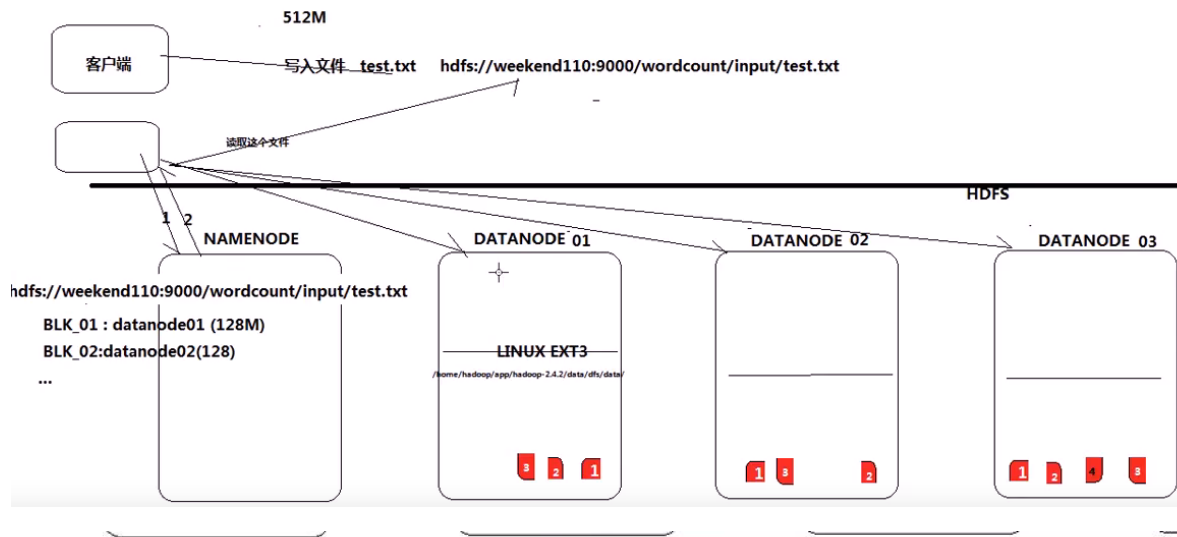
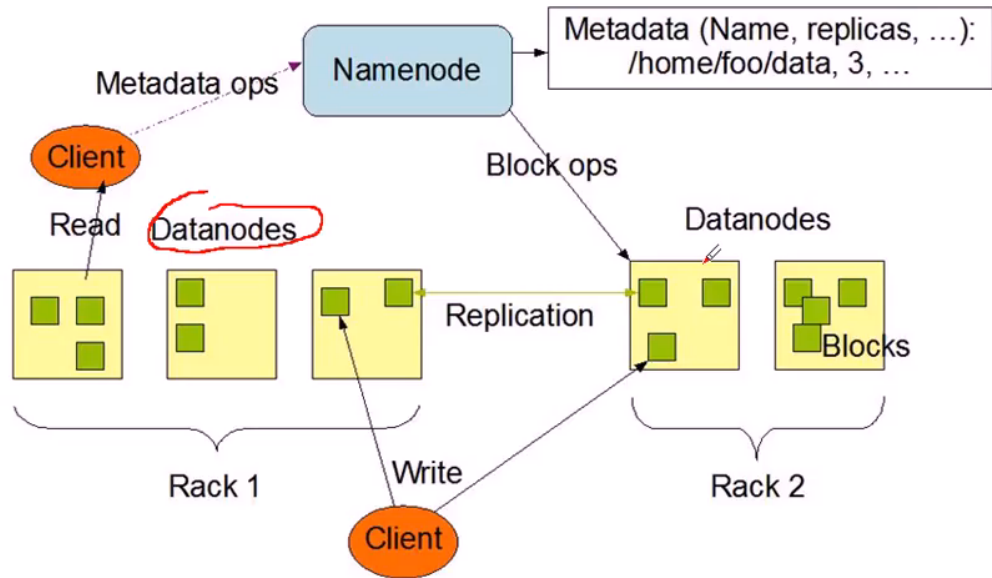
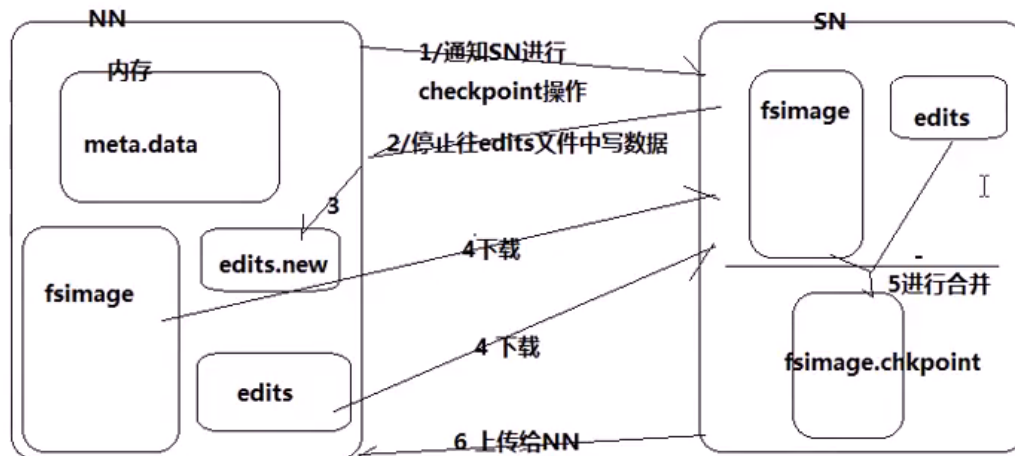


