中国民生银行 HBase 读写设计与实践

背景介绍

本项目主要解决 check 和 opinion2 张历史数据表（历史数据是指当业务发生过程中的完整中间流程和结果数据）的在线查询。原实现基于 Oracle 提供存储查询服务，随着数据量的不断增加，在写入和读取过程中面临性能问题，且历史数据仅供业务查询参考，并不影响实际流程，从系统结构上来说，放在业务链条上游比较重。本项目将其置于下游数据处理 Hadoop 分布式平台来实现此需求。下面列一些具体的需求指标：

1. 数据量：目前 check 表的累计数据量为 5000w+ 行，11GB；opinion 表的累计数据量为 3 亿 +，约 100GB。每日增量约为每张表 50 万 + 行，只做 insert，不做 update。
2. 查询要求：check 表的主键为 id（Oracle 全局 id），查询键为 check_id，一个 check_id 对应多条记录，所以需返回对应记录的 list； opinion 表的主键也是 id，查询键是 bussiness_no 和 buss_type，同理返回 list。单笔查询返回 List 大小约 50 条以下，查询频率为 100 笔 / 天左右，查询响应时间 2s。

技术选型

从数据量及查询要求来看，分布式平台上具备大数据量存储，且提供实时查询能力的组件首选 HBase。根据需求做了初步的调研和评估后，大致确定 HBase 作为主要存储组件。将需求拆解为写入和读取 HBase 两部分。

读取 HBase 相对来说方案比较确定，基本根据需求设计 RowKey，然后根据 HBase 提供的丰富 API（get，scan 等）来读取数据，满足性能要求即可。

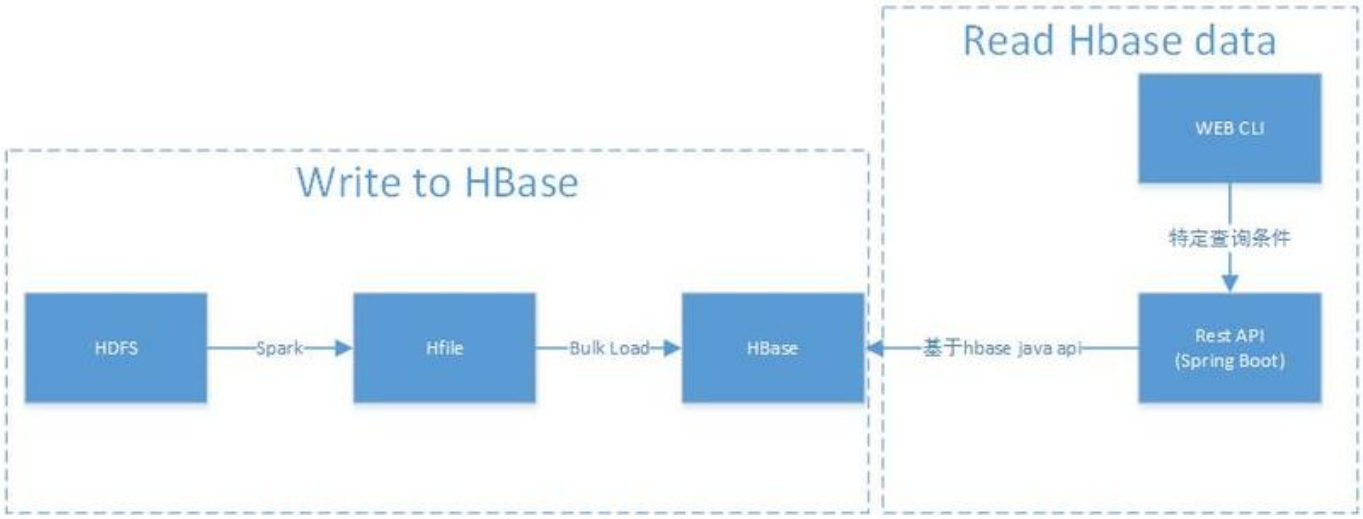
写入 HBase 的方法大致有以下几种：

1. Java 调用 HBase 原生 API，HTable.add(List(Put))。
2. MapReduce 作业，使用 TableOutputFormat 作为输出。
3. Bulk Load，先将数据按照 HBase 的内部数据格式生成持久化的 HFile 文件，然后复制到合适的位置并通知 RegionServer ，即完成海量数据的入库。其中生成 Hfile 这一步可以选择 MapReduce 或 Spark。

