数据迁移

目标

- 能够描述项目数据迁移的方案
- 了解hbase的特点
- 能够熟悉数据迁移中的数据包装和转换
- 能够完成文章数据的全量和增量迁移
- 能够完成热点文章数据的迁移

1 为什么需要自动同步

因为我们Mysql保存着我们爬取的以及自建的数据，对于爬取的数据，数据量比较大，使用mysql 存储会影响mysql的性能，并且我们需要对数据进行流式计算，对数据进行各种统计，mysq满足不了我们的需求，我们就将mysql中的全量数据同步到HBASE中，由HBASE保存海量数据，mysql中的全量数据会定期进行删除。

HBASE中保存着海量数据，我们需要计算出热点数据，并将数据同步到mysql以及MONGODB中，mysql中保存主体关系数据，MONGODB保存着具体数据信息。

因为热点数据也会失效，今天是热数据，明天就不是了，也需要定期对热点数据进行删除，我们定时删除一个月之前的热点数据，保持本月的热数据。

2 迁移方案

2.1 需求分析

2.1.1 功能需求

有了大量数据集基础后，实时计算后的热点数据需要保存起来，因为mysql保存大量文章数据会影响mysql的性能，所以采用mysql+mongoDB的方式进行存储。

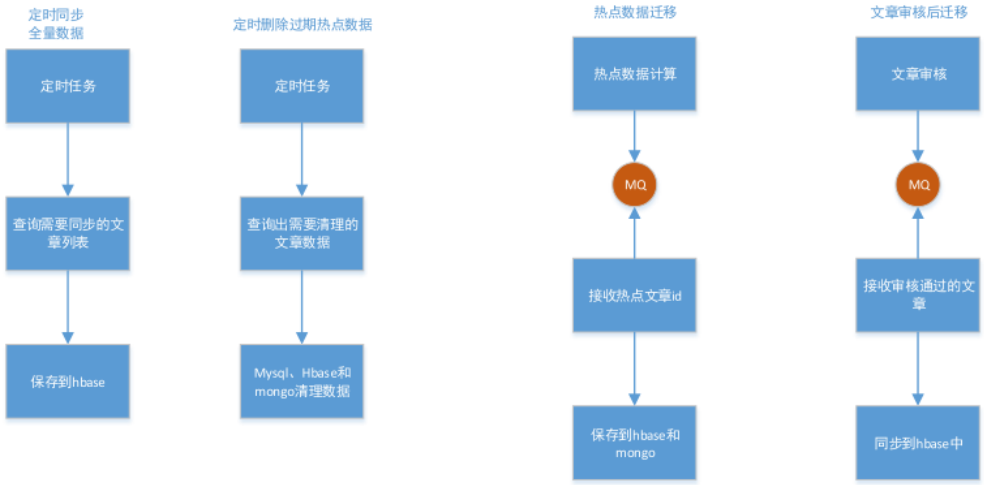
2.1.1 全量数据迁移方案

通过定时任务将mysql中爬取或者自建的文章同步到HBASE中，并将同步过的数据状态改为已同步，下次同步的时候就不会再次同步这些数据了。

关于hbase的学习请参考资料文件夹中的关于hbase相关的资料

2.1.2 热数据迁移方案

HBASE中有全量数据，大数据端计算出热点数据，需要将这些热点数据同步到MYSQL和MONGDB中，用于页面显示



2.2 设计思路

- 将mysql数据库中的全量数据定时读取出来，将多个对象打包成一个对象，保存到HBASE中，保存成功后更新数据库中的状态改为已同步，下一次就不会同步该条数据了。
- 使用KAFKA监听热点数据计算结果，接收到热点数据信息后，从HBASE得到打包的数据，并将数据进行拆分，将关系数据保存到mysql中，将具体数据保存到mongodb中。
- 因为热点数据会失效，定期清除mysql和mongodb中的过期数据

2.3 数据同步注意的问题

- HBASE数据主要靠rowKey进行查询的，rowKey设计就用mysql中的主键ID作为rowKey，查询的时候直接根据Rowkey获取数据
- 因为需要同步到HBASE的数据是多个数据表的数据，一条数据由多个对象组成，存储的时候使用列族区分不同的对象，里面存储不同的字段。

3 项目中集成hbase与Mongodb

在heima-leadnews-common 集成

hbase.properties

```
hbase.zookeeper.quorum=172.16.1.52:2181
hbase.client.keyvalue.maxsize=500000
```

mongo.properties

```
#mongoDB 连接
mongo.host=47.94.7.85
mongo.port=27017
mongo.dbname=mongoDB
```

pom.xml

```
<!--mongoDB-->
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<!--HBase-->
<dependency>
    <groupId>org.apache.hbase</groupId>
    <artifactId>hbase-client</artifactId>
    <version>2.1.5</version>
    <exclusions>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-api</artifactId>
        </exclusion>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </exclusion>
        <exclusion>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
        </exclusion>
        <exclusion>
            <artifactId>slf4j-log4j12</artifactId>
            <groupId>org.slf4j</groupId>
        </exclusion>
    </exclusions>
</dependency>
```

host文件配置

在服务器host文件中配置域名,根据自己的服务器地址更改

```
172.16.1.52 itcast
```

4 常用组件介绍

4.1 Hbase相关操作

Hbase 操作工具类用于将数据存储到Hbase中，其中有些方法用于存储或删除。

4.1.1 项目导入

导入资料文件夹中的项目heima-leadnews-migration

4.1.2 公共存储类

(1)StorageData

公共存储数据表 由多个StorageEntity组成

StorageData 是最重要的一个存储对象，他是保存一个bean信息的类，负责存储bean信息以及转换和反向转换bean 。

该类用到一个重要的工具类ReflectUtils 反射工具类和DataConvertUtils数据类型转换工具类主要用于日期类型的转换，这两个类位于

代码位置：com.heima.common.common.storage.StorageData

```
/**
 * 存储数据Data
 */
@Setter
@Getter
@ToString
public class StorageData {
    /**
     * 目标class 类名称
     */
    private String targetClassName;
    /**
     * 存储的字段列表
     */
    private List<StorageEntry> entryList = new ArrayList<StorageEntry>();

    /**
     * 添加一个entry
     *
     * @param entry
     */
    public void addStorageEntry(StorageEntry entry) {
        entryList.add(entry);
    }

    /**
     * 添加一个entry
     *
     * @param key
     * @param value
     */
    public void addStorageEntry(String key, String value) {
        entryList.add(new StorageEntry(key, value));
    }

    /**
     * 根据Map 添加entry
     *
     * @param map
     */
    public void putHBaseEntry(Map<String, String> map) {
        if (null != map && !map.isEmpty()) {
            map.forEach((k, v) -> addStorageEntry(new StorageEntry(k, v)));
        }
    }

    /**
     * 获取所有的Column 数组
     *
     * @return
     */
    public String[] getColumns() {
```

```
        List<String> columnList =
entryList.stream().map(StorageEntry::getKey).collect(Collectors.toList());
        if (null != columnList && !columnList.isEmpty()) {
            return columnList.toArray(new String[columnList.size()]);
        }
        return null;
    }

    /**
     * 获取所有的值数字
     *
     * @return
     */
    public String[] getValues() {
        List<String> valueList =
entryList.stream().map(StorageEntry::getValue).collect(Collectors.toList());
        if (null != valueList && !valueList.isEmpty()) {
            return valueList.toArray(new String[valueList.size()]);
        }
        return null;
    }

    /**
     * 获取一个Map
     *
     * @return
     */
    public Map<String, Object> getMap() {
        Map<String, Object> entryMap = new HashMap<String, Object>();
        entryList.forEach(entry -> entryMap.put(entry.getKey(),
entry.getValue()));
        return entryMap;
    }


    /**
     * 将当前的StorageData 转换为具体的对象
     *
     * @return
     */
    public Object getObjectValue() {
        Object bean = null;
        if (StringUtils.isNotEmpty(targetClassName) && null != entryList &&
!entryList.isEmpty()) {
            bean = ReflectUtils.getClassForBean(targetClassName);
            if (null != bean) {
                for (StorageEntry entry : entryList) {
                    Object value = DataConvertUtils.convert(entry.getValue(),
ReflectUtils.getFieldAnnotations(bean, entry.getKey()));
                    ReflectUtils.setPropertie(bean, entry.getKey(), value);
                }
            }
        }
        return bean;
    }

    /**
     * 将一个Bean 转换未为StorageData
```

```
     *
     * @param bean
     * @return
     */
    public static StorageData getStorageData(Object bean) {
        StorageData hbaseData = null;
        if (null != bean) {
            hbaseData = new StorageData();
            hbaseData.setTargetClassName(bean.getClass().getName());
            hbaseData.setEntryList(getStorageEntryList(bean));

        }
        return hbaseData;
    }

    /**
     * 根据bean 获取entry 列表
     *
     * @param bean
     * @return
     */
    private static List<StorageEntry> getStorageEntryList(Object bean) {
        PropertyDescriptor[] propertyDescriptorArray =
ReflectUtils.getPropertyDescriptorArray(bean);
        return
Arrays.asList(propertyDescriptorArray).stream().map(propertyDescriptor -> {
            String key = propertyDescriptor.getName();
            Object value = ReflectUtils.getPropertyDescriptorValue(bean,
propertyDescriptor);
            value = DataConvertUtils.unConvert(value,
ReflectUtils.getFieldAnnotations(bean, propertyDescriptor));
            return new StorageEntry(key, DataConvertUtils.toString(value));
        }).collect(Collectors.toList());
    }

}
```

最主要的几个方法

- 添加StorageEntry方法

```
public void addStorageEntry(StorageEntry entry)
```

该方法有几个重载方法，用于向StorageEntry列表中添加StorageEntry对象的

- 获取该对象对应的Object对象

```
public Object getObjectValue()
```

该方法用于将存储的实体数据转换为Bean的实体，用了ReflectUtils反射工具类进行操作

- 将Bean 转换为StorageData的存储结构

```
public static StorageData getStorageData(Object bean)
```

