[HandlerSocket](https://github.com/ahiguti/HandlerSocket-Plugin-for-MySQL)是日本人 akira higuchi 写的一个MySql的插件，通过这个插件，你可以直接跟MySql后端的存储引擎做key-value式的交互，省去了MySql上层的SQL解释、打开关闭表、创建查询计划等CPU消耗型的开销，按照作者给出的数据可以在数据全部在内存的情况下可以达到75W的QPS查询。具体信息可以看这篇[Blog](http://yoshinorimatsunobu.blogspot.com/2010/10/using-mysql-as-nosql-story-for.html?showComment=1287715880531#c2105763073739952882)，中文介绍可以看这篇文章《[HandlerSocket in action](http://whitesock.javaeye.com/blog/811339)》。

这个东西为什么让我很激动呢？首先性能是程序员的G点，一听高性能你不由地激动，其次，这也解决了缓存跟数据库的一致性问题，因为缓存就在数据库里面，第三，这个东西不仅仅是NoSQL，简单的CRUD你可以通过HandlerSocket，但是复杂的查询你仍然可以走MySql，完全符合我们应用的场景，并且从实际测试来看，性能确实非常优秀。但是呢，这个东西的代价也少不了，例如没有权限检查（未来可能添加）；不能启用MySql的查询缓存，否则会导致数据的不一致；协议设计也不合理，使用\t做分隔符，使用\n做换行符，那么你插入或者更新的字段数据就不能含有这些字符，否则行为将不如预期。

HandlerSocket有一个日本人的[java客户端实现](https://github.com/ahiguti/HandlerSocket-Plugin-for-MySQL)，我去尝试了下，结果发现这玩意完全不具实用性，封装的层次非常原始。因此我自己写了个新的客户端，这就是本文要介绍的HandlerSocket Client for Java，简称[hs4j](http://code.google.com/p/hs4j/)，项目放在了[googlecode](http://code.google.com/p/hs4j/)，代码的网络层复用xmemcached，重新实现了协议和上层接口，目前的状态完全可用，也希望有需要的朋友参与测试。

项目地址：http://code.google.com/p/hs4j/

HS4J的使用很简单，所有的操作都通过HSClient这个接口进行，如我们创建一个客户端对象：

|  |
| --- |
| import com.google.code.hs4j.HSClient; |
| import com.google.code.hs4j.impl.HSClientImpl; | |

|  |
| --- |
|  |
| HSClient hsClient = new HSClientImpl(new InetSocketAddress(9999)); | |

假设HandlerSocket运行在本地的9999端口，默认的9998是只读的，9999才是允许读和写。HSClient是线程安全的。

在执行操作前需要先open index：

|  |  |
| --- | --- |
| import com.google.code.hs4j.IndexSession; | |
|  |

|  |  |
| --- | --- |
| IndexSession session = hsClient.openIndexSession(db, table, | |
| "PRIMARY", columns); |

其中db是数据库名,table是表名，”PRIMARY”表示使用主键索引，columns是一个字符串数组代表你要查询的字段名称。这里没有指定 indexid，默认会产生一个indexid，你也可以指定indexid，返回表示一次open-index会话对象，IndexSession同样是线程安全的。

|  |  |
| --- | --- |
| IndexSession session = hsClient.openIndexSession(indexid,db, table, | |
| "PRIMARY", columns); |

查询操作通过find方法:

|  |  |
| --- | --- |
| import java.sql.ResultSet; | |
|  |

|  |  |
| --- | --- |
| final String[] keys = { "dennis", "killme2008@gmail.com" }; | |
| ResultSet rs = session.find(keys); |

|  |
| --- |
| while(rs.next()){ |
| String name=rs.getString(1); | |

|  |  |
| --- | --- |
| String mail=rs.getString(2); | |
| } |

find返回的是java.sql.ResultSet，你完全可以像使用jdbc那样去操作结果集。当然我的简单实现并不符合JDBC规范，只实现了最常见的一些方法，如getStrng、getLong等。find(keys)方法默认使用的op是”=”。其他重载方法可以设置其他类型的op，统一封装为枚举类型FindOperator。

更新操作：

|  |  |
| --- | --- |
| import com.google.code.hs4j.FindOperator; | |
|  |

|  |
| --- |
| int result=session.update(keys, new String[] { "1", "dennis", |
| "test@163.com", "109" }, FindOperator.EQ); | |

keys表示索引的字段列表对应的值数组，通过FindOperator.EQ比较这些值和索引，第二个参数values表示要更新的字段值，这些值跟你在open-index的时候传入的columns一一对应，最后返回作用的记录数。

删除操作：

|  |  |
| --- | --- |
| int result= session.delete(new String[] { "dennis" }, | |
| FindOperator.EQ) |

HS4J同样支持连接池，可以在构建客户端的时候传入连接池大小：

|  |
| --- |
| //100-connections pool |
| HSClient hsClient = new HSClientImpl(new InetSocketAddress(9999),100); | |

在open index的时候，会在连接池里所有的连接上都open。并且在连接因为意外情况（如网络错误）断开的时候，HS4J会自动重连，并在重连成功的情况下自动发送已经open的index，保证应用的操作不受重连影响。

因为HS4J是我在两天内写就的一个东西，可能还有不少隐藏的bug，并且HandlerSocket本身也是个新东西，如果有什么问题或者改进建议，随时欢迎告诉我，多谢。