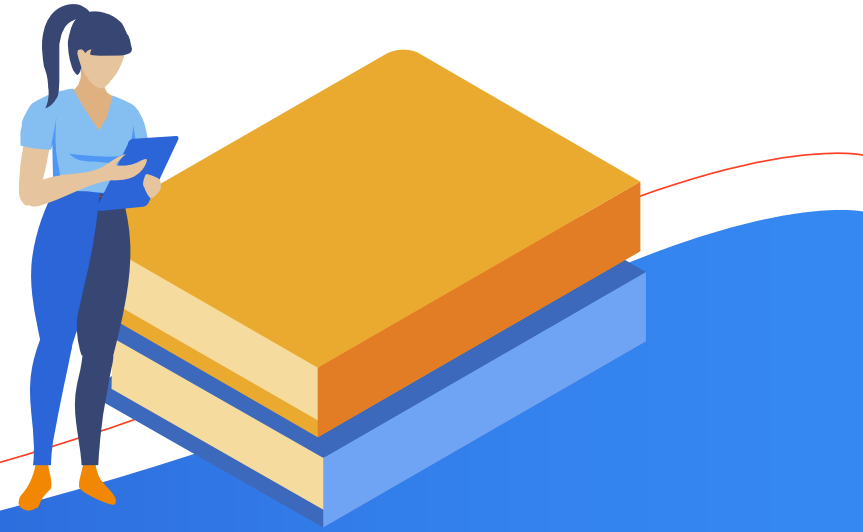
Turing AI 万维图灵 | 大数据系列课程

大数据技术-第三章：HDSF分布式文件系统

实践：编写小文件合并程序以及运行（实验6）

CONTENTS



01
项目目标

1 项目目标

01 掌握HDFS Java API的实现方式

02 掌握小文件合并操作

03 理解小文件合并原理

02
项目描述

2 项目描述

在实际项目中，输入数据往往是由许多小文件组成，这里的小文件是指小于HDFS系统Block大小的文件（默认128M）， 然而每一个存储在HDFS中的文件、目录和块都映射为一个对象，存储在NameNode服务器内存中，通常占用150个字节。 如果有1千万个文件，就需要消耗大约3GB的内存空间。如果是10亿个文件呢，简直不可想象。所以目前很多公司采用的方法就是在数据进入 Hadoop 的 HDFS 系统之前对大量的小文件进行合并，从而节约对NameNode内存空间的占用。

实验操作系统为CentOS-7-x86_64-DVD-1511，Hadoop版本为hadoop-2.6.5，Linux版本JDK为jdk-8u201-linux-x64，Windows版本JDK为jdk-8u181-windows-x64，远程连接工具为SecureCRTPortable，虚拟机工具为VMware-workstation-full-15.5.0-14665864，开发工具为Eclipse Oxygen.2 Release (4.7.2)版本，程序的数据文件为SmallFiles.rar。

本实验所用软件可根据文件名称在资源库内下载。。

03
知识准备

为了完成本实验，学生需要提前掌握以下理论知识内容：

1. Java以及面向对象基础知识。
2. HDFS Java API基础知识。
3. HDFS文件存储原理。

04

项目实施

4.1 实施思路

基于项目描述与知识准备的内容，我们已经对小文件操作知识点有了一定的理解，现在我们学习如何使用HDFS Java API完成小文件合并操作

我们将按照下述三个步骤来完成训练。

1. 准备工作：准备需要合并的小文件数据，新建项目。

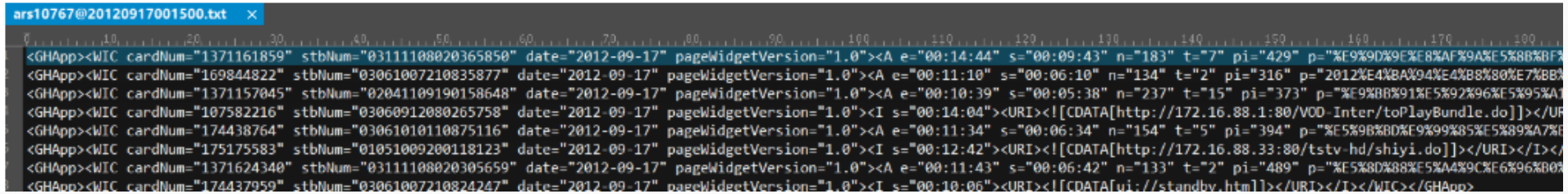
2. 项目完整代码实现：过滤噪音文件、循环获取子目录下所有文件、文件合并上传至HDFS。