该方法用于将不同的bean转换为同一种存储结构进行存储

(2)StorageEntity

公共代码存储的实体

代码位置：com.heima.common.common.storage.StorageEntity

```
/**
 * 一个存储的实体
 */
@Setter
@Getter
public class StorageEntity {

    /**
     * 存储类型的列表
     * 一个实体可以存储多个数据列表
     */
    private List<StorageData> dataList = new ArrayList<StorageData>();

    /**
     * 添加一个存储数据
     * @param storageData
     */
    public void addStorageData(StorageData storageData) {
        dataList.add(storageData);
    }
}
```

(3)StorageEntry

公共存储对象的一个key-value的字段

代码位置：com.heima.common.common.storage.StorageEntry

```
/**
 * 存储Entry
 * k-v 结构保存一个对象的字段的字段名和值
 */
@Setter
@Getter
public class StorageEntry {
    /**
     * 空的构造方法
     */
    public StorageEntry() {
    }

    /**
     * 构造方法
     *
     * @param key
     * @param value
     */
    public StorageEntry(String key, String value) {
        this.key = key;
        this.value = value;
    }
```

```
    /**
     * 字段的Key
     */
    private String key;

    /**
     * 字段的Value
     */
    private String value;

}
```

4.1.3 Hbase操作相关工具类

(1)HBaseConstants 类

配置类Hbase存储的的表名称

代码位置 com.heima.hbase.constants.HBaseConstants

```
public class HBaseConstants {

    public static final String APARTICLE_QUANTITY_TABLE_NAME =
"APARTICLE_QUANTITY_TABLE_NAME";
}
```

(2)HBaseInvok

hbase的的回调操作类

代码位置：com.heima.hbase.entity.HBaseInvok

```
/**
 * Hbase 的回调类
 * 用于我们操作的时候就行回调
 */
public interface HBaseInvok {
    /**
     * 回调方法
     */
    public void invok();
}
```

(3)HBaseStorage

hbase 的存储对象 继承自StorageEntity

代码位置：com.heima.hbase.entity.HBaseStorage

```
/**
```

```
 * Hbase 存储对象 继承 StorageEntity
 * 用于存储各种对象
 */
@Setter
@Getter
public class HBaseStorage extends StorageEntity {
    /**
     * 主键key
     *
     */
    private String rowKey;
    /**
     * Hbase 的回调接口，用于将回调方法
     */
    private HBaseInvok hBaseInvok;

    /**
     * 获取类簇数组
     * @return
     */
    public List<String> getColumnFamily() {
        return
getDataList().stream().map(StorageData::getTargetClassName).collect(Collectors.t
oList());
    }

    /**
     * 进行回调
     */
    public void invok() {
        if (null != hBaseInvok) {
            hBaseInvok.invok();
        }
    }
}
```