HDFS Architecture

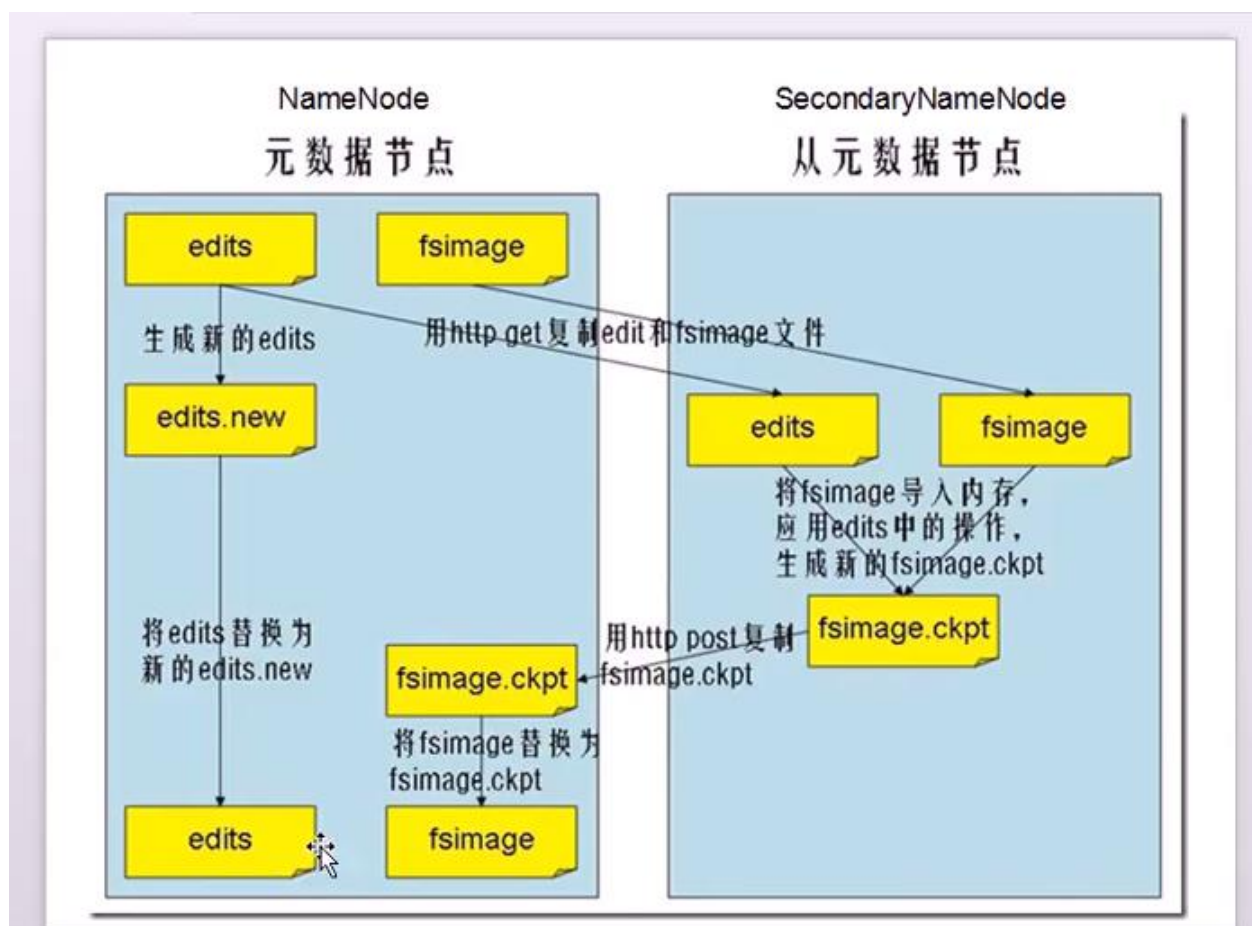


- HDFS的实现思想：
- 1、hdfs是通过分布式集群来存储文件，为客户端提供了一个便捷的访问方式，就是一个虚拟的目录结构
 - 2、文件存储到hdfs集群中去的时候是被切分成block的
 - 3、文件的block存放在若干台datanode节点上
 - 4、hdfs文件系统中的文件与真实的block之间有映射关系，由namenode管理
 - 5、每一个block在集群中会存储多个副本，好处是可以提高数据的可靠性，还可以提高访问的吞吐量

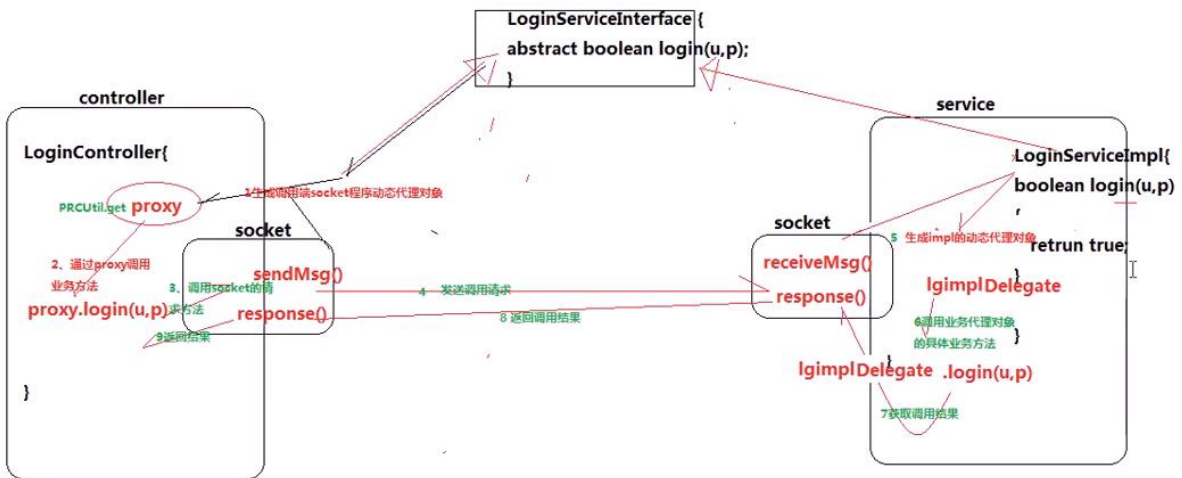
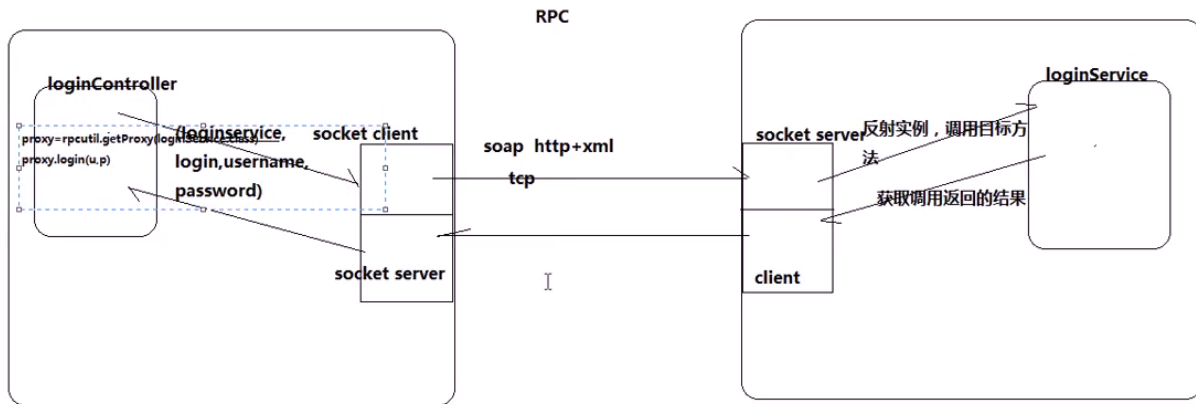