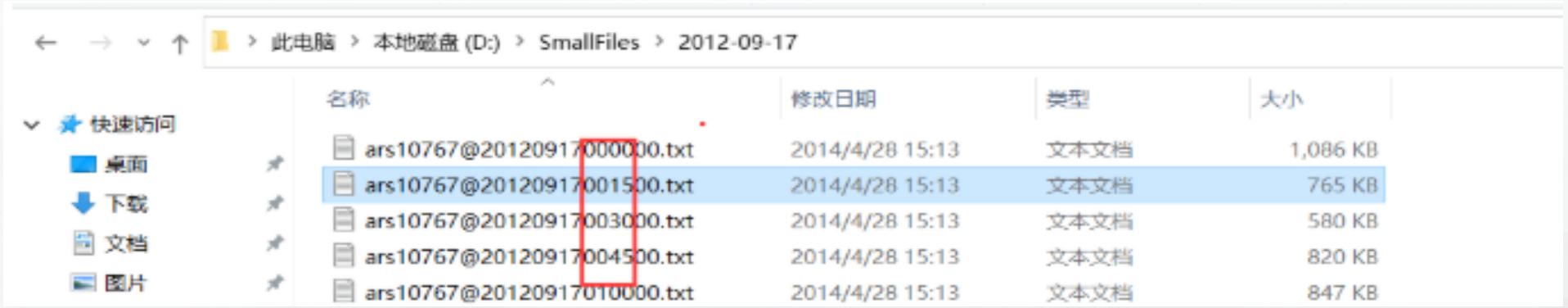
3. 代码运行及结果测试：运行代码并查看校验执行结果。

4.2 实施步骤

步骤一：准备工作：

数据准备，本项目处理的基础数据用户收看网络电视的清单数据，以日期为文件夹，每15分钟数据存方在一个文件内，在每个文件夹内有208个大小为800KB左右的小文件，如下图所示

