1.HBase读写的方式概况

主要分为：

1. 纯Java API读写HBase的方式；
2. Spark读写HBase的方式；
3. Flink读写HBase的方式；
4. HBase通过Phoenix读写的方式；

第一种方式是HBase自身提供的比较原始的高效操作方式，而第二、第三则分别是Spark、Flink集成HBase的方式，最后一种是第三方插件Phoenix集成的JDBC方式，Phoenix集成的JDBC操作方式也能在Spark、Flink中调用。

注意：

这里我们使用HBase2.1.2版本，以下代码都是基于该版本开发的。

2. 纯Java API读写HBase

2.1 连接HBase

这里我们采用静态方式连接HBase，不同于2.1.2之前的版本，无需创建HBase线程池，HBase2.1.2提供的代码已经封装好，只需创建调用即可：

```
/**
 * 声明静态配置
 */
static Configuration conf = null;
static Connection conn = null;
static {
        conf = HBaseConfiguration.create();
        conf.set("hbase.zookeeper.quorum", "hadoop01,hadoop02,hadoop03");
        conf.set("hbase.zookeeper.property.client", "2181");
        try{
            conn = ConnectionFactory.createConnection(conf);
        }catch (Exception e){
            e.printStackTrace();
        }
}
```

2.2 创建HBase的表

创建HBase表，是通过Admin来执行的，表和列簇则是分别通过TableDescriptorBuilder和ColumnFamilyDescriptorBuilder来构建。

```
/**
 * 创建只有一个列簇的表
 * @throws Exception
 */
public static void createTable() throws Exception{
    Admin admin = conn.getAdmin();
    if (!admin.tableExists(TableName.valueOf("test"))){
        TableName tableName = TableName.valueOf("test");
        //表描述器构造器
        TableDescriptorBuilder tdb = TableDescriptorBuilder.newBuilder(tableName);
        //列族描述器构造器
        ColumnFamilyDescriptorBuilder cdb = ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes("user"));
        //获得列描述器
        ColumnFamilyDescriptor cfd = cdb.build();
        //添加列族
        tdb.setColumnFamily(cfd);
        //获得表描述器
        TableDescriptor td = tdb.build();
        //创建表
        admin.createTable(td);
    }else {
        System.out.println("表已存在");
    }
    //关闭连接
}
```

2.3 HBase表添加数据

通过put api来添加数据

```
/**
 * 添加数据（多个rowKey，多个列族）
 * @throws Exception
 */
public static void insertMany() throws Exception{
    Table table = conn.getTable(TableName.valueOf("test"));
    List<Put> puts = new ArrayList<Put>();
    Put put1 = new Put(Bytes.toBytes("rowKey1"));
    put1.addColumn(Bytes.toBytes("user"), Bytes.toBytes("name"), Bytes.toBytes("wd"));

    Put put2 = new Put(Bytes.toBytes("rowKey2"));
    put2.addColumn(Bytes.toBytes("user"), Bytes.toBytes("age"), Bytes.toBytes("25"));

    Put put3 = new Put(Bytes.toBytes("rowKey3"));
    put3.addColumn(Bytes.toBytes("user"), Bytes.toBytes("weight"), Bytes.toBytes("60kg"));

    Put put4 = new Put(Bytes.toBytes("rowKey4"));
    put4.addColumn(Bytes.toBytes("user"), Bytes.toBytes("sex"), Bytes.toBytes("男"));

    puts.add(put1);
    puts.add(put2);
    puts.add(put3);
    puts.add(put4);
    table.put(puts);
    table.close();
}
```

2.4 删除HBase的列簇或列

```
/**
 * 根据rowKey删除一行数据，或者删除某一行的某个列簇，或者某一行某个列簇某列
 * @param tableName
 * @param rowKey
 * @throws Exception
 */
public static void deleteData(TableName tableName, String rowKey, String rowKey, String columnFamily, String columnName) throws Exception{
    Table table = conn.getTable(tableName);
    Delete delete = new Delete(Bytes.toBytes(rowKey));
    //①根据rowKey删除一行数据
    table.delete(delete);

    //②删除某一行的某一个列簇内容
    delete.addFamily(Bytes.toBytes(columnFamily));

    //③删除某一行某个列簇某列的值
    delete.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(columnName));
    table.close();
}
```