NN的职责：

- 1、维护元数据信息
- 2、维护hdfs的目录树
- 3、响应客户端的请求



远程过程调用的底层实现机制



```

FileSystem.c... DistributedF... DFSClient.class DFSInputStr... BlockReader... WCMapper.java WReducer.java »
1 //map 和 reduce 的数据输入输出都是以 key-value 对的形式封装的
2 //默认情况下，框架传递给我们的 mapper 的输入数据中，key 是要处理的文本中一行的起始偏移量，这一行的内容作为 value
3 public class WCMapper extends Mapper<LongWritable, Text, Text, LongWritable>{
4
5     //mapreduce 框架每读一行数据就调用一次该方法
6     @Override
7     protected void map(LongWritable key, Text value, Context context)
8         throws IOException, InterruptedException {
9         //具体业务逻辑就写在这个方法体中，而且我们业务要处理的数据已经被框架传递进来，在方法的参数中 key-value
10        //key 是这一行数据的起始偏移量 value 是这一行的文本内容
11
12        //将这一行的内容转换成 string 类型
13        String line = value.toString();
14
15        //对这一行的文本按特定分隔符切分
16        String[] words = StringUtils.split(line, " ");
17
18        //遍历这个单词数组输出为 kv 形式 k: 单词 v : 1
19        for(String word : words){
20
21            context.write(new Text(word), new LongWritable(1));
22
23        }
24
25

```

```

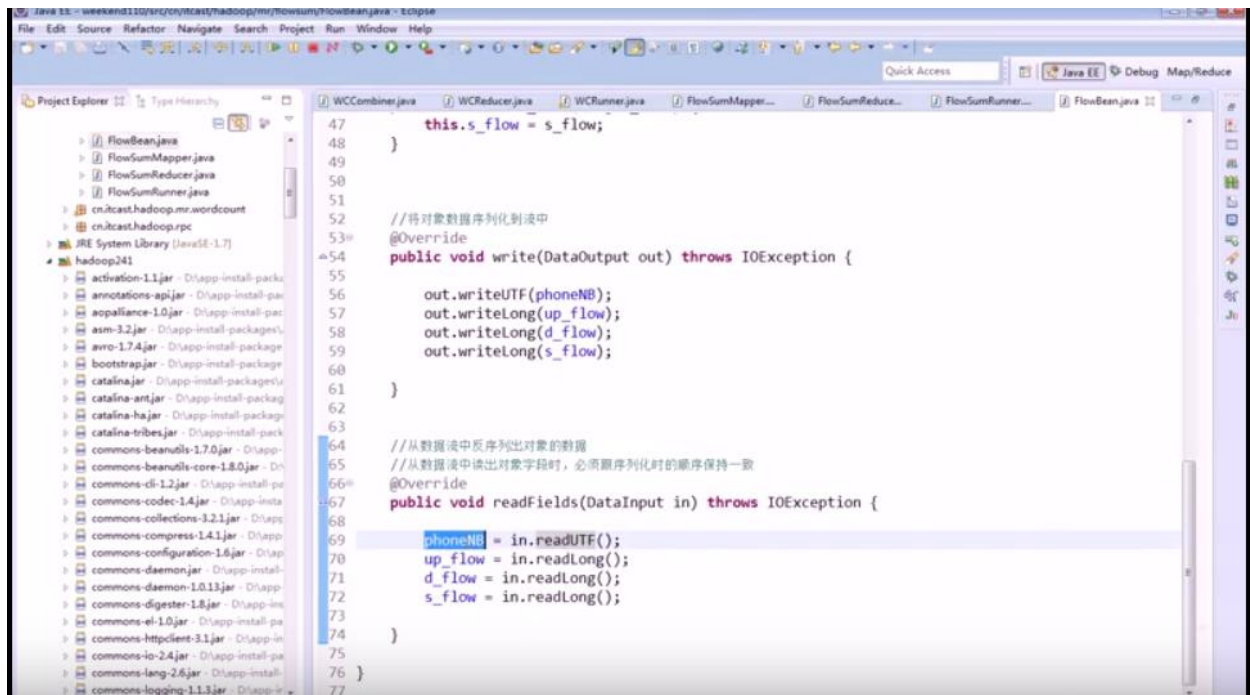
FileSystem.c... DistributedF... DFSClient.class DFSInputStr... BlockReader... WCMapper.java WReducer.java »
4
5 import org.apache.hadoop.io.LongWritable;
6 import org.apache.hadoop.io.Text;
7 import org.apache.hadoop.mapreduce.Reducer;
8
9 public class WReducer extends Reducer<Text, LongWritable, Text, LongWritable>{
10
11
12
13    //框架在 map 处理完成之后，将所有 kv 对缓存起来，进行分组，然后传递一个组<key, values{}>，调用一次 reduce 方法
14    //<hello, {1,1,1,1,1,1,1,1...}>
15    @Override
16    protected void reduce(Text key, Iterable<LongWritable> values, Context context)
17        throws IOException, InterruptedException {
18
19        long count = 0;
20        for(LongWritable value: values){
21
22            count += value.get();
23
24        }
25
26
27
28    }