(4)HBaseClent

hbase client操作的工具类

代码位置:com.heima.common.hbase.HBaseClent

```
/**
 * HBase相关的基本操作
 *
 * @since 1.0.0
 */
@Log4j2
public class HBaseClent {

    /**
     * 声明静态配置
     */
    private Configuration conf = null;
    private Connection connection = null;

    public HBaseClent(Configuration conf) {
```

```
        this.conf = conf;
        try {
            connection = ConnectionFactory.createConnection(conf);
        } catch (IOException e) {
            log.error("获取HBase连接失败");
        }
    }


    public boolean tableExists(String tableName) {
        if (StringUtils.isNotEmpty(tableName)) {
            Admin admin = null;
            try {
                admin = connection.getAdmin();
                return admin.tableExists(TableName.valueOf(tableName));
            } catch (Exception e) {
                log.debug("检查表是否存在异常", e);
            } finally {
                close(admin, null, null);
            }
        }
        return false;
    }


    /**
     * 创建表
     *
     * @param tableName    表名
     * @param columnFamily 列族名
     * @return void
     * @since 1.0.0
     */
    public boolean creatTable(String tableName, List<String> columnFamily) {
        Admin admin = null;
        try {
            admin = connection.getAdmin();

            List<ColumnFamilyDescriptor> familyDescriptors = new
ArrayList<ColumnFamilyDescriptor>(columnFamily.size());

            columnFamily.forEach(cf -> {

 familyDescriptors.add(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes(cf
)).build());
            });

            TableDescriptor tableDescriptor =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName))
                    .setColumnFamilies(familyDescriptors)
                    .build();

            if (admin.tableExists(TableName.valueOf(tableName))) {
                log.debug("table Exists!");
            } else {
                admin.createTable(tableDescriptor);
                log.debug("create table Success!");
            }
```

```
        } catch (IOException e) {
            log.error(MessageFormat.format("创建表{0}失败", tableName), e);
            return false;
        } finally {
            close(admin, null, null);
        }
        return true;
    }

    /**
     * 预分区创建表
     *
     * @param tableName    表名
     * @param columnFamily 列族名的集合
     * @param splitKeys    预分期region
     * @return 是否创建成功
     */
    public boolean createTableBySplitKeys(String tableName, List<String>
columnFamily, byte[][] splitKeys) {
        Admin admin = null;
        try {
            if (StringUtils.isBlank(tableName) || columnFamily == null
                    || columnFamily.size() == 0) {
                log.error("===Parameters tableName|columnFamily should not be
null,Please check!===");
                return false;
            }
            admin = connection.getAdmin();
            if (admin.tableExists(TableName.valueOf(tableName))) {
                return true;
            } else {
                List<ColumnFamilyDescriptor> familyDescriptors = new ArrayList<>
(columnFamily.size());

                columnFamily.forEach(cf -> {

 familyDescriptors.add(ColumnFamilyDescriptorBuilder.newBuilder(Bytes.toBytes(cf
)).build());
                });

                TableDescriptor tableDescriptor =
TableDescriptorBuilder.newBuilder(TableName.valueOf(tableName))
                        .setColumnFamilies(familyDescriptors)
                        .build();

                //指定splitkeys
                admin.createTable(tableDescriptor, splitKeys);
                log.info("===Create Table " + tableName
                        + " Success!columnFamily:" + columnFamily.toString()
                        + "===");
            }
        } catch (IOException e) {
            log.error("", e);
            return false;
        } finally {
            close(admin, null, null);
        }
```

```
        return true;
    }

    /**
     * 自定义获取分区splitKeys
     */
    public byte[][] getSplitKeys(String[] keys) {
        if (keys == null) {
            //默认为10个分区
            keys = new String[]{"1|", "2|", "3|", "4|",
                    "5|", "6|", "7|", "8|", "9|"};
        }
        byte[][] splitKeys = new byte[keys.length][];
        //升序排序
        TreeSet<byte[]> rows = new TreeSet<byte[]>(Bytes.BYTES_COMPARATOR);
        for (String key : keys) {
            rows.add(Bytes.toBytes(key));
        }

        Iterator<byte[]> rowKeyIter = rows.iterator();
        int i = 0;
        while (rowKeyIter.hasNext()) {
            byte[] tempRow = rowKeyIter.next();
            rowKeyIter.remove();
            splitKeys[i] = tempRow;
            i++;
        }
        return splitKeys;
    }

    /**
     * 按startKey和endKey，分区数获取分区
     */
    public static byte[][] getHexSplits(String startKey, String endKey, int
numRegions) {
        byte[][] splits = new byte[numRegions - 1][];
        BigInteger lowestKey = new BigInteger(startKey, 16);
        BigInteger highestKey = new BigInteger(endKey, 16);
        BigInteger range = highestKey.subtract(lowestKey);
        BigInteger regionIncrement =
range.divide(BigInteger.valueOf(numRegions));
        lowestKey = lowestKey.add(regionIncrement);
        for (int i = 0; i < numRegions - 1; i++) {
            BigInteger key =
lowestKey.add(regionIncrement.multiply(BigInteger.valueOf(i)));
            byte[] b = String.format("%016x", key).getBytes();
            splits[i] = b;
        }
        return splits;
    }

    /**
     * 获取table
     *
     * @param tableName 表名
     * @return Table
     * @throws IOException IOException
     */
```

```
    private Table getTable(String tableName) throws IOException {
        return connection.getTable(TableName.valueOf(tableName));
    }

    /**
     * 查询库中所有表的表名
     */
    public List<String> getAllTableNames() {
        List<String> result = new ArrayList<>();

        Admin admin = null;
        try {
            admin = connection.getAdmin();
            TableName[] tableNames = admin.listTableNames();

            for (TableName tableName : tableNames) {
                result.add(tableName.getNameAsString());
            }
        } catch (IOException e) {
            log.error("获取所有表的表名失败", e);
        } finally {
            close(admin, null, null);
        }
        return result;
    }


    /**
     * 查询库中所有表的表名
     */
    public List<String> getAllFamilyNames(String tableName) {
        ColumnFamilyDescriptor[] familyDescriptorList = null;
        List<String> familyNameList = new ArrayList<String>();
        try {
            Table table = getTable(tableName);
            familyDescriptorList = table.getDescriptor().getColumnFamilies();

        } catch (IOException e) {
            log.error("获取表的所有的列簇失败", e);
        } finally {
            //    close(admin, null, null);
        }
        if (null != familyDescriptorList && familyDescriptorList.length > 0) {
            familyNameList =
Arrays.stream(familyDescriptorList).map(ColumnFamilyDescriptor::getNameAsString)
.collect(Collectors.toList());
        }
        return familyNameList;
    }

    /**
     * 遍历查询指定表中的所有数据
     *
     * @param tableName 表名
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
```

```
    public Map<String, Map<String, String>> getResultScanner(String tableName) {
        Scan scan = new Scan();
        return this.queryData(tableName, scan);
    }

    /**
     * 根据startRowKey和stopRowKey遍历查询指定表中的所有数据
     *
     * @param tableName   表名
     * @param startRowKey 起始rowKey
     * @param stopRowKey  结束rowKey
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
    public Map<String, Map<String, String>> getResultScanner(String tableName,
String startRowKey, String stopRowKey) {
        Scan scan = new Scan();

        if (StringUtils.isNoneBlank(startRowKey) &&
StringUtils.isNoneBlank(stopRowKey)) {
            scan.withStartRow(Bytes.toBytes(startRowKey));
            scan.withStopRow(Bytes.toBytes(stopRowKey));
        }

        return this.queryData(tableName, scan);
    }

    /**
     * 通过行前缀过滤器查询数据
     *
     * @param tableName 表名
     * @param prefix    以prefix开始的行键
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
    public Map<String, Map<String, String>> getResultScannerPrefixFilter(String
tableName, String prefix) {
        Scan scan = new Scan();

        if (StringUtils.isNoneBlank(prefix)) {
            Filter filter = new PrefixFilter(Bytes.toBytes(prefix));
            scan.setFilter(filter);
        }

        return this.queryData(tableName, scan);
    }

    /**
     * 通过列前缀过滤器查询数据
     *
     * @param tableName 表名
     * @param prefix    以prefix开始的列名
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
```

```
    public Map<String, Map<String, String>>
getResultScannerColumnPrefixFilter(String tableName, String prefix) {
        Scan scan = new Scan();

        if (StringUtils.isNoneBlank(prefix)) {
            Filter filter = new ColumnPrefixFilter(Bytes.toBytes(prefix));
            scan.setFilter(filter);
        }

        return this.queryData(tableName, scan);
    }

    /**
     * 查询行键中包含特定字符的数据
     *
     * @param tableName 表名
     * @param keyword   包含指定关键词的行键
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
    public Map<String, Map<String, String>> getResultScannerRowFilter(String
tableName, String keyword) {
        Scan scan = new Scan();

        if (StringUtils.isNoneBlank(keyword)) {
            Filter filter = new RowFilter(CompareOperator.GREATER_OR_EQUAL, new
SubstringComparator(keyword));
            scan.setFilter(filter);
        }

        return this.queryData(tableName, scan);
    }

    /**
     * 查询列名中包含特定字符的数据
     *
     * @param tableName 表名
     * @param keyword   包含指定关键词的列名
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
    public Map<String, Map<String, String>>
getResultScannerQualifierFilter(String tableName, String keyword) {
        Scan scan = new Scan();

        if (StringUtils.isNoneBlank(keyword)) {
            Filter filter = new
QualifierFilter(CompareOperator.GREATER_OR_EQUAL, new
SubstringComparator(keyword));
            scan.setFilter(filter);
        }

        return this.queryData(tableName, scan);
    }
```

```
    /**
     * 通过表名以及过滤条件查询数据
     *
     * @param tableName 表名
     * @param scan      过滤条件
     * @return java.util.Map<java.lang.String, java.util.Map < java.lang.String,
java.lang.String>>
     * @since 1.0.0
     */
    private Map<String, Map<String, String>> queryData(String tableName, Scan
scan) {
        //<rowKey,对应的行数据>
        Map<String, Map<String, String>> result = new HashMap<>();

        ResultScanner rs = null;
        // 获取表
        Table table = null;
        try {
            table = getTable(tableName);
            rs = table.getScanner(scan);
            for (Result r : rs) {
                //每一行数据
                Map<String, String> columnMap = new HashMap<>();
                String rowKey = null;
                for (Cell cell : r.listCells()) {
                    if (rowKey == null) {
                        rowKey = Bytes.toString(cell.getRowArray(),
cell.getRowOffset(), cell.getRowLength());
                    }
                    columnMap.put(Bytes.toString(cell.getQualifierArray(),
cell.getQualifierOffset(), cell.getQualifierLength()),
Bytes.toString(cell.getValueArray(), cell.getValueOffset(),
cell.getValueLength()));
                }

                if (rowKey != null) {
                    result.put(rowKey, columnMap);
                }
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("遍历查询指定表中的所有数据失败,tableName:
{0}"
                    , tableName), e);
        } finally {
            close(null, rs, table);
        }

        return result;
    }

    /**
     * 根据tableName和rowKey精确查询一行的数据
     *
     * @param tableName 表名
     * @param rowKey    行键
     * @return java.util.Map<java.lang.String, java.lang.String> 返回一行的数据
     * @since 1.0.0
     */
```

```
    public Map<String, String> getRowData(String tableName, String rowKey) {
        //返回的键值对
        Map<String, String> result = new HashMap<>();

        Get get = new Get(Bytes.toBytes(rowKey));
        // 获取表
        Table table = null;
        try {
            table = getTable(tableName);
            Result hTableResult = table.get(get);
            if (hTableResult != null && !hTableResult.isEmpty()) {
                for (Cell cell : hTableResult.listCells()) {
//                System.out.println("family:" +
Bytes.toString(cell.getFamilyArray(), cell.getFamilyOffset(),
cell.getFamilyLength()));
//                System.out.println("qualifier:" +
Bytes.toString(cell.getQualifierArray(), cell.getQualifierOffset(),
cell.getQualifierLength()));
//                System.out.println("value:" +
Bytes.toString(cell.getValueArray(), cell.getValueOffset(),
cell.getValueLength()));
//                System.out.println("Timestamp:" + cell.getTimestamp());
//                System.out.println("-----------------------------------------
-");
                    result.put(Bytes.toString(cell.getQualifierArray(),
cell.getQualifierOffset(), cell.getQualifierLength()),
Bytes.toString(cell.getValueArray(), cell.getValueOffset(),
cell.getValueLength()));
                }
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("查询一行的数据失败,tableName:
{0},rowKey:{1}"
                    , tableName, rowKey), e);
        } finally {
            close(null, null, table);
        }

        return result;
    }


    public Result getHbaseResult(String tableName, String rowKey) {
        Get get = new Get(Bytes.toBytes(rowKey));
        // 获取表
        Table table = null;
        Result hTableResult = null;
        try {
            table = getTable(tableName);
            hTableResult = table.get(get);
        } catch (IOException e) {
            log.error(MessageFormat.format("查询一行的数据失败,tableName:
{0},rowKey:{1}"
                    , tableName, rowKey), e);
        } finally {
            close(null, null, table);
        }
        return hTableResult;
```

```
    }

    /**
     * 根据tableName和rowKey精确查询一行的数据
     *
     * @param tableName 表名
     * @param rowKey    行键
     * @return java.util.Map<java.lang.String, java.lang.String> 返回一行的数据
     * @since 1.0.0
     */
    public StorageData getStorageData(String tableName, String rowKey, String
familyName) {
        Result hTableResult = getHbaseResult(tableName, rowKey);
        return getStorageDataForfamilyName(hTableResult, familyName);
    }


    public List<StorageData> getStorageDataList(String tableName, String rowKey,
List<String> familyNameList) {
        List<StorageData> hBaseDataList = new ArrayList<StorageData>();
        Result hTableResult = getHbaseResult(tableName, rowKey);
        for (String familyName : familyNameList) {
            StorageData hBaseData = getStorageDataForfamilyName(hTableResult,
familyName);
            if (null != hBaseData) {
                hBaseDataList.add(hBaseData);
            }
        }
        return hBaseDataList;
    }

    public StorageData getStorageDataForfamilyName(Result hTableResult, String
familyName) {
        StorageData storageData = null;
        if (hTableResult != null && !hTableResult.isEmpty()) {
            Map<byte[], byte[]> familyMap =
hTableResult.getFamilyMap(Bytes.toBytes(familyName));
            if (null != familyMap && !familyMap.isEmpty()) {
                storageData = new StorageData();
                for (Map.Entry<byte[], byte[]> entry : familyMap.entrySet()) {
                    storageData.setTargetClassName(familyName);
                    storageData.addStorageEntry(Bytes.toString(entry.getKey()),
Bytes.toString(entry.getValue()));
                }
            }
        }
        return storageData;
    }


    /**
     * 根据tableName、rowKey、familyName、column查询指定单元格的数据
     *
     * @param tableName  表名
     * @param rowKey     rowKey
     * @param familyName 列族名
     * @param columnName 列名
     * @return java.lang.String
```

```
     * @since 1.0.0
     */
    public String getColumnValue(String tableName, String rowKey, String
familyName, String columnName) {
        String str = null;
        Get get = new Get(Bytes.toBytes(rowKey));
        // 获取表
        Table table = null;
        try {
            table = getTable(tableName);
            Result result = table.get(get);
            if (result != null && !result.isEmpty()) {
                Cell cell =
result.getColumnLatestCell(Bytes.toBytes(familyName),
Bytes.toBytes(columnName));
                if (cell != null) {
                    str = Bytes.toString(cell.getValueArray(),
cell.getValueOffset(), cell.getValueLength());
                }
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("查询指定单元格的数据失败,tableName:
{0},rowKey:{1},familyName:{2},columnName:{3}"
                    , tableName, rowKey, familyName, columnName), e);
        } finally {
            close(null, null, table);
        }

        return str;
    }

    /**
     * 根据tableName、rowKey、familyName、column查询指定单元格多个版本的数据
     *
     * @param tableName  表名
     * @param rowKey     rowKey
     * @param familyName 列族名
     * @param columnName 列名
     * @param versions   需要查询的版本数
     * @return java.util.List<java.lang.String>
     * @since 1.0.0
     */
    public List<String> getColumnValuesByVersion(String tableName, String
rowKey, String familyName, String columnName, int versions) {
        //返回数据
        List<String> result = new ArrayList<>(versions);

        // 获取表
        Table table = null;
        try {
            table = getTable(tableName);
            Get get = new Get(Bytes.toBytes(rowKey));
            get.addColumn(Bytes.toBytes(familyName), Bytes.toBytes(columnName));
            //读取多少个版本
            get.readVersions(versions);
            Result hTableResult = table.get(get);
            if (hTableResult != null && !hTableResult.isEmpty()) {
                for (Cell cell : hTableResult.listCells()) {
```

```
                    result.add(Bytes.toString(cell.getValueArray(),
cell.getValueOffset(), cell.getValueLength()));
                }
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("查询指定单元格多个版本的数据失
败,tableName:{0},rowKey:{1},familyName:{2},columnName:{3}"
                    , tableName, rowKey, familyName, columnName), e);
        } finally {
            close(null, null, table);
        }

        return result;
    }

    /**
     * 为表添加 or 更新数据
     *
     * @param tableName  表名
     * @param rowKey     rowKey
     * @param familyName 列族名
     * @param columns    列名数组
     * @param values     列值得数组
     * @since 1.0.0
     */
    public void putData(String tableName, String rowKey, String familyName,
String[] columns, String[] values) {
        // 获取表
        Table table = null;
        try {
            table = getTable(tableName);

            putData(table, rowKey, tableName, familyName, columns, values);
        } catch (Exception e) {
            log.error(MessageFormat.format("为表添加 or 更新数据失败,tableName:
{0},rowKey:{1},familyName:{2}"
                    , tableName, rowKey, familyName), e);
        } finally {
            close(null, null, table);
        }
    }

    /**
     * 为表添加 or 更新数据
     *
     * @param table      Table
     * @param rowKey     rowKey
     * @param tableName  表名
     * @param familyName 列族名
     * @param columns    列名数组
     * @param values     列值得数组
     * @since 1.0.0
     */
    private void putData(Table table, String rowKey, String tableName, String
familyName, String[] columns, String[] values) {
        try {
            //设置rowkey
            Put put = new Put(Bytes.toBytes(rowKey));
```

```
            if (columns != null && values != null && columns.length ==
values.length) {
                for (int i = 0; i < columns.length; i++) {
                    if (columns[i] != null && values[i] != null) {
                        put.addColumn(Bytes.toBytes(familyName),
Bytes.toBytes(columns[i]), Bytes.toBytes(values[i]));
                    } else {
                        throw new NullPointerException(MessageFormat.format("列名
和列数据都不能为空,column:{0},value:{1}"
                                , columns[i], values[i]));
                    }
                }
            }

            table.put(put);
            log.debug("putData add or update data Success,rowKey:" + rowKey);
            table.close();
        } catch (Exception e) {
            log.error(MessageFormat.format("为表添加 or 更新数据失败,tableName:
{0},rowKey:{1},familyName:{2}"
                    , tableName, rowKey, familyName), e);
        }
    }

    /**
     * 为表的某个单元格赋值
     *
     * @param tableName  表名
     * @param rowKey     rowKey
     * @param familyName 列族名
     * @param column1    列名
     * @param value1     列值
     * @since 1.0.0
     */
    public void setColumnValue(String tableName, String rowKey, String
familyName, String column1, String value1) {
        Table table = null;
        try {
            // 获取表
            table = getTable(tableName);
            // 设置rowKey
            Put put = new Put(Bytes.toBytes(rowKey));
            put.addColumn(Bytes.toBytes(familyName), Bytes.toBytes(column1),
Bytes.toBytes(value1));

            table.put(put);
            log.debug("add data Success!");
        } catch (IOException e) {
            log.error(MessageFormat.format("为表的某个单元格赋值失败,tableName:
{0},rowKey:{1},familyName:{2},column:{3}"
                    , tableName, rowKey, familyName, column1), e);
        } finally {
            close(null, null, table);
        }
    }

    /**
```

```
     * 删除指定的单元格
     *
     * @param tableName  表名
     * @param rowKey     rowKey
     * @param familyName 列族名
     * @param columnName 列名
     * @return boolean
     * @since 1.0.0
     */
    public boolean deleteColumn(String tableName, String rowKey, String
familyName, String columnName) {
        Table table = null;
        Admin admin = null;
        try {
            admin = connection.getAdmin();

            if (admin.tableExists(TableName.valueOf(tableName))) {
                // 获取表
                table = getTable(tableName);
                Delete delete = new Delete(Bytes.toBytes(rowKey));
                // 设置待删除的列
                delete.addColumns(Bytes.toBytes(familyName),
Bytes.toBytes(columnName));

                table.delete(delete);

 log.debug(MessageFormat.format("familyName({0}):columnName({1})is deleted!",
familyName, columnName));
            }

        } catch (IOException e) {
            log.error(MessageFormat.format("删除指定的列失败,tableName:{0},rowKey:
{1},familyName:{2},column:{3}"
                    , tableName, rowKey, familyName, columnName), e);
            return false;
        } finally {
            close(admin, null, table);
        }
        return true;
    }

    /**
     * 根据rowKey删除指定的行
     *
     * @param tableName 表名
     * @param rowKey    rowKey
     * @return boolean
     * @since 1.0.0
     */
    public boolean deleteRow(String tableName, String rowKey) {
        Table table = null;
        Admin admin = null;
        try {
            admin = connection.getAdmin();

            if (admin.tableExists(TableName.valueOf(tableName))) {
                // 获取表
                table = getTable(tableName);
```

```
                Delete delete = new Delete(Bytes.toBytes(rowKey));

                table.delete(delete);
                log.debug(MessageFormat.format("row({0}) is deleted!", rowKey));
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("删除指定的行失败,tableName:{0},rowKey:
{1}"
                    , tableName, rowKey), e);
            return false;
        } finally {
            close(admin, null, table);
        }
        return true;
    }

    /**
     * 根据columnFamily删除指定的列族
     *
     * @param tableName    表名
     * @param columnFamily 列族
     * @return boolean
     * @since 1.0.0
     */
    public boolean deleteColumnFamily(String tableName, String columnFamily) {
        Admin admin = null;
        try {
            admin = connection.getAdmin();

            if (admin.tableExists(TableName.valueOf(tableName))) {
                admin.deleteColumnFamily(TableName.valueOf(tableName),
Bytes.toBytes(columnFamily));
                log.debug(MessageFormat.format("familyName({0}) is deleted!",
columnFamily));
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("删除指定的列族失败,tableName:
{0},columnFamily:{1}"
                    , tableName, columnFamily), e);
            return false;
        } finally {
            close(admin, null, null);
        }
        return true;
    }

    /**
     * 删除表
     *
     * @param tableName 表名
     * @since 1.0.0
     */
    public boolean deleteTable(String tableName) {
        Admin admin = null;
        try {
            admin = connection.getAdmin();

            if (admin.tableExists(TableName.valueOf(tableName))) {
```

```
                admin.disableTable(TableName.valueOf(tableName));
                admin.deleteTable(TableName.valueOf(tableName));
                log.debug(tableName + "is deleted!");
            }
        } catch (IOException e) {
            log.error(MessageFormat.format("删除指定的表失败,tableName:{0}"
                    , tableName), e);
            return false;
        } finally {
            close(admin, null, null);
        }
        return true;
    }

    /**
     * 关闭流
     */
    private void close(Admin admin, ResultScanner rs, Table table) {
        if (admin != null) {
            try {
                admin.close();
            } catch (IOException e) {
                log.error("关闭Admin失败", e);
            }
        }

        if (rs != null) {
            rs.close();
        }

        if (table != null) {
            try {
                table.close();
            } catch (IOException e) {
                log.error("关闭Table失败", e);
            }
        }
    }
}
```