本文采用第 3 种方式，Spark + Bulk Load 写入 HBase。该方法相对其他 2 种方式有以下优势：

1. BulkLoad 不会写 WAL，也不会产生 flush 以及 split。
2. 如果我们大量调用 PUT 接口插入数据，可能会导致大量的 GC 操作。除了影响性能之外，严重时甚至可能会对 HBase 节点的稳定性造成影响，采用 BulkLoad 无此顾虑。
3. 过程中没有大量的接口调用消耗性能。
4. 可以利用 Spark 强大的计算能力。

图示如下：



如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

设计

环境信息

Hadoop 2.5-2.7
HBase 0.98.6
Spark 2.0.0-2.1.1
Sqoop 1.4.6

表设计

本段的重点在于讨论 HBase 表的设计，其中 RowKey
是最重要的部分。为了方便说明问题，我们先来看看数据格式。以下以 check 举例，opinion
同理。

check 表（原表字段有 18 个，为方便描述，本文截选 5 个字段示意）

id	check_id	rule_no	rule_type	rule_desc
59C1A1FDCBBE01ECE053C5013807B6BE	A208848994	0001	ABCD	规则01
59C1A1FDCBBF01ECE053C5013807B6BE	A208848994	0002	CDBA	规则03
59C1A1FDCBC001ECE053C5013807B6BE	A208848994	0002	CDBB	规则02

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

如上图所示，主键为 id，32 位字母和数字随机组成，业务查询字段 check_id
为不定长字段（不超过 32 位），字母和数字组成，同一 check_id
可能对应多条记录，其他为相关业务字段。众所周知，HBase 是基于 RowKey 提供查询，且要求
RowKey 是唯一的。RowKey 的设计主要考虑的是数据将怎样被访问。初步来看，我们有 2
种设计方法。

1. 拆成 2 张表，一张表 id 作为 RowKey，列为 check
   表对应的各列；另一张表为索引表，RowKey 为 check_id，每一列对应一个
   id。查询时，先找到 check_id 对应的 id list，然后根据 id 找到对应的记录。均为 HBase
   的 get 操作。
2. 将本需求可看成是一个范围查询，而不是单条查询。将 check_id 作为 RowKey
   的前缀，后面跟 id。查询时设置 Scan 的 startRow 和 stopRow，找到对应的记录 list。

第一种方法优点是表结构简单，RowKey 容易设计，缺点为
1）数据写入时，一行原始数据需要写入到 2 张表，且索引表写入前需要先扫描该 RowKey
是否存在，如果存在，则加入一列，否则新建一行，2）读取的时候，即便是采用 List,
也至少需要读取 2 次表。第二种设计方法，RowKey
设计较为复杂，但是写入和读取都是一次性的。综合考虑，我们采用第二种设计方法。

RowKey 设计

热点问题

HBase 中的行是以 RowKey 的字典序排序的，其热点问题通常发生在大量的客户端直接访问集群
的一个或极少数节点。默认情况下，在开始建表时，表只会有一个 region，并随着 region
增大而拆分成更多的 region，这些 region 才能分布在多个 regionserver
上从而使负载均分。对于我们的业务需求，存量数据已经较大，因此有必要在一开始就将 HBase
的负载均摊到每个 regionserver，即做 pre-split。常见的防治热点的方法为加盐，hash
散列，自增部分（如时间戳）翻转等。

RowKey 设计

Step1：确定预分区数目，创建 HBase Table

不同的业务场景及数据特点确定数目的方式不一样，我个人认为应该综合考虑数据量大小和集群
大小等因素。比如 check 表大小约为 11G，测试集群大小为 10
台机器，hbase.hregion.max.filesize=3G（当 region 的大小超过这个数时，将拆分为 2
个），所以初始化时尽量使得一个 region 的大小为 1~2G（不会一上来就 split），region
数据分到 11G/2G=6 个，但为了充分利用集群资源，本文中 check 表划分为 10
个分区。如果数据量为 100G，且不断增长，集群情况不变，则 region 数目增大到 100G/2G=50
个左右较合适。Hbase check 表建表语句如下：

```
create 'tinawang:check',
{ NAME => 'f', COMPRESSION => 'SNAPPY',DATA_BLOCK_ENCODING => 'FAST_DIFF',BLOOMFILTER=>'ROW'},
{SPLITS => [ '1','2','3', '4','5','6','7','8','9']}
```