```

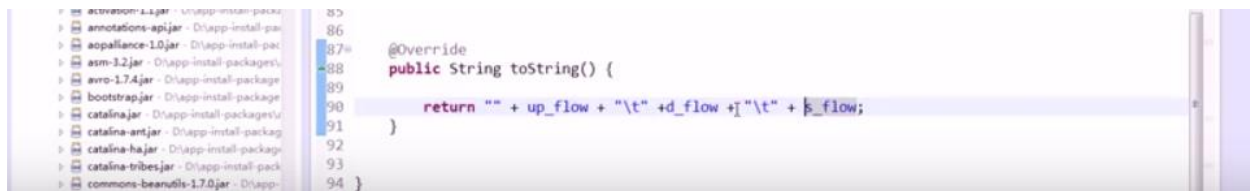
```
context.write(key, new LongWritable(count));
```

```
Quick Access
SolrDao.java  DFSCient.class  DFSInputStr...  BlockReader....  *WCRRunner.java

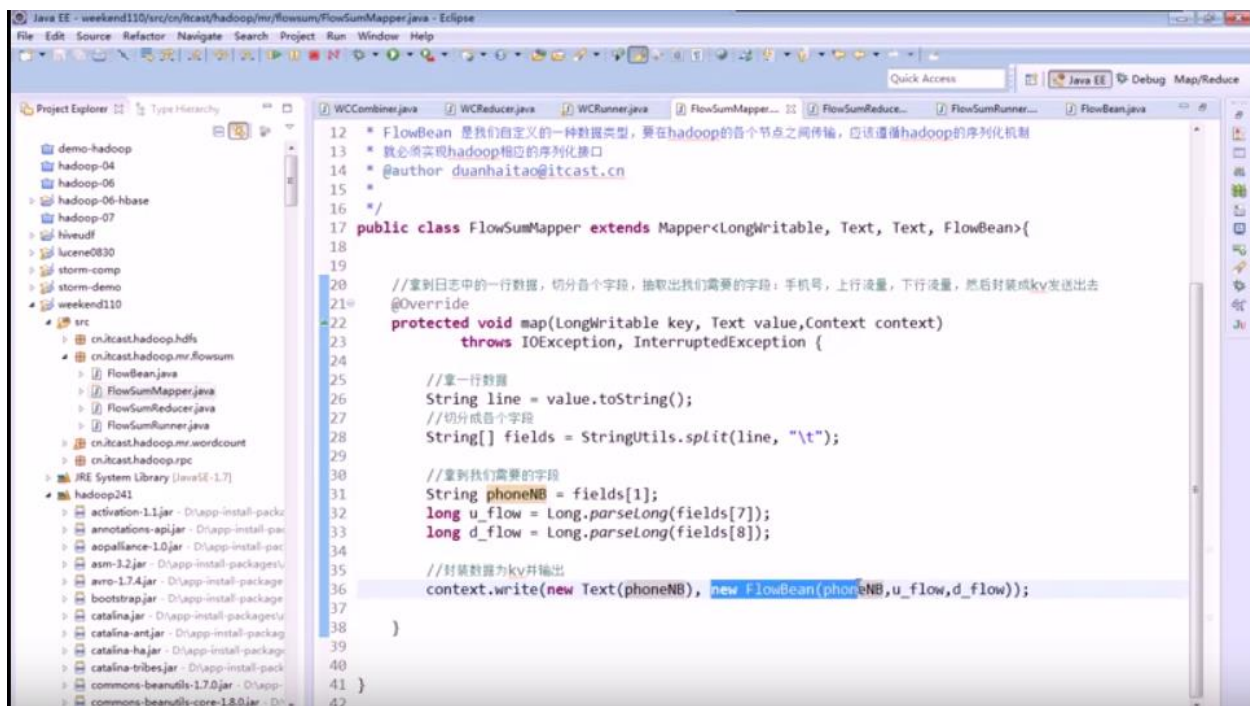
20
21 public static void main(String[] args) throws Exception {
22
23     Configuration conf = new Configuration();
24
25     Job job = Job.getInstance(conf);
26
27     //本job使用的mapper和reducer的类
28     job.setMapperClass(WCMapper.class);
29     job.setReducerClass(WCReducer.class);
30
31
32     //指定reduce的输出数据kv类型
33     job.setOutputKeyClass(Text.class);
34     job.setOutputValueClass(LongWritable.class);
35
36
37     job.setMapOutputKeyClass(Text.class);
38     job.setMapOutputValueClass(LongWritable.class);
39
40     wcjob.setOutputKeyClass(Text.class);
41     wcjob.setOutputValueClass(LongWritable.class);
42
43     //指定mapper的输出数据kv类型
44     wcjob.setMapOutputKeyClass(Text.class);
45     wcjob.setMapOutputValueClass(LongWritable.class);
46
47
48     //指定要处理的输入数据存放路径
49     FileInputFormat.setInputPaths(wcjob, new Path("/wc/srcdata/"));
50
51     //指定处理结果的输出数据存放路径
52     FileOutputFormat.setOutputPath(wcjob, new Path("/wc/output/"));
53
54     //将job提交给集群运行
55     wcjob.waitForCompletion(true);
56
57
58 }
```