(5)HBaseConfig

用于将HbaseClient对象的相关配置

代码位置:com.heima.common.hbase.HBaseConfig

```
/**
 * HBase 配置类，读取 hbase.properties 配置文件
 */
@Setter
@Getter
@Configuration
@PropertySource("classpath:hbase.properties")
public class HBaseConfig {

    /**
     * hBase 注册地址
```

```
     */
    @Value("${hbase.zookeeper.quorum}")
    private String zookip_quorum;
    /**
     * 超时时间
     */
    @Value("${hbase.client.keyvalue.maxsize}")
    private String maxsize;

    /**
     * 创建HBaseClien
     *
     * @return
     */
    @Bean
    public HBaseClent getHBaseClient() {
        org.apache.hadoop.conf.Configuration hBaseConfiguration =
getHbaseConfiguration();
        return new HBaseClent(hBaseConfiguration);
    }

    /**
     * 获取HbaseConfiguration 对象
     * @return
     */
    private org.apache.hadoop.conf.Configuration getHbaseConfiguration() {
        org.apache.hadoop.conf.Configuration hBaseConfiguration = new
org.apache.hadoop.conf.Configuration();
        hBaseConfiguration.set("hbase.zookeeper.quorum", zookip_quorum);
        hBaseConfiguration.set("hbase.client.keyvalue.maxsize", maxsize);
        return hBaseConfiguration;
    }
}
```