其中，Column Family ='f'，越短越好。

COMPRESSION => 'SNAPPY'，HBase 支持 3 种压缩 LZO, GZIP and Snappy。GZIP 压缩率高，但是耗 CPU。后两者差不多，Snappy 稍微胜出一点，cpu 消耗的比 GZIP 少。一般在 IO 和 CPU 均衡下，选择 Snappy。

DATA_BLOCK_ENCODING => 'FAST_DIFF'，本案例中 RowKey 较为接近，通过以下命令查看 key 长度相对 value 较长。

./hbase org.apache.hadoop.hbase.io.hfile.HFile -m -f /apps/hbase/data/data/tinawang/check/a661f0f95598662a53b3d8b1ae469fdf/f/a5fefc880f87492d908672e1634f2eed_SeqId_2_

```
    compression=none,
    cacheConf=CacheConfig:enabled [cacheDataOnRead=true] [cacheDataOnWrite=false] [cacheIndexesOnWrite=false] [cacheBloomsOnWrite=false] [cacheEvictOnClose=false] [cacheCompressed=false][prefetchOnOpen=false],
    firstKey=900052453daa98926457d0c593b34ca5a59C1A2421E9B01ECE053C5013807B6BE/f:artificial_ver_result/1507040642943/Put,
    lastKey=9fffdc7330c99997760f7cb3fa2aabb1759C1A2196CEC01ECE053C5013807B6BE/f:update_date/1507040642943/Put,
    avgKeyLen=87,
    avgValueLen=10,
    entries=10574712,
    length=1131004289
```

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Step2：RowKey 组成
Salt

让数据均衡的分布到各个 Region 上，结合 pre-split，我们对查询键即 check 表的 check_id 求 hashcode 值，然后 modulus(numRegions) 作为前缀，注意补齐数据。

StringUtils.leftPad(Integer.toString(Math.abs(check_id.hashCode() % numRegion)),1,'0')

说明：如果数据量达上百 G 以上，则 numRegions 自然到 2 位数，则 salt 也为 2 位。

Hash 散列

因为 check_id 本身是不定长的字符数字串，为使数据散列化，方便 RowKey 查询和比较，我们对 check_id 采用 SHA1 散列化，并使之 32 位定长化。

MD5Hash.getMD5AsHex(Bytes.toBytes(check_id))

唯一性

以上 salt+hash 作为 RowKey 前缀，加上 check 表的主键 id 来保障 RowKey 唯一性。综上，check 表的 RowKey 设计如下：（check_id=A208849559）

Salt（即Region 分区号，1位）	Hash（check_id），32位	Id，32位
7	7c9498b4a83974da56b252122b9752bf	56B63AB98C2E00B4E053C501380709AD

为增强可读性，中间还可以加上自定义的分割符，如'+','|'等。

7+7c9498b4a83974da56b252122b9752bf+56B63AB98C2E00B4E053C501380709AD

以上设计能保证对每次查询而言，其 salt+hash 前缀值是确定的，并且落在同一个 region 中。需要说明的是 HBase 中 check 表的各列同数据源 Oracle 中 check 表的各列存储。

WEB 查询设计

RowKey 设计与查询息息相关，查询方式决定 RowKey 设计，反之基于以上 RowKey 设计，查询时通过设置 Scan 的 [startRow，stopRow], 即可完成扫描。以查询 check_id=A208849559 为例，根据 RowKey 的设计原则，对其进行 salt+hash 计算，得前缀。

startRow = 7+7c9498b4a83974da56b252122b9752bf
stopRow = 7+7c9498b4a83974da56b252122b9752bg

代码实现关键流程

Spark write to HBase

Step0: prepare work

因为是从上游系统承接的业务数据，存量数据采用 sqoop 抽到
hdfs；增量数据每日以文件的形式从 ftp 站点获取。因为业务数据字段中包含一些换行符，且
sqoop1.4.6 目前只支持单字节，所以本文选择'0x01'作为列分隔符，'0x10'作为行分隔符。

Step1: Spark read hdfs text file

```
Configuration hadoopConfig = new Configuration(sc.hadoopConfiguration());
hadoopConfig.set("textinputformat.record.delimiter", Constants.lineSepAsciiStr);
// read input file line into Value obj Text,with specified line seperating chara
cter "0x10"
String path = this.table.getInputPathPrefix() + "/" + batchDate;
JavaRDD<String> textRDD = sc.newAPIHadoopFile(
                path, TextInputFormat.class, LongWritable.class, Text.class, had
oopConfig)
                .map( t2 -> t2._2.toString());
```