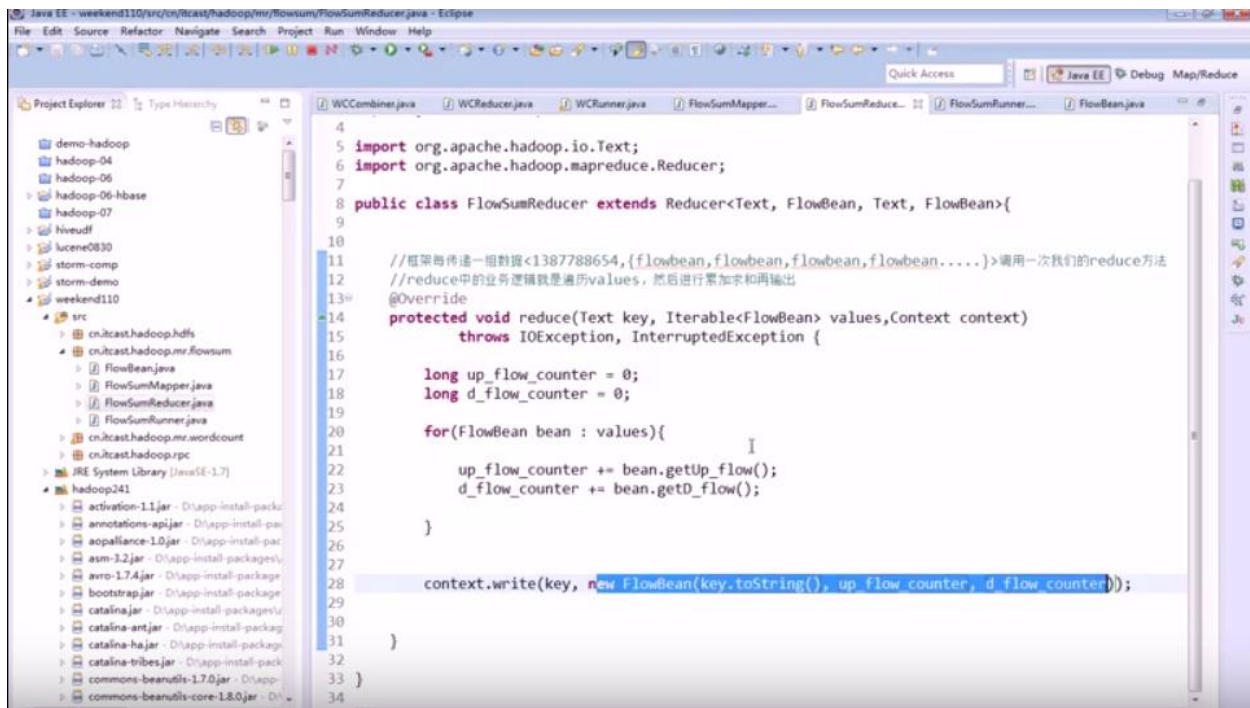
```
47         this.s_flow = s_flow;
48     }
49
50
51
52     //将对象数据序列化到流中
53     @Override
54     public void write(DataOutput out) throws IOException {
55
56         out.writeUTF(phoneNB);
57         out.writeLong(up_flow);
58         out.writeLong(d_flow);
59         out.writeLong(s_flow);
60     }
61
62
63
64     //从数据流中反序列化对象的数据
65     //从数据流中读出对象字段时，必须跟序列化时的顺序保持一致
66     @Override
67     public void readFields(DataInput in) throws IOException {
68
69         phoneNB = in.readUTF();
70         up_flow = in.readLong();
71         d_flow = in.readLong();
72         s_flow = in.readLong();
73     }
74
75
76
77
```