2.5 更新HBase表的列

使用Put api直接替换掉即可

```
/**
 * 根据RowKey ， 列簇， 列名修改值
 * @param tableName
 * @param rowKey
 * @param columnFamily
 * @param columnName
 * @param columnValue
 * @throws Exception
 */
public static void updateData(TableName tableName, String rowKey, String columnFamily, String columnName, String columnValue) throws Exception{
    Table table = conn.getTable(tableName);
    Put put1 = new Put(Bytes.toBytes(rowKey));
    put1.addColumn(Bytes.toBytes(columnFamily), Bytes.toBytes(columnName), Bytes.toBytes(columnValue));
    table.put(put1);
    table.close();
}
```

2.6 HBase查询

HBase查询分为get、scan、scan和hfiter结合。hfiter过滤器又分为RowFilter（rowKey过滤器）、SingleColumnValueFilter（列值过滤器）、ColumnPrefixFilter（列名前缀过滤器）。

```
/**
 * 根据rowKey查询数据
 * @param tableName
 * @param rowKey
 * @throws Exception
 */
public static void getResult(TableName tableName, String rowKey) throws Exception{
    Table table = conn.getTable(tableName);
    //获得一行
    Get get = new Get(Bytes.toBytes(rowKey));
    Result set = table.get(get);
    Cell[] cells = set.rawCells();
    for (Cell cell: cells){
        System.out.println(Bytes.toString(cell.getQualifierArray(), cell.getQualifierOffset(), cell.getQualifierLength()) + ":" +
        Bytes.toString(cell.getValueArray(), cell.getValueOffset(), cell.getValueLength()));
    }
    table.close();
}

//过滤器 LESS <  LESS_OR_EQUAL <=   EQUAL =   NOT_EQUAL <>   GREATER_OR_EQUAL >=   GREATER >   NO_OP 排除所有

/**
 * @param tableName
 * @throws Exception
 */
public static void scanTable(TableName tableName) throws Exception{
    Table table = conn.getTable(tableName);

    //①全表扫描
    Scan scan1 = new Scan();
    ResultScanner rscan1 = table.getScanner(scan1);

    //②rowKey过滤器
    Scan scan2 = new Scan();
    //str$ 末尾匹配，相当于sql中的 %str   ^str开头匹配，相当于sql中的str%
    RowFilter filter = new RowFilter(CompareOperator.EQUAL, new RegexStringComparator("Key1$"));
    scan2.setFilter(filter);
    ResultScanner rscan2 = table.getScanner(scan2);

    //③列值过滤器
    Scan scan3 = new Scan();
    //下列参数分别为列族，列名，比较符号，值
    SingleColumnValueFilter filter3 = new SingleColumnValueFilter(Bytes.toBytes("author"), Bytes.toBytes("name"),
                CompareOperator.EQUAL, Bytes.toBytes("spark"));
    scan3.setFilter(filter3);
    ResultScanner rscan3 = table.getScanner(scan3);

    //列名前缀过滤器
    Scan scan4 = new Scan();
    ColumnPrefixFilter filter4 = new ColumnPrefixFilter(Bytes.toBytes("name"));
    scan4.setFilter(filter4);
    ResultScanner rscan4 = table.getScanner(scan4);

    //过滤器集合
    Scan scan5 = new Scan();
    FilterList list = new FilterList(FilterList.Operator.MUST_PASS_ALL);
    SingleColumnValueFilter filter51 = new SingleColumnValueFilter(Bytes.toBytes("author"), Bytes.toBytes("name"),
            CompareOperator.EQUAL, Bytes.toBytes("spark"));
    ColumnPrefixFilter filter52 = new ColumnPrefixFilter(Bytes.toBytes("name"));
    list.addFilter(filter51);
    list.addFilter(filter52);
    scan5.setFilter(list);
    ResultScanner rscan5 = table.getScanner(scan5);

    for (Result rs : rscan){
        String rowKey = Bytes.toString(rs.getRow());
        System.out.println("row key :" + rowKey);
        Cell[] cells = rs.rawCells();
        for (Cell cell: cells){
            System.out.println(Bytes.toString(cell.getFamilyArray(), cell.getFamilyOffset(), cell.getFamilyLength()) + ":"
                    + Bytes.toString(cell.getQualifierArray(), cell.getQualifierOffset(), cell.getQualifierLength()) + ":"
                    + Bytes.toString(cell.getValueArray(), cell.getValueOffset(), cell.getValueLength()));
        }
        System.out.println("----------------------------------------");
    }
}
```