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

SparkContext.textfile() 默认行分隔符为"₩n"，此处我们用"0x10"，需要在 Configuration
中配置。应用配置，我们调用 newAPIHadoopFile 方法来读取 hdfs 文件，返回
JavaPairRDD，其中 LongWritable 和 Text 分别为 Hadoop 中的 Long 类型和 String 类型（所有
Hadoop 数据类型和 java
的数据类型都很相像，除了它们是针对网络序列化而做的特殊优化）。我们需要的数据文件放在
pairRDD 的 value 中，即 Text 指代。为后续处理方便，可将 JavaPairRDD 转换为 JavaRDD。

Step2: Transfer and sort RDD

① 将 avaRDD 转换成
JavaPairRDD，其中参数依次表示为，RowKey，col，value。做这样转换是因为 HBase
的基本原理是基于 RowKey 排序的，并且当采用 bulk load
方式将数据写入多个预分区（region）时，要求 Spark 各 partition
的数据是有序的，RowKey，column family（cf），col name
均需要有序。在本案例中因为只有一个列簇，所以将 RowKey 和 col name 组织出来为 Tuple2
格式的 key。请注意原本数据库中的一行记录（n 个字段），此时会被拆成 n 行。

② 基于 JavaPairRDD 进行 RowKey，col 的二次排序。如果不做排序，会报以下异常：

java.io.IOException: Added a key notlexically larger than previous key

③ 将数据组织成 HFile 要求的 JavaPairRDDhfileRDD。

Step3：create hfile and bulk load to HBase

①主要调用 saveAsNewAPIHadoopFile 方法：

```
hfileRdd.saveAsNewAPIHadoopFile(hfilePath,ImmutableBytesWritable.class,
KeyValue.class,HFileOutputFormat2.class,config);
```

② hfilebulk load to HBase

```
final Job job = Job.getInstance();
job.setMapOutputKeyClass(ImmutableBytesWritable.class);
job.setMapOutputValueClass(KeyValue.class);
HFileOutputFormat2.configureIncrementalLoad(job,htable);
LoadIncrementalHFiles bulkLoader = newLoadIncrementalHFiles(config);
bulkLoader.doBulkLoad(newPath(hfilePath),htable);
```

注：如果集群开启了 kerberos，step4 需要放置在 ugi.doAs（）方法中，在进行如下验证后实现

```
UserGroupInformation ugi = UserGroupInformation.loginUserFromKeytabAndReturnUGI(keyU
ser,keytabPath);
UserGroupInformation.setLoginUser(ugi);
```

访问 HBase 集群的 60010 端口 web，可以看到 region 分布情况。

Table Regions

Name	Region Server	Start Key	End Key	Requests
check,,1506742420309.838d63be11b5095c1836bbb857fca254.	BIGLBTMP:60020		1	161219
check,1,1506742420309.d7d633a318156d432f488a0221e1e95b.	BIGL8TMP:60020	1	2	153938
check,2,1506742420309.51c2ae8985810238a9e4979ed7ddbf58.	BIGL6TMP:60020	2	3	115519
check,3,1506742420309.6ca837a5acbdee82abdba63fd9ee02b8.	BIGL10TMP:60020	3	4	145442
check,4,1506742420309.3ffcc1cf86fec312e6f988432f145c5c.	BIGL5TMP:60020	4	5	143826
check,5,1506742420309.4d55640e26a80b604d19e9da55449f77.	BIGL7TMP:60020	5	6	130968
check,6,1506742420309.69fd3da092537ef6a608cb8993768236.	BIGL4TMP:60020	6	7	136994
check,7,1506742420309.64c4455ca0950079020acc10074baad5.	BIGLATMP:60020	7	8	138191
check,8,1506742420309.ff4515edd27d69b63df6f5237749dd39.	BIGL3TMP:60020	8	9	159656
check,9,1506742420309.3f6c3295f43a6a25dbf38a97ee9ee5f8.	BIGL9TMP:60020	9		122886

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

Read from HBase

本文基于 spring boot 框架来开发 web 端访问 HBase 内数据。

use connection pool(使用连接池)

创建连接是一个比较重的操作，在实际 HBase 工程中，我们引入连接池来共享 zk 连接，meta 信息缓存，region server 和 master 的连接。

```
HConnection connection = HConnectionManager.createConnection(config);
HTableInterface table = connection.getTable("table1");
try {
 // Use the table as needed, for a single operation and a single thread
} finally {
 table.close();
}
```