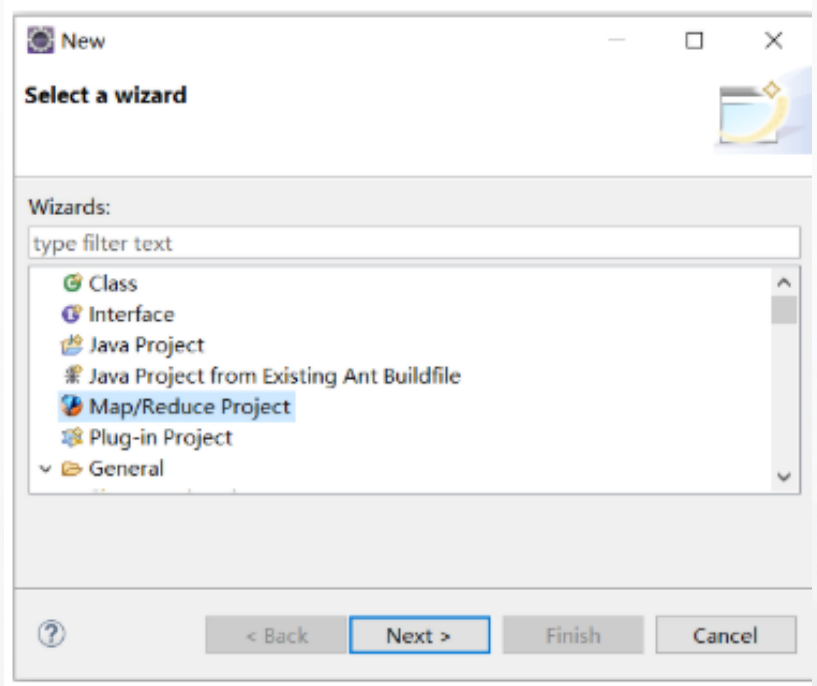


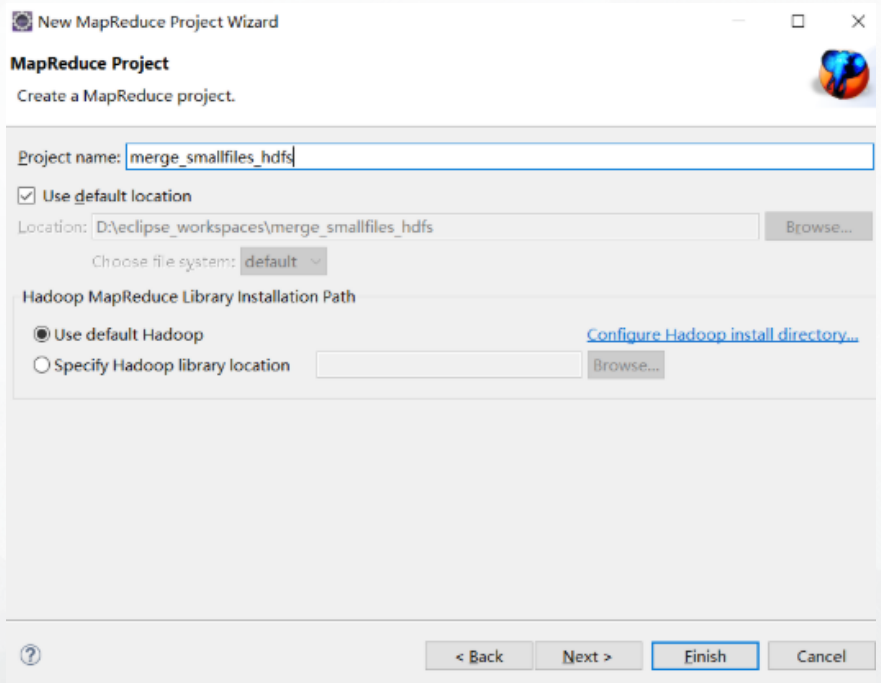


4.2 实施步骤

步骤一：准备工作：

使用项目3中部署完成的Eclipse环境新建Map/Reduce Project项目merge_smallfiles_hdfs，如下图所示

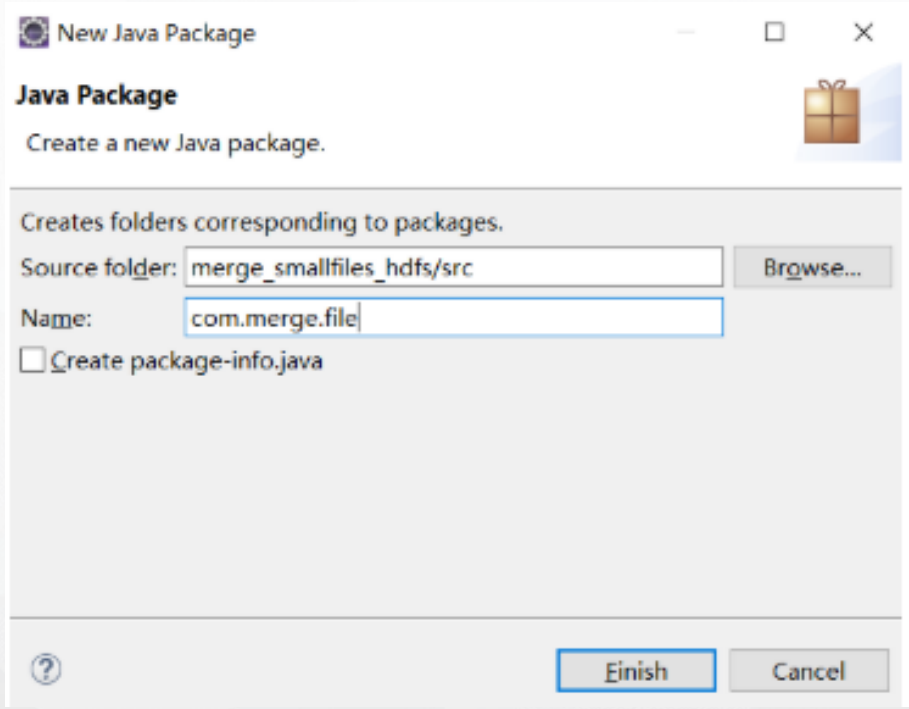




4.2 实施步骤

步骤一：准备工作：

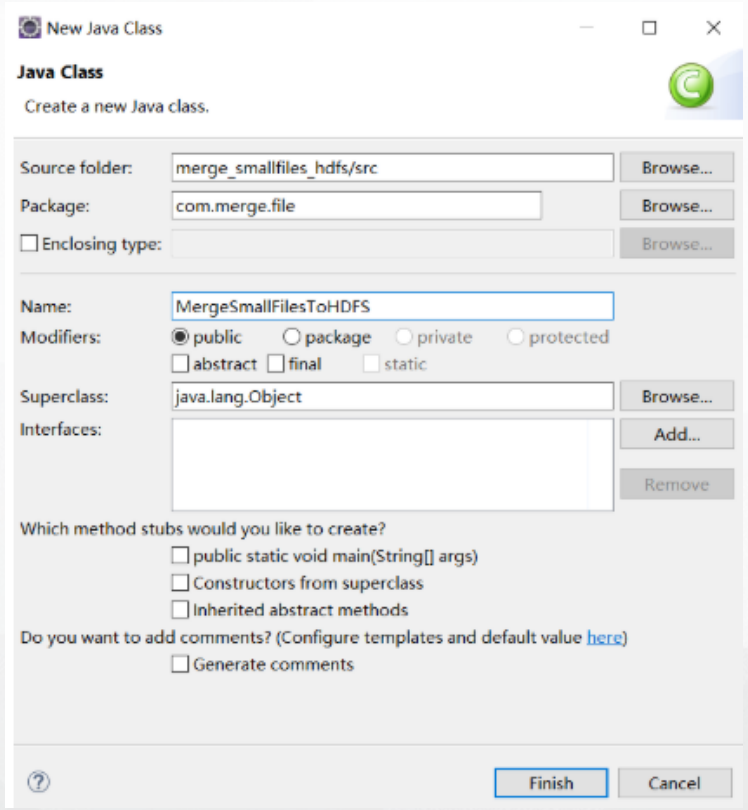
在项目src目录下新建Package，如下图所示



4.2 实施步骤

步骤一：准备工作：

在com.merge.file包内新建MergeSmallFilesToHDFS主类



4.2 实施步骤

步骤二：项目完整代码实现：

在MergeSmallFilesToHDFS编写以下代码合并小文件并上传至HDFS文件系统指定目录，该程序在不影响数据完整性的前提下将每个文件夹的小文件合并成一个以文件夹日期命名的大文件上传至平台，从而达到减少上传文件数量的目的，具体的代码解析参考代码内部注释。

```
/**
 * function 合并小文件至 HDFS
 */
public class MergeSmallFilesToHDFS {
    private static FileSystem fs = null;
    private static FileSystem local = null;
    /**
     * @function main
     * @param args
     * @throws IOException
     * @throws URISyntaxException
     */
    public static void main(String[] args) throws IOException,URISyntaxException {
        list();
    }
```

4.2 实施步骤

步骤二：项目完整代码实现：

```
    public static void list() throws IOException, URISyntaxException {
        // 读取hadoop文件系统的配置
        Configuration conf = new Configuration();
        //文件系统访问接口
        URI uri = new URI("hdfs://192.168.128.133:9000");
        //创建FileSystem对象