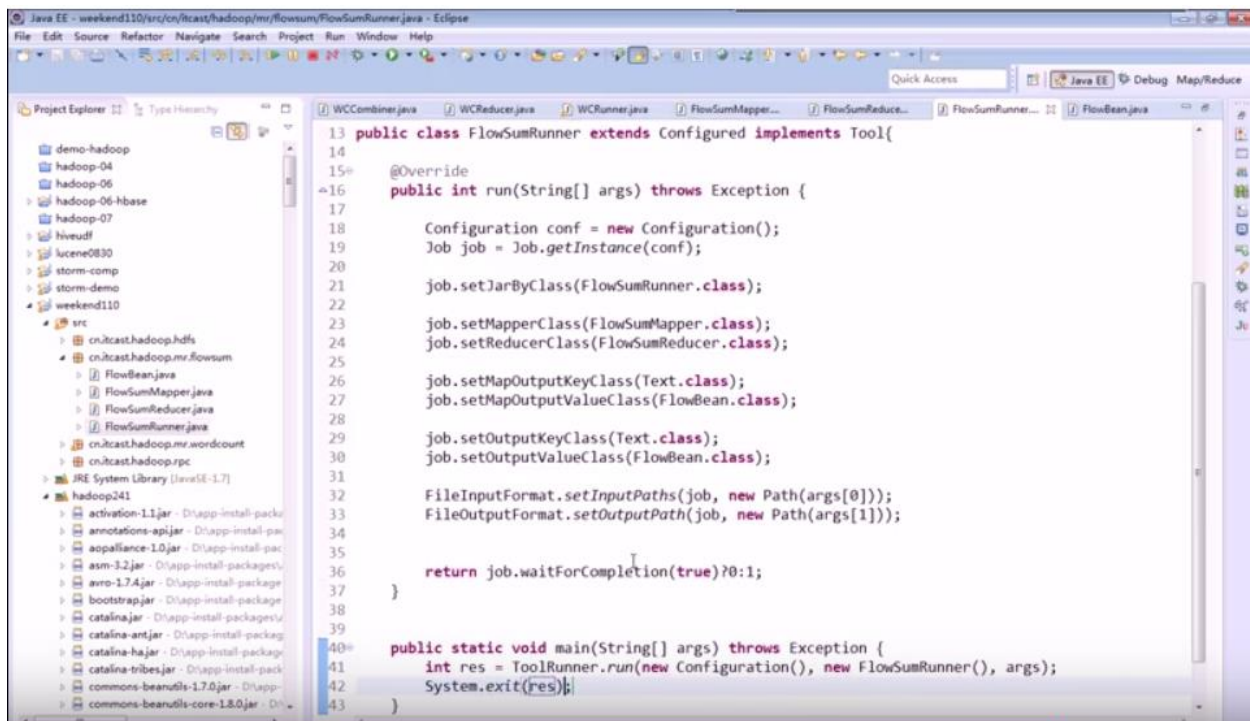
```
86
87
88     @Override
89     public String toString() {
90
91         return "" + up_flow + "\t" + d_flow + "\t" + s_flow;
92     }
93
94
```



```
12  * FlowBean 是我们自定义的一种数据类型，要在hadoop的各个节点之间传输，应该遵循hadoop的序列化机制
13  * 数必须实现hadoop相应的序列化接口
14  * @author duanhaitao@itcast.cn
15  */
16
17 public class FlowSumMapper extends Mapper<LongWritable, Text, Text, FlowBean>{
18
19
20     //拿到日志中的一行数据，切分各个字段，抽取我们需要的字段：手机号，上行流量，下行流量，然后封装成kv发送出去
21     @Override
22     protected void map(LongWritable key, Text value, Context context)
23         throws IOException, InterruptedException {
24
25         //拿一行数据
26         String line = value.toString();
27         //切分成各个字段
28         String[] fields = StringUtils.split(line, "\t");
29
30         //拿到我们需要的字段
31         String phoneNB = fields[1];
32         long u_flow = Long.parseLong(fields[7]);
33         long d_flow = Long.parseLong(fields[8]);
34
35         //封装数据为kv并输出
36         context.write(new Text(phoneNB), new FlowBean(phoneNB, u_flow, d_flow));
37     }
38
39
40
41
42
```



```
4
5 import org.apache.hadoop.io.Text;
6 import org.apache.hadoop.mapreduce.Reducer;
7
8 public class FlowSumReducer extends Reducer<Text, FlowBean, Text, FlowBean>{
9
10
11 //框架会传递一组数据<1387788654,{flowbean,flowbean,flowbean,flowbean....}>调用一次我们的reduce方法
12 //reduce中的业务逻辑就是遍历values, 然后进行累加和再输出
13 @Override
14 protected void reduce(Text key, Iterable<FlowBean> values,Context context)
15     throws IOException, InterruptedException {
16
17     long up_flow_counter = 0;
18     long d_flow_counter = 0;
19
20     for(FlowBean bean : values){
21
22         up_flow_counter += bean.getUp_flow();
23         d_flow_counter += bean.getD_flow();
24     }
25
26     context.write(key, new FlowBean(key.toString(), up_flow_counter, d_flow_counter));
27
28 }
29
30
31
32
33 }
34
```



```
13 public class FlowSumRunner extends Configured implements Tool{
14
15
16 @Override
17 public int run(String[] args) throws Exception {
18
19     Configuration conf = new Configuration();
20     Job job = Job.getInstance(conf);
21
22     job.setJarByClass(FlowSumRunner.class);
23
24     job.setMapperClass(FlowSumMapper.class);
25     job.setReducerClass(FlowSumReducer.class);
26
27     job.setMapOutputKeyClass(Text.class);
28     job.setMapOutputValueClass(FlowBean.class);
29
30     job.setOutputKeyClass(Text.class);
31     job.setOutputValueClass(FlowBean.class);
32
33     FileInputFormat.setInputPaths(job, new Path(args[0]));
34     FileOutputFormat.setOutputPath(job, new Path(args[1]));
35
36     return job.waitForCompletion(true)?0:1;
37 }
38
39
40 public static void main(String[] args) throws Exception {
41     int res = ToolRunner.run(new Configuration(), new FlowSumRunner(), args);
42     System.exit(res);
43 }
44
```



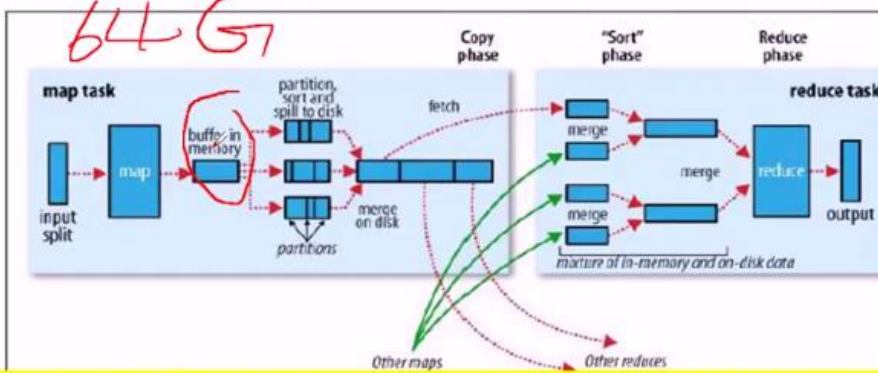
```
//设置我们自定义的分组逻辑定义
job.setPartitionerClass(AreaPartitioner.class);
```