也可以通过以下方法，覆盖默认线程池。

```
HConnection createConnection(Configuration conf,ExecutorService pool);
```

process query

Step1: 根据查询条件，确定 RowKey 前缀

根据 3.3 RowKey 设计介绍，HBase 的写和读都遵循该设计规则。此处我们采用相同的方法，将 web 调用方传入的查询条件，转化成对应的 RowKey 前缀。例如，查询 check 表传递过来的 check_id=A208849559，生成前缀 7+7c9498b4a83974da56b252122b9752bf。

Step2：确定 scan 范围

A208849559 对应的查询结果数据即在 RowKey 前缀为 7+7c9498b4a83974da56b252122b9752bf 对应的 RowKey 及 value 中。

```
scan.setStartRow(Bytes.toBytes(rowkey_pre)); //scan, 7+7c9498b4a83974da56b252122b9752bf
byte[] stopRow = Bytes.toBytes(rowkey_pre);
stopRow[stopRow.length-1]++;
scan.setStopRow(stopRow);// 7+7c9498b4a83974da56b252122b9752bg
```

Step3：查询结果组成返回对象

遍历 ResultScanner 对象，将每一行对应的数据封装成 table entity，组成 list 返回。

测试

从原始数据中随机抓取 1000 个 check_id，用于模拟测试，连续发起 3 次请求数为 2000（200 个线程并发，循环 10 次），平均响应时间为 51ms，错误率为 0。

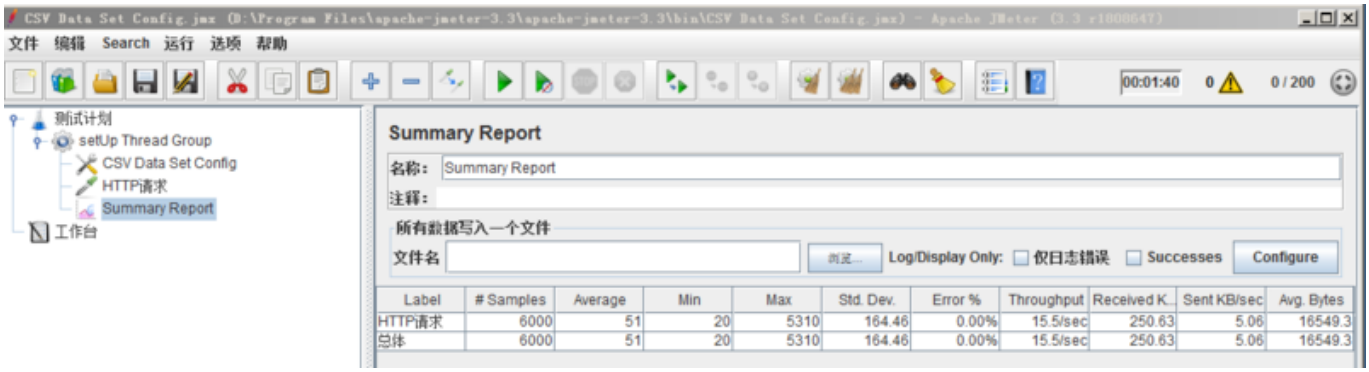


Table Regions

Name	Region Server	Start Key	End Key	Requests
:check,,1506742420309.838d63be11b5095c1836bbb857fca254.	BIGLBTMP:60020		1	161219
:check,1,1506742420309.d7d633a318156d432f488a0221e1e95b.	BIGL8TMP:60020	1	2	153938
:check,2,1506742420309.51c2ae8985810238a9e4979ed7ddbf58.	BIGL6TMP:60020	2	3	115519
:check,3,1506742420309.6ca837a5acbdee82abdba63fd9ee02b8.	BIGL10TMP:60020	3	4	145442
:check,4,1506742420309.3ffcc1cf86fec312e6f988432f145c5c.	BIGL5TMP:60020	4	5	143826
:check,5,1506742420309.4d55640e26a80b604d19e9da55449f77.	BIGL7TMP:60020	5	6	130968
:check,6,1506742420309.69fd3da092537ef6a608cb8993768236.	BIGL4TMP:60020	6	7	136994
:check,7,1506742420309.64c4455ca0950079020acc10074baad5.	BIGLATMP:60020	7	8	138191
:check,8,1506742420309.ff4515edd27d69b63df6f5237749dd39.	BIGL3TMP:60020	8	9	159656
:check,9,1506742420309.3f6c3295f43a6a25dbf38a97ee9ee5f8.	BIGL9TMP:60020	9		122886