        fs = FileSystem.get(uri, conf);
        // 获得本地文件系统
        local = FileSystem.getLocal(conf);
        //过滤目录下的 svn 文件
        FileStatus[] dirstatus = local.globStatus(new Path("D://SmallFiles/*"),new
RegexExcludePathFilter("^.*svn$"));
        //获取SmallFiles目录下的所有文件路径
        Path[] dirs = FileUtil.stat2Paths(dirstatus);
        FSDataOutputStream out = null;
        FSDataInputStream in = null;
        for (Path dir : dirs) {
            //2012-09-17
            String fileName = dir.getName().replace("-", "");//文件名称
            //只接受日期目录下的.txt文件
            FileStatus[] localStatus = local.globStatus(new Path(dir+"/*"),new
RegexAcceptPathFilter("^.*txt$"));
            // 获得日期目录下的所有文件
            Path[] listedPaths = FileUtil.stat2Paths(localStatus);
            //输出路径
            Path block = new Path("hdfs://192.168.128.133:9000/tv/"+ fileName + ".txt");
            System.out.println("合并后的文件名称："+fileName+".txt");
            // 打开输出流
            out = fs.create(block);
            for (Path p : listedPaths) {
                in = local.open(p);// 打开输入流
                IOUtils.copyBytes(in, out, 4096, false); // 复制数据
                // 关闭输入流
                in.close();
            }
```