(6)HBaseStorageClient

Hbase 存储客户端工具类 是对HbaseClient工具类的封装

这个类是自己封装的存储客户端

该类位于heima-leadnews-common 包下的com.heima.hbase.HBaseStorageClient

其中用到了**HBaseClent** 客户端工具，他是一个操作工具类，不需要我们具体的写拿过来用就可以

代码位置:com.heima.common.hbase.HBaseStorageClient

```
/**
 * HBase存储客户端
 */
@Component
@Log4j2
public class HBaseStorageClient {

    /**
     * 注入的HBaseClent 工具类
     */
    @Autowired
    private HBaseClent hBaseClent;
```

```
    /**
     * 添加一个存储列表到Hbase
     *
     * @param tableName        表明
     * @param hBaseStorageList 存储列表
     */
    public void addHBaseStorage(final String tableName, List<HBaseStorage>
hBaseStorageList) {
        if (null != hBaseStorageList && !hBaseStorageList.isEmpty()) {
            hBaseStorageList.stream().forEach(hBaseStorage -> {
                addHBaseStorage(tableName, hBaseStorage);
            });
        }
    }

    /**
     * 添加一个存储到Hbase
     *
     * @param tableName    表明
     * @param hBaseStorage 存储
     */
    public void addHBaseStorage(String tableName, HBaseStorage hBaseStorage) {
        if (null != hBaseStorage && StringUtils.isNotEmpty(tableName)) {
            hBaseClent.creatTable(tableName, hBaseStorage.getColumnFamily());
            String rowKey = hBaseStorage.getRowKey();
            List<StorageData> storageDataList = hBaseStorage.getDataList();
            boolean result = addStorageData(tableName, rowKey, storageDataList);
            if (result) {
                hBaseStorage.invok();
            }
        }

    }

    /**
     * 添加 数据到Hbase
     *
     * @param tableName       表明
     * @param rowKey          主键
     * @param storageDataList 存储数据集合
     * @return
     */
    public boolean addStorageData(String tableName, String rowKey,
List<StorageData> storageDataList) {
        long currentTime = System.currentTimeMillis();
        log.info("开始添加StorageData到Hbase,tableName:{},rowKey:{}", tableName,
rowKey);
        if (null != storageDataList && !storageDataList.isEmpty()) {
            storageDataList.forEach(hBaseData -> {
                String columnFamliyName = hBaseData.getTargetClassName();
                String[] columnArray = hBaseData.getColumns();
                String[] valueArray = hBaseData.getValues();
                if (null != columnArray && null != valueArray) {
                    hBaseClent.putData(tableName, rowKey, columnFamliyName,
columnArray, valueArray);
                }
            });
        }
```

```
        log.info("添加StorageData到Hbase完成,tableName:{},rowKey:{},duration:{}",
tableName, rowKey, System.currentTimeMillis() - currentTime);
        return true;
    }

    /**
     * 根据表明以及rowKey 获取一个对象
     *
     * @param tableName 表明
     * @param rowKey    主键
     * @param tClass    需要获取的对象类型
     * @param <T>       泛型T
     * @return 返回要返回的数
     */
    public <T> T getStorageDataEntity(String tableName, String rowKey, Class<T>
tClass) {
        T tValue = null;
        if (StringUtils.isNotEmpty(tableName)) {
            StorageData hBaseData = hBaseClent.getStorageData(tableName, rowKey,
tClass.getName());
            if (null != hBaseData) {
                tValue = (T) hBaseData.getObjectValue();
            }
        }
        return tValue;
    }

    /**
     * 根据 类型列表 , 表明 rowkey 返回一个数据类型的列表
     *
     * @param tableName 表明
     * @param rowKey    rowKey
     * @param typeList  类型列表
     * @return 返回的对象列表
     */
    public List<Object> getStorageDataEntityList(String tableName, String
rowKey, List<Class> typeList) {
        List<Object> entityList = new ArrayList<Object>();
        List<String> strTypeList = typeList.stream().map(x ->
x.getName()).collect(Collectors.toList());
        List<StorageData> storageDataList =
hBaseClent.getStorageDataList(tableName, rowKey, strTypeList);
        for (StorageData storageData : storageDataList) {
            entityList.add(storageData.getObjectValue());
        }
        return entityList;
    }

    /**
     * 获取HBaseClent 客户端
     *
     * @return
     */
    public HBaseClent gethBaseClent() {
        return hBaseClent;
    }
}
```

4.1.4 测试代码

```
@SpringBootTest
@RunWith(SpringRunner.class)
public class HbaseTest {

    @Autowired
    private HBaseClent hBaseClent;

    @Test
    public void testCreateTable(){

        List<String> columnFamily = new ArrayList<>();
        columnFamily.add("test_cloumn_family1");
        columnFamily.add("test_cloumn_family2");
        boolean ret = hBaseClent.creatTable("hbase_test_table_name",
columnFamily);
    }

    @Test
    public void testDelTable(){
        hBaseClent.deleteTable("hbase_test_table_name");
    }

    @Test
    public void testSaveData(){
        String []columns ={"name","age"};
        String [] values = {"zhangsan","28"};

 hBaseClent.putData("hbase_test_table_name","test_row_key_001","test_cloumn_fami
ly1",columns,values);
    }

    @Test
    public void testFindByRowKey(){
        Result hbaseResult = hBaseClent.getHbaseResult("hbase_test_table_name",
"test_row_key_001");
        System.out.println(hbaseResult);
    }
}
```

4.2 MongoDB操作工具类

mongoDB是一个文档型数据库，也需要存储多个不同的对象，我们也用到了HBASE中用到的StorageEntity 存储结构，我们下面会讲 我们用到了Spring MongoTemplate 来操作数据库 介绍以下我们的实体

(1)MongoConstant

mongoDB操作的常量定义了操作mongodb的表名称

代码位置：com.heima.common.mongo.constants.MongoConstant

```
public class MongoConstant {
    public static final String APARTICLE_MIGRATION_TABLE =
"APARTICLE_MIGRATION_TABLE";
}
```

(2)MongoStorageEntity

MongoStorageEntity 是我们存储MongoDB数据的存储结构主要是基于StorageEntity 结构来的

mongoDB操作的实体类继承了StorageEntity 制定了 表明以及实体类型

代码位置：com.heima.common.mongo.entity.MongoStorageEntity

```
/**
 * mongoDB 存储实体
 * Document 是指表明是什么
 */
@Document(collection = "mongo_storage_data")
@Setter
@Getter
public class MongoStorageEntity extends StorageEntity {
    /**
     * 主键的Key
     *
     * @Id 标明该字段是主键
     */
    @Id
    private String rowKey;

}
```

(3)MongoDBconfigure

对mongdb操作的配置类

代码位置：com.heima.common.mongo.MongoDBconfigure

```
@Configuration
@PropertySource("classpath:mongo.properties")
public class MongoDBconfigure {

    @Value("${mongo.host}")
    private String host;
    @Value("${mongo.port}")
    private int port;
    @Value("${mongo.dbname}")
    private String dbName;

    @Bean
    public MongoTemplate getMongoTemplate() {
        return new MongoTemplate(getSimpleMongoDbFactory());
    }

    public SimpleMongoDbFactory getSimpleMongoDbFactory() {
        return new SimpleMongoDbFactory(new MongoClient(host, port), dbName);
    }
```

```
}
```

(4)测试代码

```
@SpringBootTest(classes = MigrationApplication.class)
@RunWith(SpringJUnit4ClassRunner.class)
public class MongoTest {

    @Autowired
    private MongoTemplate mongotemplate;


    @Autowired
    private HBaseStorageClient hBaseStorageClient;


    @Test
    public void test() {
        Class<?>[] classes = new Class<?>[]{ApArticle.class,
ApArticleContent.class, ApAuthor.class};
        //List<Object> entityList =
hBaseStorageClient.getHbaseDataEntityList(HBaseConstants.APARTICLE_QUANTITY_TABL
E_NAME, "1", Arrays.asList(classes));
        List<String> strList = Arrays.asList(classes).stream().map(x ->
x.getName()).collect(Collectors.toList());
        List<StorageData> storageDataList =
hBaseStorageClient.gethBaseClent().getStorageDataList(HBaseConstants.APARTICLE_Q
UANTITY_TABLE_NAME, "1", strList);
        MongoStorageEntity mongoStorageEntity = new MongoStorageEntity();
        mongoStorageEntity.setDataList(storageDataList);
        mongoStorageEntity.setRowKey("1");
        MongoStorageEntity tmp = mongotemplate.findById("1",
MongoStorageEntity.class);
        if (null != tmp) {
            mongotemplate.remove(tmp);
        }
        MongoStorageEntity tq = mongotemplate.insert(mongoStorageEntity);
        System.out.println(tq);

    }

    @Test
    public void test1() {
        MongoStorageEntity mongoStorageEntity = mongotemplate.findById("1",
MongoStorageEntity.class);
        if (null != mongoStorageEntity && null !=
mongoStorageEntity.getDataList()) {
            mongoStorageEntity.getDataList().forEach(x -> {
                System.out.println(x.getObjectValue());
            });
        }
    }
}
```

7 业务层代码

7.1 Habse操作实体类

(1)ArticleCallBack

Hbase相关回调操作的工具类

代码位置：com.heima.migration.entity.ArticleCallBack

```
public interface ArticleCallBack {
    public void callBack(ApArticle apArticle);
}
```

(2)ArticleHBaseInvok

Hbase 对回调对象的封装，以及对回调的invok执行对象

代码位置：com.heima.migration.entity.ArticleHBaseInvok

```
/**
 * 回调对象
 */
@Setter
@Getter
public class ArticleHBaseInvok implements HBaseInvok {

    /**
     * 回调需要传输的对象
     */
    private ApArticle apArticle;
    /**
     * 回调需要对应的回调接口
     */
    private ArticleCallBack articleCallBack;

    public ArticleHBaseInvok(ApArticle apArticle, ArticleCallBack
articleCallBack) {
        this.apArticle = apArticle;
        this.articleCallBack = articleCallBack;
    }

    /**
     * 执行回调方法
     */
    @Override
    public void invok() {
        if (null != apArticle && null != articleCallBack) {
            articleCallBack.callBack(apArticle);
        }
    }
}
```

(3)ArticleQuantity

对整个需要存储的对象的封装

代码位置：com.heima.migration.entity.ArticleQuantity

```
/**
 * Article 封装数据的工具类
 */
@Setter
@Getter
public class ArticleQuantity {

    /**
     * 文章关系数据实体
     */
    private ApArticle apArticle;

    /**
     * 文章配置实体
     */
    private ApArticleConfig apArticleConfig;
    /**
     * 文章内容实体
     */
    private ApArticleContent apArticleContent;
    /**
     * 文章作者实体
     */
    private ApAuthor apAuthor;
    /**
     * 回调接口
     */
    private HBaseInvok hBaseInvok;


    public Integer getApArticleId() {
        if (null != apArticle) {
            return apArticle.getId();
        }
        return null;
    }

    /**
     * 将ArticleQuantity 对象转换为HBaseStorage对象
     *
     * @return
     */
    public HBaseStorage getHbaseStorage() {
        HBaseStorage hbaseStorage = new HBaseStorage();
        hbaseStorage.setRowKey(String.valueOf(apArticle.getId()));
        hbaseStorage.setHBaseInvok(hBaseInvok);
        StorageData apArticleData = StorageData.getStorageData(apArticle);
        if (null != apArticleData) {
            hbaseStorage.addStorageData(apArticleData);
        }

        StorageData apArticleConfigData =
StorageData.getStorageData(apArticleConfig);
```

```
        if (null != apArticleConfigData) {
            hbaseStorage.addStorageData(apArticleConfigData);
        }

        StorageData apArticleContentData =
StorageData.getStorageData(apArticleContent);
        if (null != apArticleContentData) {
            hbaseStorage.addStorageData(apArticleContentData);
        }

        StorageData apAuthorData = StorageData.getStorageData(apAuthor);
        if (null != apAuthorData) {
            hbaseStorage.addStorageData(apAuthorData);
        }
        return hbaseStorage;
    }


    /**
     * 获取 StorageData 列表
     *
     * @return
     */
    public List<StorageData> getStorageDataList() {
        List<StorageData> storageDataList = new ArrayList<StorageData>();
        StorageData apArticleStorageData =
StorageData.getStorageData(apArticle);
        if (null != apArticleStorageData) {
            storageDataList.add(apArticleStorageData);
        }

        StorageData apArticleContentStorageData =
StorageData.getStorageData(apArticleContent);
        if (null != apArticleContentStorageData) {
            storageDataList.add(apArticleContentStorageData);
        }


        StorageData apArticleConfigStorageData =
StorageData.getStorageData(apArticleConfig);
        if (null != apArticleConfigStorageData) {
            storageDataList.add(apArticleConfigStorageData);
        }

        StorageData apAuthorStorageData = StorageData.getStorageData(apAuthor);
        if (null != apAuthorStorageData) {
            storageDataList.add(apAuthorStorageData);
        }
        return storageDataList;
    }


    public ApHotArticles getApHotArticles() {
        ApHotArticles apHotArticles = null;
        if (null != apArticle) {
            apHotArticles = new ApHotArticles();
            apHotArticles.setArticleId(apArticle.getId());
            apHotArticles.setReleaseDate(apArticle.getPublishTime());
```

```
            apHotArticles.setScore(1);
            // apHotArticles.setTagId();
            apHotArticles.setTagName(apArticle.getLabels());
            apHotArticles.setCreatedTime(new Date());
        }
        return apHotArticles;

    }

}
```