如果想及时了
解Spark、Hadoop或者Hbase相关的文章，欢迎关注微信公共帐号：iteblog_hadoop

如上图，经历 N 次累计测试后，各个 region 上的 Requests
数较为接近，符合负载均衡设计之初。

踩坑记录

kerberos 认证问题

如果集群开启了安全认证，那么在进行 Spark 提交作业以及访问 HBase 时，均需要进行 kerberos 认证。

本文采用 yarn cluster 模式，像提交普通作业一样，可能会报以下错误。

```
ERROR StartApp: job failure,
java.lang.NullPointerException
 at com.tinawang.spark.hbase.utils.HbaseKerberos.<init>(HbaseKerberos.java:18)
 at com.tinawang.spark.hbase.job.SparkWriteHbaseJob.run(SparkWriteHbaseJob.java:60)
```

定位到 HbaseKerberos.java:18，代码如下：

```
this.keytabPath = (Thread.currentThread().getContextClassLoader().getResource(prop.getProperty("hbase.keytab"))).getPath();
```

这是因为 executor 在进行 HBase 连接时，需要重新认证，通过 --keytab 上传的 tina.keytab 并未被 HBase 认证程序块获取到，所以认证的 keytab 文件需要另外通过 --files 上传。示意如下

```
--keytab /path/tina.keytab ₩
--principal tina@GNUHPC.ORG ₩
--files "/path/tina.keytab.hbase"
```

其中 tina.keytab.hbase 是将 tina.keytab 复制并重命名而得。因为 Spark 不允许同一个文件重复上传。

序列化

```
org.apache.spark.SparkException: Task not serializable
 at org.apache.spark.util.ClosureCleaner$.ensureSerializable(ClosureCleaner.scala:298)
 at org.apache.spark.util.ClosureCleaner$.org$apache$spark$util$ClosureCleaner$$clean(Clos
ureCleaner.scala:288)
 at org.apache.spark.util.ClosureCleaner$.clean(ClosureCleaner.scala:108)
 at org.apache.spark.SparkContext.clean(SparkContext.scala:2101)
 at org.apache.spark.rdd.RDD$$anonfun$map$1.apply(RDD.scala:370)
 at org.apache.spark.rdd.RDD$$anonfun$map$1.apply(RDD.scala:369)
 ...
org.apache.spark.deploy.yarn.ApplicationMaster$$anon$2.run(ApplicationMaster.scala:637)
Caused by: java.io.NotSerializableException: org.apache.spark.api.java.JavaSparkContext
Serialization stack:
 - object not serializable (class: org.apache.spark.api.java.JavaSparkContext, value: org.apache.
spark.api.java.JavaSparkContext@24a16d8c)
 - field (class: com.tinawang.spark.hbase.processor.SparkReadFileRDD, name: sc, type: class or
g.apache.spark.api.java.JavaSparkContext)
...
```

解决方法一：

如果 sc 作为类的成员变量，在方法中被引用，则加 transient 关键字，使其不被序列化。

```
private transient JavaSparkContext sc;
```

解决方法二：

将 sc 作为方法参数传递，同时使涉及 RDD 操作的类 implements Serializable。
代码中采用第二种方法。详见代码。

批量请求测试

Exception in thread "http-
nio-8091-Acceptor-0" java.lang.NoClassDefFoundError: org/apache/tomcat/util/ExceptionUtils

或者

Exception in thread "http-nio-8091-exec-34" java.lang.NoClassDefFoundError: ch/qos/logback/
classic/spi/ThrowableProxy

查看下面 issue 以及一次排查问题的过程，可能是 open file 超过限制。

https://github.com/spring-projects/spring-boot/issues/1106
http://mp.weixin.qq.com/s/34GVlaYDOdY1OQ9eZs-iXg

使用 ulimit-a 查看每个用户默认打开的文件数为 1024。

在系统文件 /etc/security/limits.conf 中修改这个数量限制，在文件中加入以下内容,
即可解决问题。

- soft nofile 65536
- hard nofile 65536

作者介绍

汪婷，中国民生银行大数据开发工程师，专注于 Spark 大规模数据处理和 Hbase 系统设计。

参考文献

http://hbase.apache.org/book.html#perf.writing
http://www.opencore.com/blog/2016/10/efficient-bulk-load-of-hbase-using-spark/
http:/hbasefly.com/2016/03/23/hbase_writer/
https://github.com/spring-projects/spring-boot/issues/1106
http://mp.weixin.qq.com/s/34GVlaYDOdY1OQ9eZs-iXg

本文链接: 【】（）