获取连接**

**这是一个单独的线程：**



class ScatteredAcquireTask implements Runnable

**单独开辟的线程去获取连接**

**target 3：**

1. **如何封装原生异常**

**我们分析一下包结构：**

**cfg：**

**codegen**

**debug**

**example**

**filter**

**impl**

**jboss**

**management**

**mbean**

**stmt**

**subst**

**test**

**util**

**others**

**resourcepool**

**第一遍过滤掉明显不重要的包**

**filter：细看发现跟日志有关，可以再次过滤**

**impl：这个比较重要了，似乎大部分都是java.sql的实现**

**以及自己接口的实现，可以把接口整理一下**

**management：一些管理类**

**stmt：视乎跟一些缓存有关**

**subst：不太重要，类型也很简单**

**resourcepool：资源池**

**others：一些datasource**

**再次过滤就是：**

**impl：这个比较重要了，似乎大部分都是java.sql的实现**

**以及自己接口的实现，可以把接口整理一下**

**management：一些管理类**

**stmt：视乎跟一些缓存有关**

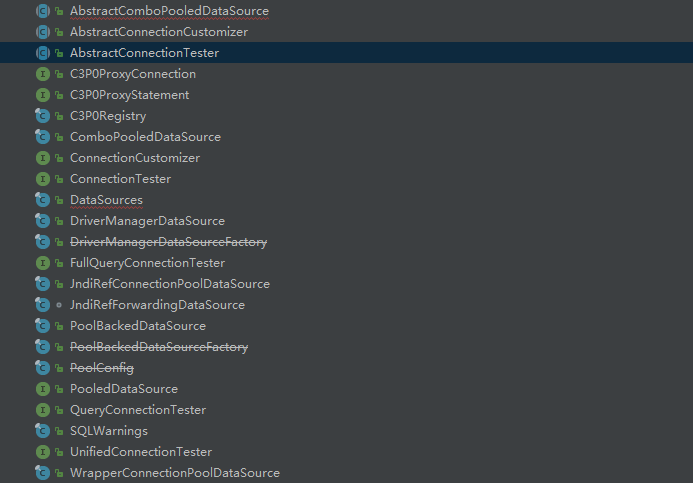
**resourcepool：资源池**

**others：一些datasource**

**主要就是这五个包了**

**先看下others**

**接口是实现类抽象类放在一起：**



public synchronized Connection getConnection()  
 throws SQLException  
 {   
 if ( exposedProxy != null)  
 {  
 //DEBUG  
 //System.err.println("[DOUBLE\_GET\_TESTER] -- double getting a Connection from " + this );  
 //new Exception("[DOUBLE\_GET\_TESTER] -- Double-Get Stack Trace").printStackTrace();  
 //origGet.printStackTrace();  
  
// System.err.println("c3p0 -- Uh oh... getConnection() was called on a PooledConnection when " +  
// "it had already provided a client with a Connection that has not yet been " +  
// "closed. This probably indicates a bug in the connection pool!!!");  
  
 *logger*.warning("c3p0 -- Uh oh... getConnection() was called on a PooledConnection when " +  
 "it had already provided a client with a Connection that has not yet been " +  
 "closed. This probably indicates a bug in the connection pool!!!");  
  
 return exposedProxy;  
 }  
 else  
 { return getCreateNewConnection(); }  
 }

**这个方法里似乎想用单例模式**

map->connetion pool->connetion

pool manager

datasource

datasource

**里面采用的是反射的方式，这个proxy从代码上只有一个**