概述

- 大数据的特征：数据量大、数据类型繁多、处理速度快、价值密度低
- 大数据技术包含 P15
- 大数据的核心：（猜测）云计算、物联网 P18
- 大数据的来源
  - 大数据的发展历程 P6
  - 数据来源：交易数据、移动通信数据、人为数据、机器和传感数据、互联网上的开放数据

Hadoop

- 安装模式：单机模式、伪分布模式、全分布模式 P33
- 核心是 HDFS 和 MapReduce P31
- 生态系统 P31
- 版本 P30
- 局限性 P155

HDFS

- 结构模型 P43、P47（NameNode、DataNode）
- 特点：设计需求P44、实现目标P45
- SecondaryNameNode 的作用 P47
- 通信协议 P49
- 冗余因子 P50
- 数据存储策略 P51
- 读写特征 P53
- 常用命令 P55

MapReduce

- 体系结构：（百度）Client、JobTracker、TaskTracker以及Task
- 工作流程：P134
- shuffle过程：P136
- Reduce端的shuffle过程：领取数据、归并数据、把数据输入给Reduce任务 P138

HBase

- NoSQL的列族数据库：P100
- 底层数据存在：HDFS P64
- 表的索引：行键、列祖、列限定符、时间戳 P66
- 三层结构：P73
- 强大的计算能力：MapReduce P64
- 系统架构：P74
- 启动后进程：NameNode、SecondaryNameNode、DataNode、HRegionServer、Jps、HQuorumPeer、HMaster
- 基本shell命令 P78

Spark

- 特点：P173
- 生态系统包含组件：Spark Core、Spark SQL、Spark Streaming、MLib、GraphX P176
- 运行框架 P177

RDD

- 特点：P181
- 依赖关系分类：P182
- 操作分类： P187
- 转换操作和行动操作：P188

HBase 编程

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.hbase.*;
import org.apache.hadoop.hbase.client.*;
import java.io.IOException;

public class ExampleForHbase{
    public static Configuration configuration;
    public static Connection connection;
    public static Admin admin;

    //主函数中的语句请逐句执行，只需删除其前的//即可，如：执行insertRow时请将其他语句注释
    public static void main(String[] args)throws IOException{
        //创建一个表，表名为Score，列族为sname,course
        createTable("Score",new String[]{"sname","course"});

        //在Score表中插入一条数据，其行键为95001,sname为Mary（因为sname列族下没有子列所以第四个参数为空）
```

```
        //等价命令：put 'Score','95001','sname','Mary'
        //insertRow("Score", "95001", "sname", "", "Mary");
        //在Score表中插入一条数据，其行键为95001,course:Math为88（course为列族，Math为course下的子列）
        //等价命令：put 'Score','95001','score:Math','88'
        //insertRow("Score", "95001", "course", "Math", "88");
        //在Score表中插入一条数据，其行键为95001,course:English为85（course为列族，English为course下的子列）
        //等价命令：put 'Score','95001','score:English','85'
        //insertRow("Score", "95001", "course", "English", "85");

        //1、删除Score表中指定列数据，其行键为95001,列族为course，列为Math
        //执行这句代码前请deleteRow方法的定义中，将删除指定列数据的代码取消注释注释，将删除制定列族的代码注释
        //等价命令：delete 'Score','95001','score:Math'
        //deleteRow("Score", "95001", "course", "Math");

        //2、删除Score表中指定列族数据，其行键为95001,列族为course（95001的Math和English的值都会被删除）
        //执行这句代码前请deleteRow方法的定义中，将删除指定列数据的代码注释，将删除制定列族的代码取消注释
        //等价命令：delete 'Score','95001','score'
        //deleteRow("Score", "95001", "course", "");

        //3、删除Score表中指定行数据，其行键为95001
        //执行这句代码前请deleteRow方法的定义中，将删除指定列数据的代码注释，以及将删除制定列族的代码注释
        //等价命令：deleteall 'Score','95001'
        //deleteRow("Score", "95001", "", "");

        //查询Score表中，行键为95001，列族为course，列为Math的值
        //getData("Score", "95001", "course", "Math");
        //查询Score表中，行键为95001，列族为sname的值（因为sname列族下没有子列所以第四个参数为空）
        //getData("Score", "95001", "sname", "");

