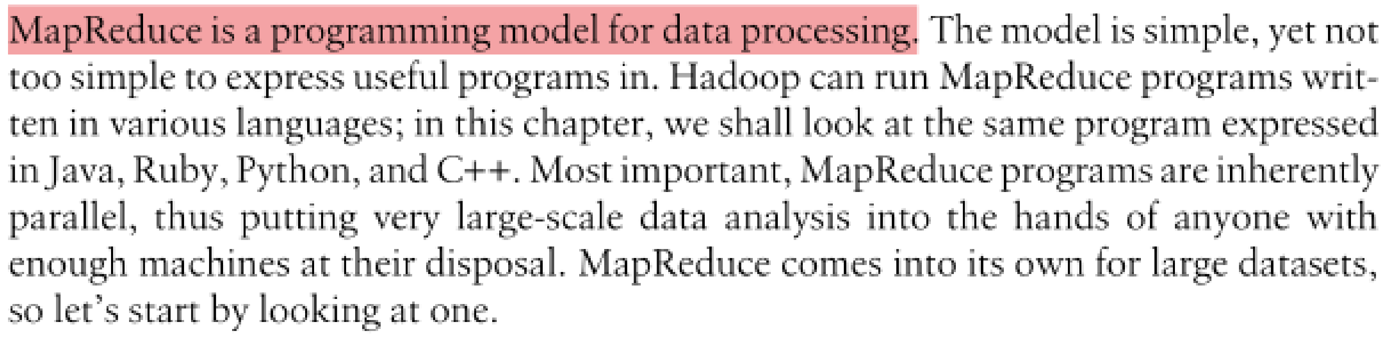
MapReduce
并行计算框架简介

MapReduece是一个并行计算框架

MapReduce采用了“分而治之”的思想来并行化处理大量数据。
在编程过程中，只需要实现Map和Reduce这两个函数即可。


MapReduce is a programming model for data processing. The model is simple, yet not too simple to express useful programs in. Hadoop can run MapReduce programs written in various languages; in this chapter, we shall look at the same program expressed in Java, Ruby, Python, and C++. Most important, MapReduce programs are inherently parallel, thus putting very large-scale data analysis into the hands of anyone with enough machines at their disposal. MapReduce comes into its own for large datasets, so let’s start by looking at one.

MapReduece思想借鉴于于Lisp语言

Lisp中的Map: (map’vector#+#(1 2 3 4) # (4 3 2 1))

Lisp中的Reduce: (reduce#’+#(1 2 3 4))

A Weather Dataset

For our example, we will write a program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi-structured and record-oriented.

半结构化数据
面向记录

Example 2-1. Format of a National Climate Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

A Weather Dataset

For our example, we will write a program that mines weather data. Weather sensors collecting data every hour at many locations across the globe gather a large volume of log data, which is a good candidate for analysis with MapReduce, since it is semi-structured and record-oriented.

半结构化数据
面向记录

Example 2-1. Format of a National Climate Data Center record

```
0057
332130   # USAF weather station identifier
99999    # WBAN weather station identifier
19500101 # observation date
0300     # observation time
4
+51317   # latitude (degrees x 1000)
+028783  # longitude (degrees x 1000)
FM-12
+0171    # elevation (meters)
99999
V020
320      # wind direction (degrees)
1        # quality code
N
0072
1
00450    # sky ceiling height (meters)
1        # quality code
C
N
010000   # visibility distance (meters)
1        # quality code
N
9
-0128    # air temperature (degrees Celsius x 10)
1        # quality code
-0139    # dew point temperature (degrees Celsius x 10)
1        # quality code
10268    # atmospheric pressure (hectopascals x 10)
1        # quality code
```