7.2 文章配置接口

7.2.1 mapper

ApArticleConfigMapper中新增方法

```
 List<ApArticleConfig> selectByArticleIds(List<String> articleIds);
```

ApArticleConfigMapper.xml

```
<select id="selectByArticleIds" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    from ap_article_config
    where article_id in
    <foreach item="item" index="index" collection="list" open="(" separator="," close=")">
        #{item}
    </foreach>
</select>
```

7.2.2 service

对文章配置操作的service

接口位置：com.heima.migration.service.ApArticleConfigService

```
public interface ApArticleConfigService {

    List<ApArticleConfig> queryByArticleIds(List<String> ids);

    ApArticleConfig getByArticleId(Integer id);
}
```

ApArticleConfigServiceImpl

是对ApArticleConfig的操作

代码位置:com.heima.migration.service.impl.ApArticleConfigServiceImpl

```
@Service
public class ApArticleConfigServiceImpl implements ApArticleConfigService {
```

```
    @Autowired
    private ApArticleConfigMapper apArticleConfigMapper;

    @Override
    public List<ApArticleConfig> queryByArticleIds(List<String> ids) {
        return apArticleConfigMapper.selectByArticleIds(ids);
    }

    @Override
    public ApArticleConfig getByArticleId(Integer id) {
        return apArticleConfigMapper.selectByArticleId(id);
    }
}
```

7.3 文章内容接口

7.3.1 mapper定义

ApArticleContentMapper新增方法

```
 List<ApArticleContent> selectByArticleIds(List<String> articleIds);
```

ApArticleContentMapper.xml

```
<select id="selectByArticleIds" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    ,
    <include refid="Blob_Column_List"/>
    from ap_article_content
    where article_id IN
    <foreach collection="list" item="id" index="index" open="(" close=")"
separator=",">
        #{id}
    </foreach>
</select>
```

7.3.2 service

对文章内容操作的Service

接口位置：com.heima.migration.service.ApArticleContenService

```
public interface ApArticleContenService {

    List<ApArticleContent> queryByArticleIds(List<String> ids);


    ApArticleContent getByArticleIds(Integer id);
}
```

ApArticleContenServiceImpl

对ApArticleConten相关的操作

代码位置：com.heima.migration.service.impl.ApArticleContenServiceImpl

```
@Service
public class ApArticleContenServiceImpl implements ApArticleContenService {

    @Autowired
    private ApArticleContentMapper apArticleContentMapper;

    @Override
    public List<ApArticleContent> queryByArticleIds(List<String> ids) {
        return apArticleContentMapper.selectByArticleIds(ids);
    }

    @Override
    public ApArticleContent getByArticleIds(Integer id) {
        return apArticleContentMapper.selectByArticleId(id);
    }
}
```

7.4 文章接口

7.4.1 mapper定义

ApArticleMapper新增方法

```
/**
     * 查询
     *
     * @param apArticle
     * @return
     */
List<ApArticle> selectList(ApArticle apArticle);
/**
     * 更新
     * @param apArticle
     */
void updateSyncStatus(ApArticle apArticle);
```

ApArticleMapper.xml

```
<sql id="Base_Column_Where">
    <where>
        <if test="title!=null and title!=''">
            and title = #{title}
        </if>
        <if test="authorId!=null and authorId!=''">
            and author_id = #{authorId}
        </if>
        <if test="authorName!=null and authorName!=''">
            and author_name = #{authorName}
        </if>
        <if test="channelId!=null and channelId!=''">
```

```
            and channel_id = #{channelId}
        </if>
        <if test="channelName!=null and channelName!=''">
            and channel_name = #{channelName}
        </if>
        <if test="layout!=null and layout!=''">
            and layout = #{layout}
        </if>
        <if test="flag!=null and flag!=''">
            and flag = #{flag}
        </if>
        <if test="views!=null and views!=''">
            and views = #{views}
        </if>
        <if test="syncStatus!=null">
            and sync_status = #{syncStatus}
        </if>
    </where>
</sql>
<select id="selectList" resultMap="resultMap">
    select
    <include refid="Base_Column_List"/>
    from ap_article
    <include refid="Base_Column_Where"/>
</select>
<update id="updateSyncStatus">
    UPDATE ap_article SET sync_status = #{syncStatus} WHERE id=#{id}
</update>
```

7.4.2 service

对ApArticle操作的Service

接口位置：com.heima.migration.service.ApArticleService

```
public interface ApArticleService {

    public ApArticle getById(Long id);

    /**
     * 获取未同步的数据
     *
     * @return
     */
    public List<ApArticle> getUnsyncApArticleList();

    /**
     * 更新同步状态
     *
     * @param apArticle
     */
    void updateSyncStatus(ApArticle apArticle);
}
```

ApArticleServiceImpl

对ApArticleService相关的操作

代码位置：com.heima.migration.service.impl.ApArticleServiceImpl

```
@Log4j2
@Service
public class ApArticleServiceImpl implements ApArticleService {

    @Autowired
    private ApArticleMapper apArticleMapper;


    public ApArticle getById(Long id) {
        return apArticleMapper.selectById(id);
    }

    /**
     * 获取未同步的数据
     *
     * @return
     */
    public List<ApArticle> getUnsyncApArticleList() {
        ApArticle apArticleQuery = new ApArticle();
        apArticleQuery.setSyncStatus(false);
        return apArticleMapper.selectList(apArticleQuery);
    }

    /**
     * 更新数据同步状态
     *
     * @param apArticle
     */

    public void updateSyncStatus(ApArticle apArticle) {
        log.info("开始更新数据同步状态，apArticle：{}", apArticle);
        if (null != apArticle) {
            apArticle.setSyncStatus(true);
            apArticleMapper.updateSyncStatus(apArticle);
        }
    }

}
```

7.5 文章作者接口

7.5.1 mapper定义

ApAuthorMapper

```
 List<ApAuthor> selectByIds(List<Integer> ids);
```

ApAuthorMapper.xml

```
<select id="selectByIds" resultMap="BaseResultMap">
    select * from ap_author
    where id in
    <foreach item="item" index="index" collection="list" open="(" separator="," close=")">
        #{item}
    </foreach>
</select>
```

7.5.2 service

对ApAuthor操作的Service

接口位置:com.heima.migration.service.ApAuthorService

```
public interface ApAuthorService {

    List<ApAuthor> queryByIds(List<Integer> ids);

    ApAuthor getById(Long id);
}
```

ApAuthorServiceImpl

对ApAuthor相关的操作

代码位置:com.heima.migration.service.impl.ApAuthorServiceImpl

```
@Service
public class ApAuthorServiceImpl implements ApAuthorService {

    @Autowired
    private ApAuthorMapper apAuthorMapper;

    @Override
    public List<ApAuthor> queryByIds(List<Integer> ids) {
        return apAuthorMapper.selectByIds(ids);
    }

    @Override
    public ApAuthor getById(Long id) {
        if (null != id) {
            return apAuthorMapper.selectById(id.intValue());
        }
        return null;

    }
}
```

7.6 综合迁移接口

ArticleQuantityService

操作ArticleQuantity对象的Service ArticleQuantity对象封装了文章相关的数据

接口位置：com.heima.migration.service.ArticleQuantityService

```
public interface ArticleQuantityService {

    /**
     * 获取ArticleQuantity列表
     * @return
     */
    public List<ArticleQuantity> getArticleQuantityList();

    /**
     * 根据ArticleId获取ArticleQuantity
     * @param id
     * @return
     */
    public ArticleQuantity getArticleQuantityByArticleId(Long id);

    /**
     * 根据ByArticleId从Hbase中获取ArticleQuantity
     * @param id
     * @return
     */
    public ArticleQuantity getArticleQuantityByArticleIdForHbase(Long id);

    /**
     * 数据库到Hbase的同步
     */
    public void dbToHbase();

    /**
     * 根据articleId 将数据库的数据同步到Hbase
     * @param articleId
     */
    public void dbToHbase(Integer articleId);


}
```