        //删除Score表
        //deleteTable("Score");
    }

    //建立连接
    public static void init(){
        configuration  = HBaseConfiguration.create();
        configuration.set("hbase.rootdir","hdfs://localhost:9000/hbase");
        try{
            connection = ConnectionFactory.createConnection(configuration);
            admin = connection.getAdmin();
        }catch (IOException e){
            e.printStackTrace();
        }
    }
    //关闭连接
    public static void close(){
        try{
            if(admin != null){
                admin.close();
            }
            if(null != connection){
                connection.close();
```

```
            }
        }catch (IOException e){
            e.printStackTrace();
        }
    }

    /**
     * 建表。HBase的表中会有一个系统默认的属性作为主键，主键无需自行创建，默认为put命令操作中表名后第一个数据，因此此处无需创建id列
     * @param myTableName 表名
     * @param colFamily 列族名
     * @throws IOException
     */
    public static void createTable(String myTableName,String[] colFamily) throws IOException {

        init();
        TableName tableName = TableName.valueOf(myTableName);

        if(admin.tableExists(tableName)){
            System.out.println("talbe is exists!");
        }else {
            HTableDescriptor hTableDescriptor = new HTableDescriptor(tableName);
            for(String str:colFamily){
                HColumnDescriptor hColumnDescriptor = new HColumnDescriptor(str);
                hTableDescriptor.addFamily(hColumnDescriptor);
            }
            admin.createTable(hTableDescriptor);
            System.out.println("create table success");
        }
        close();
    }
    /**
     * 删除指定表
     * @param tableName 表名
     * @throws IOException
     */
    public static void deleteTable(String tableName) throws IOException {
        init();
        TableName tn = TableName.valueOf(tableName);
        if (admin.tableExists(tn)) {
            admin.disableTable(tn);
            admin.deleteTable(tn);
        }
        close();
    }

    /**
     * 查看已有表
     * @throws IOException
     */
    public static void listTables() throws IOException {
        init();
        HTableDescriptor hTableDescriptors[] = admin.listTables();
        for(HTableDescriptor hTableDescriptor :hTableDescriptors){
            System.out.println(hTableDescriptor.getNameAsString());
        }
        close();
```

```
    }
    /**
     * 向某一行的某一列插入数据
     * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族名
     * @param col 列名（如果其列族下没有子列，此参数可为空）
     * @param val 值
     * @throws IOException
     */
    public static void insertRow(String tableName,String rowKey,String
colFamily,String col,String val) throws IOException {
        init();
        Table table = connection.getTable(TableName.valueOf(tableName));
        Put put = new Put(rowKey.getBytes());
        put.addColumn(colFamily.getBytes(), col.getBytes(), val.getBytes());
        table.put(put);
        table.close();
        close();
    }

    /**
     * 删除数据
     * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族名
     * @param col 列名
     * @throws IOException
     */
    public static void deleteRow(String tableName,String rowKey,String
colFamily,String col) throws IOException {
        init();
        Table table = connection.getTable(TableName.valueOf(tableName));
        Delete delete = new Delete(rowKey.getBytes());
        //删除指定列族的所有数据
        //delete.addFamily(colFamily.getBytes());
        //删除指定列的数据
        //delete.addColumn(colFamily.getBytes(), col.getBytes());

        table.delete(delete);
        table.close();
        close();
    }
    /**
     * 根据行键rowkey查找数据
     * @param tableName 表名
     * @param rowKey 行键
     * @param colFamily 列族名
     * @param col 列名
     * @throws IOException
     */
    public static void getData(String tableName,String rowKey,String colFamily,String
col)throws  IOException{
        init();
        Table table = connection.getTable(TableName.valueOf(tableName));
        Get get = new Get(rowKey.getBytes());
        get.addColumn(colFamily.getBytes(),col.getBytes());
        Result result = table.get(get);
```

```
        showCell(result);
        table.close();
        close();
    }
    /**
     * 格式化输出
     * @param result
     */
    public static void showCell(Result result){
        Cell[] cells = result.rawCells();
        for(Cell cell:cells){
            System.out.println("RowName:"+new String(CellUtil.cloneRow(cell))+" ");
            System.out.println("Timetamp:"+cell.getTimestamp()+" ");
            System.out.println("column Family:"+new
String(CellUtil.cloneFamily(cell))+" ");
            System.out.println("row Name:"+new String(CellUtil.cloneQualifier(cell))+"
");
            System.out.println("value:"+new String(CellUtil.cloneValue(cell))+" ");
        }
    }
}
```