4.2 实施步骤

步骤二：项目完整代码实现：

过滤 regex 格式的文件

```
/**
 *
 * @function 过滤 regex 格式的文件
 *
 */
public static class RegexExcludePathFilter implements PathFilter {
    private final String regex;

    public RegexExcludePathFilter(String regex) {
        this.regex = regex;
    }

    @Override
    public boolean accept(Path path) {
        // TODO Auto-generated method stub
        boolean flag = path.toString().matches(regex);
        return !flag;
    }

}
```

4.2 实施步骤

步骤二：项目完整代码实现：

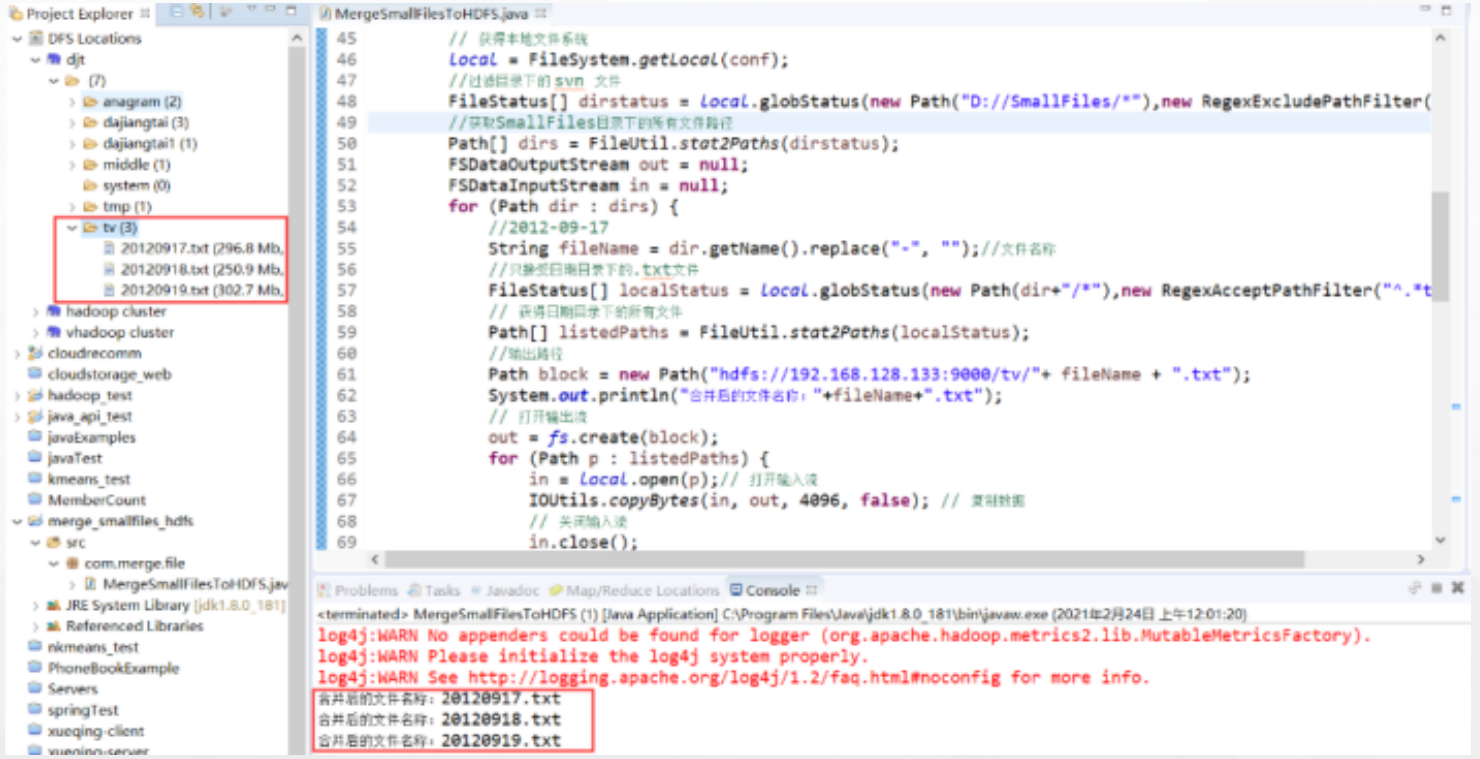
接受 regex 格式的文件

```
/**
 *
 * @function 接受 regex 格式的文件
 *
 */
public static class RegexAcceptPathFilter implements PathFilter {
    private final String regex;

    public RegexAcceptPathFilter(String regex) {
        this.regex = regex;
    }

    @Override
    public boolean accept(Path path) {
        // TODO Auto-generated method stub
        boolean flag = path.toString().matches(regex);
        return flag;
    }

}
```