ArticleQuantityServiceImpl

对ArticleQuantity的相关操作

代码位置：com.heima.migration.service.impl.ArticleQuantityServiceImpl

```
/**
 * 查询未同步的数据，并封装成ArticleQuantity 对象
 */
@Service
@Log4j2
public class ArticleQuantityServiceImpl implements ArticleQuantityService {

    @Autowired
    private ApArticleContenService apArticleContenService;
    @Autowired
    private ApArticleConfigService apArticleConfigService;
    @Autowired
```

```
    private ApAuthorService apAuthorService;

    @Autowired
    private HBaseStorageClient hBaseStorageClient;


    @Autowired
    private ApArticleService apArticleService;

    /**
     * 查询位同步数据的列表
     *
     * @return
     */
    public List<ArticleQuantity> getArticleQuantityList() {
        log.info("生成ArticleQuantity列表");
        //查询未同步的庶数据
        List<ApArticle> apArticleList =
apArticleService.getUnsyncApArticleList();
        if (apArticleList.isEmpty()) {
            return null;
        }
        //获取ArticleId 的list
        List<String> apArticleIdList = apArticleList.stream().map(apArticle ->
String.valueOf(apArticle.getId())).collect(Collectors.toList());
        //获取AuthorId 的 list
        List<Integer> apAuthorIdList = apArticleList.stream().map(apAuthor ->
apAuthor.getAuthorId() == null ? null :
apAuthor.getAuthorId().intValue()).filter(x -> x !=
null).collect(Collectors.toList());
        //根据apArticleIdList 批量查询出内容列表
        List<ApArticleContent> apArticleContentList =
apArticleContenService.queryByArticleIds(apArticleIdList);
        //根据apArticleIdList 批量查询出配置列表
        List<ApArticleConfig> apArticleConfigList =
apArticleConfigService.queryByArticleIds(apArticleIdList);
        //根据apAuthorIdList 批量查询出作者列
        List<ApAuthor> apAuthorList =
apAuthorService.queryByIds(apAuthorIdList);

        //将不同的对象转换为 ArticleQuantity 对象
        List<ArticleQuantity> articleQuantityList =
apArticleList.stream().map(apArticle -> {
            return new ArticleQuantity() {{
                //设置apArticle 对象
                setApArticle(apArticle);
                // 根据apArticle.getId() 过滤出符合要求的 ApArticleContent 对象
                List<ApArticleContent> apArticleContents =
apArticleContentList.stream().filter(x ->
x.getArticleId().equals(apArticle.getId())).collect(Collectors.toList());
                if (null != apArticleContents && !apArticleContents.isEmpty()) {
                    setApArticleContent(apArticleContents.get(0));
                }
                // 根据 apArticle.getId 过滤出 ApArticleConfig 对象
                List<ApArticleConfig> apArticleConfigs =
apArticleConfigList.stream().filter(x ->
x.getArticleId().equals(apArticle.getId())).collect(Collectors.toList());
                if (null != apArticleConfigs && !apArticleConfigs.isEmpty()) {
```

```
                    setApArticleConfig(apArticleConfigs.get(0));
                }
                // 根据 apArticle.getAuthorId().intValue() 过滤出 ApAuthor 对象
                List<ApAuthor> apAuthors = apAuthorList.stream().filter(x ->
x.getId().equals(apArticle.getAuthorId().intValue())).collect(Collectors.toList(
));
                if (null != apAuthors && !apAuthors.isEmpty()) {
                    setApAuthor(apAuthors.get(0));
                }
                //设置回调方法 用户方法的回调 用于修改同步状态 插入Hbase 成功后同步状态改为
已同步
                setHBaseInvok(new ArticleHBaseInvok(apArticle, (x) ->
apArticleService.updateSyncStatus(x)));
            }};
        }).collect(Collectors.toList());
        if (null != articleQuantityList && !articleQuantityList.isEmpty()) {
            log.info("生成ArticleQuantity列表完成，size:{}",
articleQuantityList.size());
        } else {
            log.info("生成ArticleQuantity列表完成，size:{}", 0);
        }

        return articleQuantityList;
    }


    public ArticleQuantity getArticleQuantityByArticleId(Long id) {
        if (null == id) {
            return null;
        }
        ArticleQuantity articleQuantity = null;
        ApArticle apArticle = apArticleService.getById(id);
        if (null != apArticle) {
            articleQuantity = new ArticleQuantity();
            articleQuantity.setApArticle(apArticle);
            ApArticleContent apArticleContent =
apArticleContenService.getByArticleIds(id.intValue());
            articleQuantity.setApArticleContent(apArticleContent);
            ApArticleConfig apArticleConfig =
apArticleConfigService.getByArticleId(id.intValue());
            articleQuantity.setApArticleConfig(apArticleConfig);
            ApAuthor apAuthor =
apAuthorService.getById(apArticle.getAuthorId());
            articleQuantity.setApAuthor(apAuthor);
        }
        return articleQuantity;
    }


    public ArticleQuantity getArticleQuantityByArticleIdForHbase(Long id) {
        if (null == id) {
            return null;
        }
        ArticleQuantity articleQuantity = null;
        List<Class> typeList = Arrays.asList(ApArticle.class,
ApArticleContent.class, ApArticleConfig.class, ApAuthor.class);
```

```
        List<Object> objectList =
hBaseStorageClient.getStorageDataEntityList(HBaseConstants.APARTICLE_QUANTITY_TA
BLE_NAME, DataConvertUtils.toString(id), typeList);
        if (null != objectList && !objectList.isEmpty()) {
            articleQuantity = new ArticleQuantity();
            for (Object value : objectList) {
                if (value instanceof ApArticle) {
                    articleQuantity.setApArticle((ApArticle) value);
                } else if (value instanceof ApArticleContent) {
                    articleQuantity.setApArticleContent((ApArticleContent)
value);
                } else if (value instanceof ApArticleConfig) {
                    articleQuantity.setApArticleConfig((ApArticleConfig) value);
                } else if (value instanceof ApAuthor) {
                    articleQuantity.setApAuthor((ApAuthor) value);
                }
            }
        }
        return articleQuantity;
    }

    /**
     * 数据库到Hbase同步
     */
    public void dbToHbase() {
        long cutrrentTime = System.currentTimeMillis();
        List<ArticleQuantity> articleQuantitList = getArticleQuantityList();
        if (null != articleQuantitList && !articleQuantitList.isEmpty()) {
            log.info("开始进行定时数据库到HBASE同步，筛选出未同步数据量：{}",
articleQuantitList.size());
            if (null != articleQuantitList && !articleQuantitList.isEmpty()) {
                List<HBaseStorage> hbaseStorageList =
articleQuantitList.stream().map(ArticleQuantity::getHbaseStorage).collect(Collec
tors.toList());

 hBaseStorageClient.addHBaseStorage(HBaseConstants.APARTICLE_QUANTITY_TABLE_NAME
, hbaseStorageList);
            }
        } else {
            log.info("定时数据库到HBASE同步为筛选出数据");
        }

        log.info("定时数据库到HBASE同步结束，耗时:{}", System.currentTimeMillis() -
cutrrentTime);
    }

    @Override
    public void dbToHbase(Integer articleId) {
        long cutrrentTime = System.currentTimeMillis();
        log.info("开始进行异步数据库到HBASE同步，articleId：{}", articleId);
        if (null != articleId) {
            ArticleQuantity articleQuantity =
getArticleQuantityByArticleId(articleId.longValue());
            if (null != articleQuantity) {
                HBaseStorage hBaseStorage = articleQuantity.getHbaseStorage();

 hBaseStorageClient.addHBaseStorage(HBaseConstants.APARTICLE_QUANTITY_TABLE_NAME
, hBaseStorage);
```

```
            }
        }
        log.info("异步数据库到HBASE同步结束，articleId: {}，耗时:{}", articleId,
System.currentTimeMillis() - cutrrentTime);
    }
}
```

7.7 热点文章接口

ApHotArticleService

对ApHotArticle操作Service

接口位置：com.heima.migration.service.ApHotArticleService

```
public interface ApHotArticleService {

    List<ApHotArticles> selectList(ApHotArticles apHotArticlesQuery);

    void insert(ApHotArticles apHotArticles);

    /**
     * 热数据 Hbase 同步
     *
     * @param apArticleId
     */
    public void hotApArticleSync(Integer apArticleId);


    void deleteById(Integer id);

    /**
     * 查询过期的数据
     *
     * @return
     */
    public List<ApHotArticles> selectExpireMonth();


    void deleteHotData(ApHotArticles apHotArticle);
}
```