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(FlowBean.class);
```

```
//设置reduce的任务并发数，应该跟分组的数量保持一致
job.setNumReduceTasks(6);
```

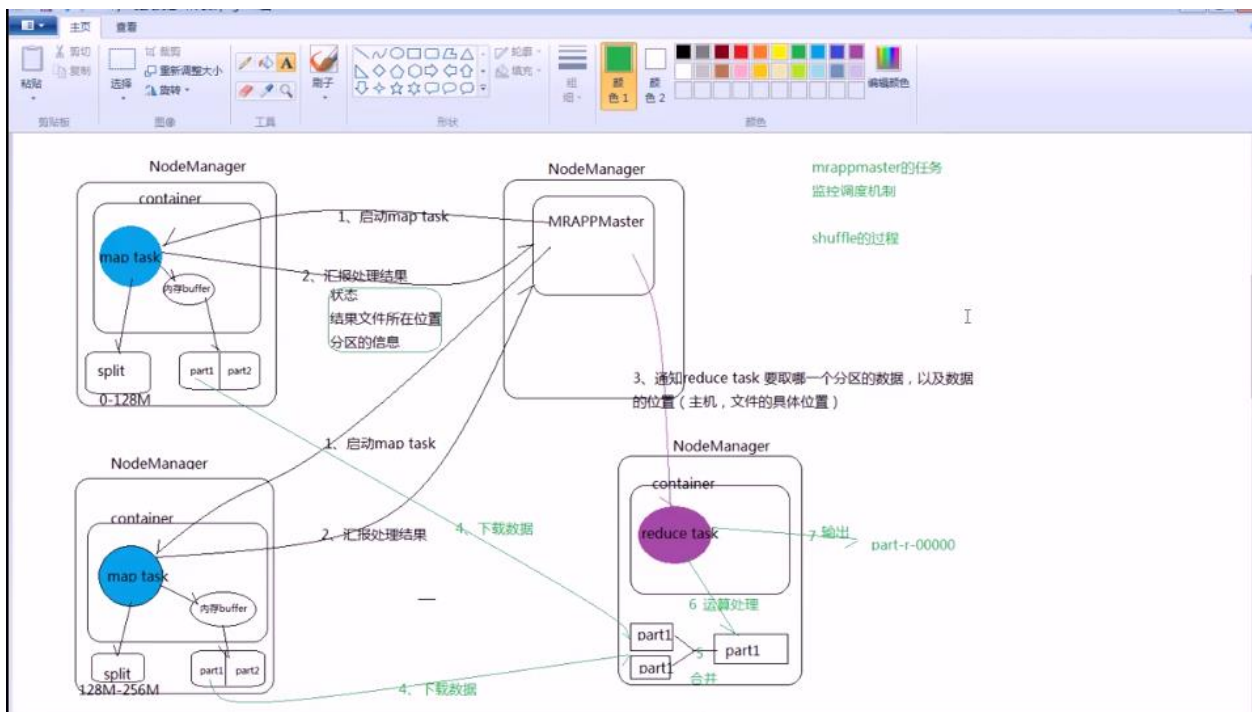
```
FlowSumArea.java  FlowSumMapper.java  AreaPartitioner.java  11
3 import java.util.HashMap;
4
5 import org.apache.hadoop.mapreduce.Partitioner;
6
7 public class AreaPartitioner<KEY, VALUE> extends Partitioner<KEY, VALUE>{
8
9     private static HashMap<String,Integer> areaMap = new HashMap<>();
10
11     static{
12         areaMap.put("135", 0);
13         areaMap.put("136", 1);
14         areaMap.put("137", 2);
15         areaMap.put("138", 3);
16         areaMap.put("139", 4);
17     }
18
19
20
21
22
23 @Override
24 public int getPartition(KEY key, VALUE value, int numPartitions) {
25     //从key中拿到手机号，查询手机归属地字典，不同的省份返回不同的组号
26