4.2 实施步骤

步骤三：代码运行及结果测试：

执行程序，运行结果如下图表示文件合并上传成功，从结果可以看到，合并后文件数量为3个，大小为300MB左右，合并的目的就是为了提高Hadoop平台的执行效率。



05
项目拓展

知识点拓展

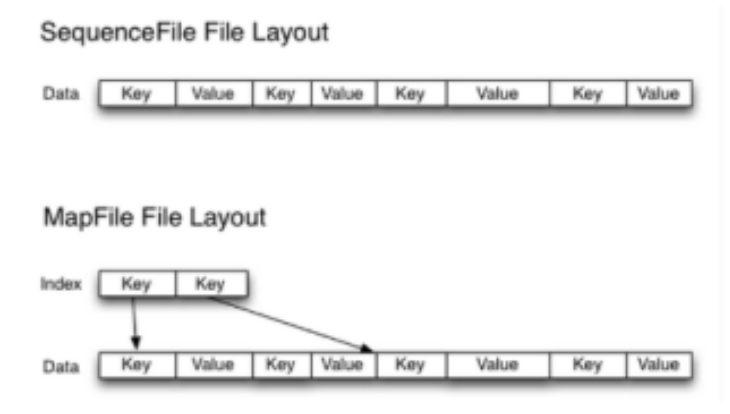
5 知识拓展

在前面的任务中，我们利学习了如何利用Hadoop提供的API接口合并小文件解决以此所带来的问题，而本实验内的只是介绍了其中一种解决小文件问题的方法对于在实际生产环境内可能会出现其他情况。接下来让我们来介绍解决Hadoop小文件存储的其他方法（拓展知识点）。

1.Sequence file

Sequence file由一系列的二进制key/value组成，如果为key小文件名，value为文件内容，则可以将大批小文件合并成一个大文件。

Hadoop-0.21.0中提供了SequenceFile，包括Writer，Reader和SequenceFileSorter类进行写，读和排序操作。创建sequence file的过程可以使用mapreduce工作方式完成，对于index，需要改进查找算法。



5 知识拓展

2.Hadoop Archive

Hadoop Archive或者HAR，是一个高效地将小文件放入HDFS块中的文件存档工具，它能够将多个小文件打包成一个HAR文件，这样在减少namenode内存使用的同时，仍然允许对文件进行透明的访问。

3.CombineFileInputFormat

CombineFileInputFormat是一种新的inputformat，用于将多个文件合并成一个单独的split，另外，它会考虑数据的存储位置。

06

项目实训

课后练习

6.1 实训习题

- 简答题：简述Hadoop平台小文件数量过多导致哪些问题？

Turing AI 万维图灵 | 大数据系列课程

大数据

BIG DATA

智 / 能 / 科 / 技　　放 / 眼 / 未 / 来