ApHotArticleServiceImpl

对ApHotArticle的相关操作

代码位置：com.heima.migration.service.impl.ApHotArticleServiceImpl

```
/**
 * 热点数据操作Service 类
 */
@Service
@Log4j2
public class ApHotArticleServiceImpl implements ApHotArticleService {

    @Autowired
```

```
    private ApHotArticlesMapper apHotArticlesMapper;

    @Autowired
    private MongoTemplate mongoTemplate;

    @Autowired
    private ArticleQuantityService articleQuantityService;

    @Autowired
    private HBaseStorageClient hBaseStorageClient;

    @Override
    public List<ApHotArticles> selectList(ApHotArticles apHotArticlesQuery) {
        return apHotArticlesMapper.selectList(apHotArticlesQuery);
    }

    /**
     * 根据ID删除
     *
     * @param id
     */
    @Override
    public void deleteById(Integer id) {
        log.info("删除热数据，apArticleId：{}", id);
        apHotArticlesMapper.deleteById(id);
    }

    /**
     * 查询一个月之前的数据
     *
     * @return
     */
    @Override
    public List<ApHotArticles> selectExpireMonth() {
        return apHotArticlesMapper.selectExpireMonth();
    }

    /**
     * 删除过去的热数据
     *
     * @param apHotArticle
     */
    @Override
    public void deleteHotData(ApHotArticles apHotArticle) {
        deleteById(apHotArticle.getId());
        String rowKey = DataConvertUtils.toString(apHotArticle.getId());

 hBaseStorageClient.gethBaseClent().deleteRow(HBaseConstants.APARTICLE_QUANTITY_
TABLE_NAME, rowKey);
        MongoStorageEntity mongoStorageEntity = mongoTemplate.findById(rowKey,
MongoStorageEntity.class);
        if (null != mongoStorageEntity) {
            mongoTemplate.remove(mongoStorageEntity);
        }
    }

    /**
     * 插入操作
```

```
     *
     * @param apHotArticles
     */
    @Override
    public void insert(ApHotArticles apHotArticles) {
        apHotArticlesMapper.insert(apHotArticles);
    }

    /**
     * 热点数据同步方法
     *
     * @param apArticleId
     */
    @Override
    public void hotApArticleSync(Integer apArticleId) {
        log.info("开始将热数据同步，apArticleId：{}", apArticleId);
        ArticleQuantity articleQuantity = getHotArticleQuantity(apArticleId);
        if (null != articleQuantity) {
            //热点数据同步到DB中
            hotApArticleToDBSync(articleQuantity);
            //热点数据同步到MONGO
            hotApArticleMongoSync(articleQuantity);
            log.info("热数据同步完成，apArticleId：{}", apArticleId);
        } else {
            log.error("找不到对应的热数据，apArticleId：{}", apArticleId);
        }
    }

    /**
     * 获取热数据的ArticleQuantity 对象
     *
     * @param apArticleId
     * @return
     */
    private ArticleQuantity getHotArticleQuantity(Integer apArticleId) {
        Long id = Long.valueOf(apArticleId);
        ArticleQuantity articleQuantity =
articleQuantityService.getArticleQuantityByArticleId(id);
        if (null == articleQuantity) {
            articleQuantity =
articleQuantityService.getArticleQuantityByArticleIdForHbase(id);
        }
        return articleQuantity;
    }


    /**
     * 热数据 到数据库Mysql的同步
     *
     * @param articleQuantity
     */
    public void hotApArticleToDBSync(ArticleQuantity articleQuantity) {
        Integer apArticleId = articleQuantity.getApArticleId();
        log.info("开始将热数据从Hbase同步到mysql，apArticleId：{}", apArticleId);
        if (null == apArticleId) {
            log.error("apArticleId不存在无法进行同步");
            return;
        }
```

```
        ApHotArticles apHotArticlesQuery = new ApHotArticles() {{
            setArticleId(apArticleId);
        }};
        List<ApHotArticles> apHotArticlesList =
apHotArticlesMapper.selectList(apHotArticlesQuery);
        if (null != apHotArticlesList && !apHotArticlesList.isEmpty()) {
            log.info("Mysql数据已同步过不需要再次同步,apArticleId:{}", apArticleId);
        } else {
            ApHotArticles apHotArticles = articleQuantity.getApHotArticles();
            apHotArticlesMapper.insert(apHotArticles);
        }
        log.info("将热数据从Hbase同步到mysql完成，apArticleId：{}", apArticleId);
    }

    /**
     * 热数据向从Hbase到Mongodb同步
     *
     * @param articleQuantity
     */
    public void hotApArticleMongoSync(ArticleQuantity articleQuantity) {
        Integer apArticleId = articleQuantity.getApArticleId();
        log.info("开始将热数据从Hbase同步到MongoDB，apArticleId：{}", apArticleId);
        if (null == apArticleId) {
            log.error("apArticleId不存在无法进行同步");
            return;
        }
        String rowKeyId = DataConvertUtils.toString(apArticleId);
        MongoStorageEntity mongoStorageEntity = mongoTemplate.findById(rowKeyId,
MongoStorageEntity.class);
        if (null != mongoStorageEntity) {
            log.info("MongoDB数据已同步过不需要再次同步,apArticleId:{}",
apArticleId);
        } else {
            List<StorageData> storageDataList =
articleQuantity.getStorageDataList();
            if (null != storageDataList && !storageDataList.isEmpty()) {
                mongoStorageEntity = new MongoStorageEntity();
                mongoStorageEntity.setDataList(storageDataList);
                mongoStorageEntity.setRowKey(rowKeyId);
                mongoTemplate.insert(mongoStorageEntity);
            }

        }
        log.info("将热数据从Hbase同步到MongoDB完成，apArticleId：{}", apArticleId);
    }
}
```

8 定时同步数据

8.1 全量数据从mysql同步到HBase

```
@Component
@DisallowConcurrentExecution
@Log4j2
/**
```

```
 * 全量数据从mysql 同步到HBase
 */
public class MigrationDbToHBaseQuartz extends AbstractJob {

    @Autowired
    private ArticleQuantityService articleQuantityService;


    @Override
    public String[] triggerCron() {
        /**
         * 2019/8/9 10:15:00
         * 2019/8/9 10:20:00
         * 2019/8/9 10:25:00
         * 2019/8/9 10:30:00
         * 2019/8/9 10:35:00
         */
        return new String[]{"0 0/5 * * * ?"};
    }

    @Override
    protected void executeInternal(JobExecutionContext jobExecutionContext)
throws JobExecutionException {
        log.info("开始进行数据库到HBASE同步任务");
        articleQuantityService.dbToHbase();
        log.info("数据库到HBASE同步任务完成");
    }

}
```

8.2 定期删除过期的数据

```
/**
 * 定期删除过期的数据
 */
@Component
@Log4j2
public class MigrationDeleteHotDataQuartz extends AbstractJob {

    @Autowired
    private ApHotArticleService apHotArticleService;


    @Override
    public String[] triggerCron() {
        /**
         * 2019/8/9 22:30:00
         * 2019/8/10 22:30:00
         * 2019/8/11 22:30:00
         * 2019/8/12 22:30:00
         * 2019/8/13 22:30:00
         */
        return new String[]{"0 30 22 * * ?"};
    }

    @Override
```

```
    protected void executeInternal(JobExecutionContext jobExecutionContext)
throws JobExecutionException {
        long cutrrentTime = System.currentTimeMillis();
        log.info("开始删除数据库过期数据");
        deleteExpireHotData();
        log.info("删除数据库过期数据结束，耗时:{}", System.currentTimeMillis() -
cutrrentTime);
    }

    /**
     * 删除过期的热数据
     */
    public void deleteExpireHotData() {
        List<ApHotArticles> apHotArticlesList =
apHotArticleService.selectExpireMonth();
        if (null != apHotArticlesList && !apHotArticlesList.isEmpty()) {
            for (ApHotArticles apHotArticle : apHotArticlesList) {
                apHotArticleService.deleteHotData(apHotArticle);
            }
        }
    }

}
```

9 消息接收同步数据

9.1 文章审核成功同步

9.1.1 消息发送

（1）消息名称定义及消息发送方法声明

maven_test.properties

```
kafka.topic.article-audit-success=kafka.topic.article.audit.success.sigle.test
```

kafka.properties

```
kafka.topic.article-audit-success=${kafka.topic.article-audit-success}
```

com.heima.common.kafka.KafkaTopicConfig新增属性

```
/**
     * 审核成功
     */
String articleAuditSuccess;
```

com.heima.common.kafka.KafkaSender

```
/**
     * 发送审核成功消息
     */
public void sendArticleAuditSuccessMessage(ArticleAuditSuccess message) {
    ArticleAuditSuccessMessage temp = new ArticleAuditSuccessMessage();
    temp.setData(message);
    this.sendMesssage(kafkaTopicConfig.getArticleAuditSuccess(),
UUID.randomUUID().toString(), temp);
}
```

（2）修改自动审核代码，爬虫和自媒体都要修改

在审核成功后，发送消息

爬虫

```
//文章审核成功
ArticleAuditSuccess articleAuditSuccess = new ArticleAuditSuccess();
articleAuditSuccess.setArticleId(apArticle.getId());
articleAuditSuccess.setType(ArticleAuditSuccess.ArticleType.CRAWLER);
articleAuditSuccess.setChannelId(apArticle.getChannelId());

kafkaSender.sendArticleAuditSuccessMessage(articleAuditSuccess);
```

自媒体

```
//文章审核成功
ArticleAuditSuccess articleAuditSuccess = new ArticleAuditSuccess();
articleAuditSuccess.setArticleId(apArticle.getId());
articleAuditSuccess.setType(ArticleAuditSuccess.ArticleType.MEDIA);
articleAuditSuccess.setChannelId(apArticle.getChannelId());

kafkaSender.sendArticleAuditSuccessMessage(articleAuditSuccess);
```

9.1.2消息接收

```
/**
 * 热点文章监听类
 */
@Component
@Log4j2
public class MigrationAuditSucessArticleListener implements
KafkaListener<String, String> {
    /**
     * 通用转换mapper
     */
    @Autowired
    ObjectMapper mapper;
    /**
     * kafka 主题 配置
     */
    @Autowired
    KafkaTopicConfig kafkaTopicConfig;
```

```
    @Autowired
    private ArticleQuantityService articleQuantityService;


    @Override
    public String topic() {
        return kafkaTopicConfig.getArticleAuditSuccess();
    }

    /**
     * 监听消息
     *
     * @param data
     * @param consumer
     */
    @Override
    public void onMessage(ConsumerRecord<String, String> data, Consumer<?, ?>
consumer) {
        log.info("kafka接收到审核通过消息:{}", data);
        String value = (String) data.value();
        if (null != value) {
            ArticleAuditSuccessMessage message = null;
            try {
                message = mapper.readValue(value,
ArticleAuditSuccessMessage.class);
            } catch (IOException e) {
                e.printStackTrace();
            }
            ArticleAuditSuccess auto = message.getData();
            if (null != auto) {
                //调用方法 将HBAESE中的热数据进行同步
                Integer articleId = auto.getArticleId();
                if (null != articleId) {
                    articleQuantityService.dbToHbase(articleId);
                }
            }
        }

    }
}
```

9.2 热点文章同步

创建监听类：com.heima.migration.kafka.listener.MigrationHotArticleListener

```
/**
 * 热点文章监听类
 */
@Component
@Log4j2
public class MigrationHotArticleListener implements KafkaListener<String,
String> {
    /**
     * 通用转换mapper
     */
    @Autowired
```

```
    ObjectMapper mapper;
    /**
     * kafka 主题 配置
     */
    @Autowired
    KafkaTopicConfig kafkaTopicConfig;
    /**
     * 热点文章service注入
     */
    @Autowired
    private ApHotArticleService apHotArticleService;


    @Override
    public String topic() {
        return kafkaTopicConfig.getHotArticle();
    }

    /**
     * 监听消息
     *
     * @param data
     * @param consumer
     */
    @Override
    public void onMessage(ConsumerRecord<String, String> data, Consumer<?, ?>
consumer) {
        log.info("kafka接收到热数据同步消息:{}", data);
        String value = (String) data.value();
        if (null != value) {
            ApHotArticleMessage message = null;
            try {
                message = mapper.readValue(value, ApHotArticleMessage.class);
            } catch (IOException e) {
                e.printStackTrace();
            }
            Integer articleId = message.getData().getArticleId();
            if (null != articleId) {
                //调用方法 将HBAESE中的热数据进行同步
                apHotArticleService.hotApArticleSync(articleId);
            }
        }
    }
}
```