27     int areaCoder = areaMap.get(key.toString().substring(0, 3))==null?5:areaMap.get(key.toSt
28
29     return areaCoder;
30 }
31
32 }
```

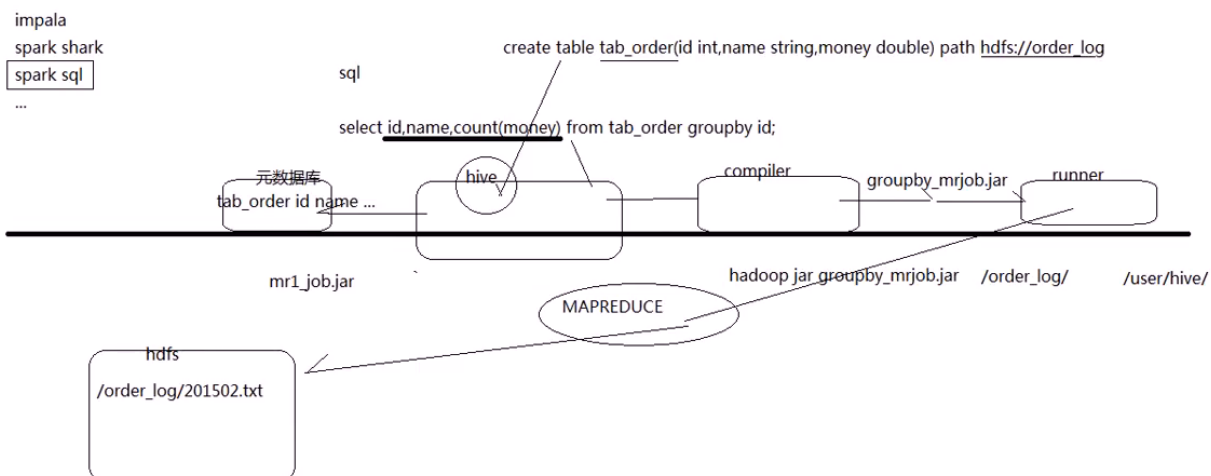
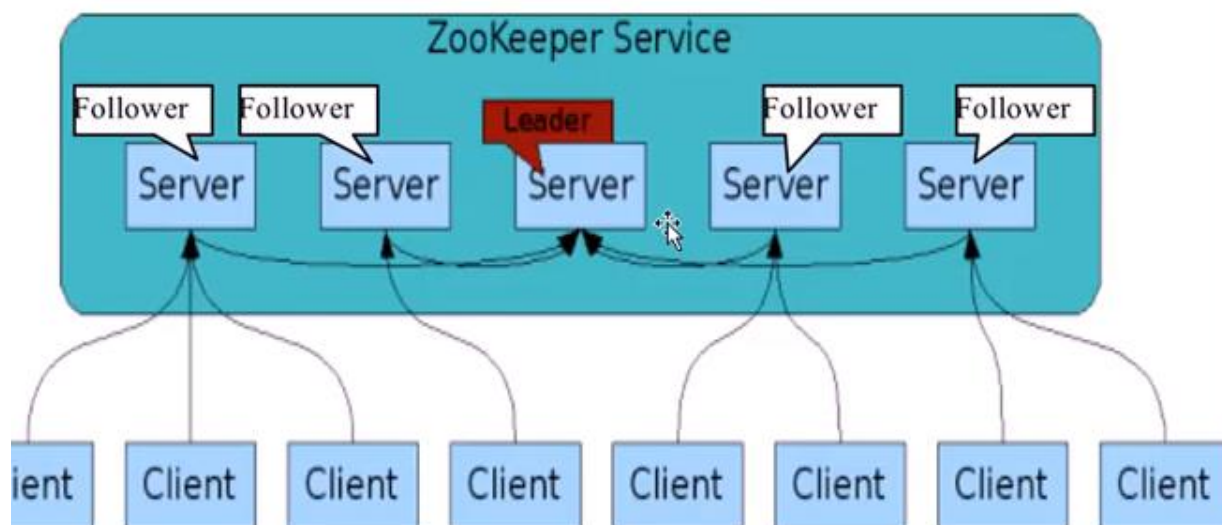
Shuffle

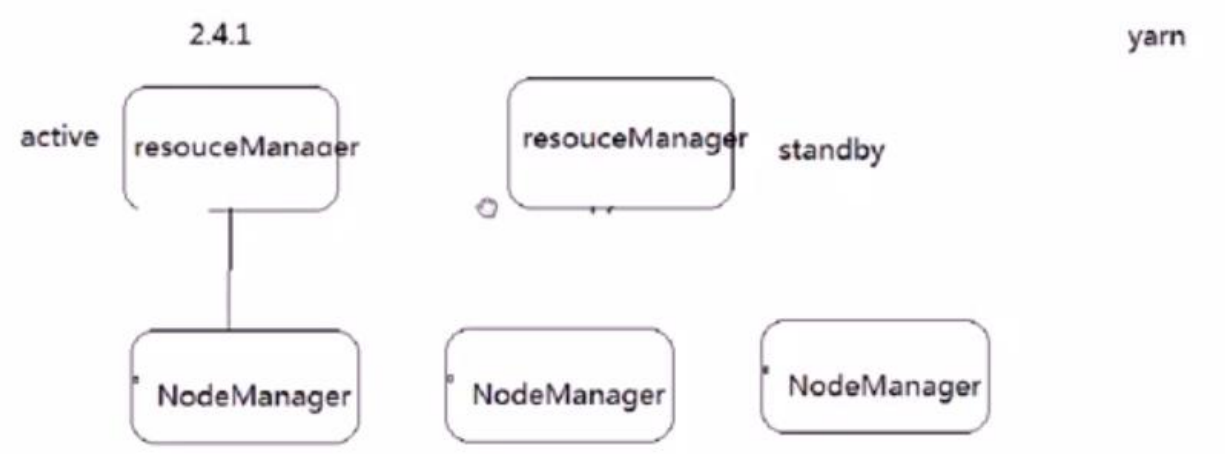
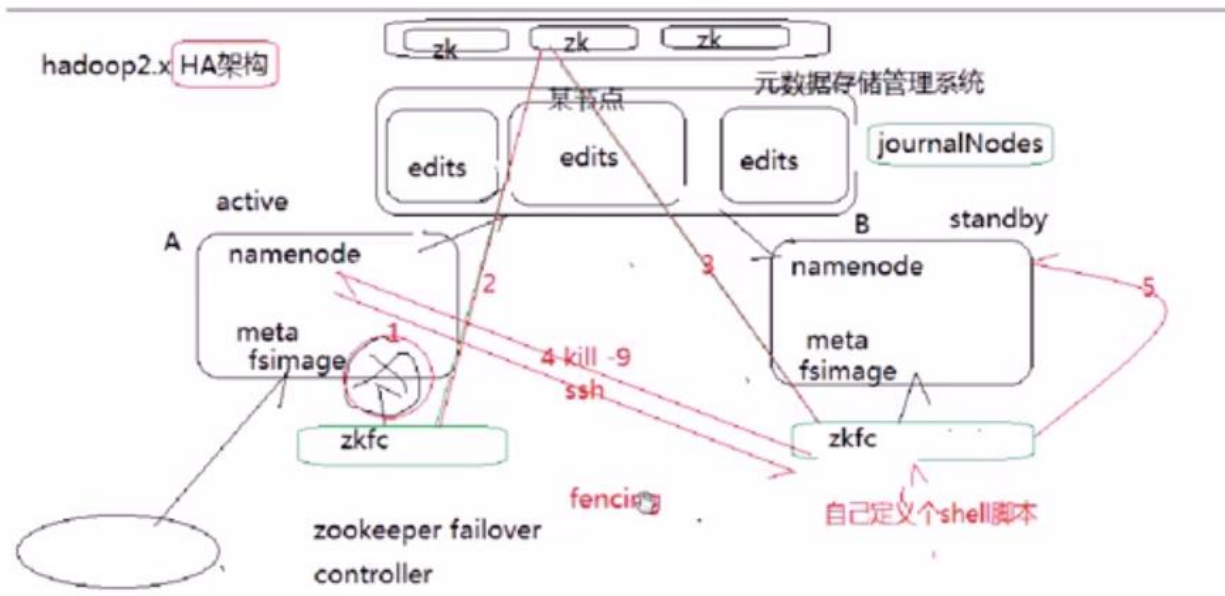


- 1.每个map有一个环形内存缓冲区，用于存储任务的输出。默认大小100MB（io.sort.mb属性），一旦达到阈值0.8（io.sort.spill.percent），一个后台线程把内容写到(spill)磁盘的指定目录（mapred.local.dir）下的新建的一个溢出写文件。
- 2.写磁盘前，要partition,sort。如果有combiner，combine排序后数据。
- 3.等最后记录写完，合并全部溢出写文件为一个分区且排序的文件。

- 1.Reducer通过Http方式得到输出文件的分区。
- 2.TaskTracker为分区文件运行Reduce任务。复制阶段把Map输出复制到Reducer的内存或磁盘。一个Map任务完成，Reduce就开始复制输出。
- 3.排序阶段合并map输出。然后走Reduce阶段。







ndase	列族 baseinfo			列族 extrainfo	
000001	name:fengjie	age:18	career:huyou	hobbies:chuiniu	married:true
000002	name:furong	age:28	xueli:benke	jiguan:hubei	