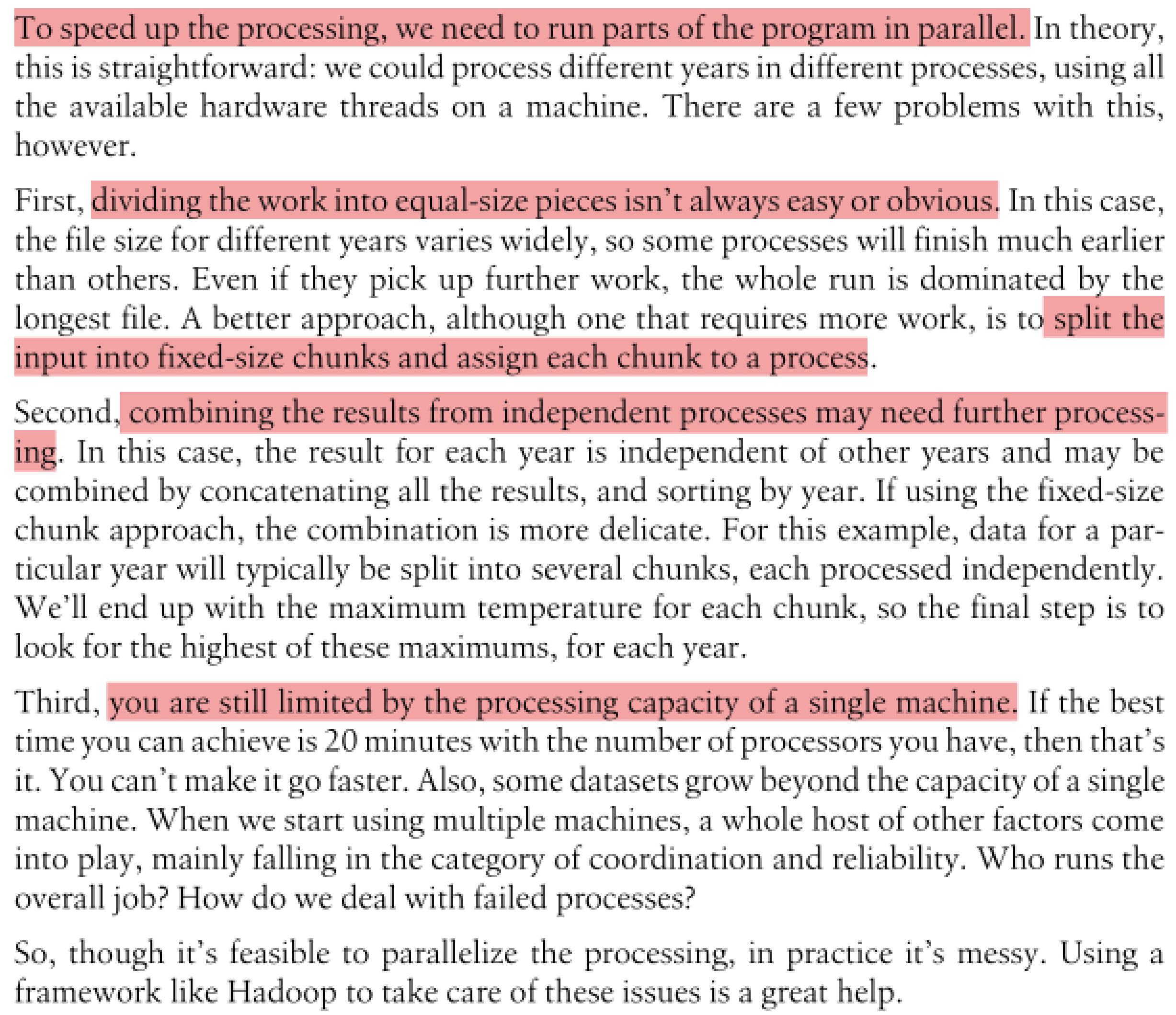
Example 2-2. A program for finding the maximum recorded temperature by year from NCDC weather records

```
#!/usr/bin/env bash
for year in all/*
do
  echo -ne `basename $year .gz`"\t"
  gunzip -c $year | \
    awk '{ temp = substr($0, 88, 5) + 0;
           q = substr($0, 93, 1);
           if (temp !=9999 && q ~ /[01459]/ && temp > max) max = temp }
         END { print max }'
done
```

Unix脚本分析程序
串行计算，42分钟

如何提速？ How to improve?

Parallel 并行计算



To speed up the processing, we need to run parts of the program in parallel. In theory, this is straightforward: we could process different years in different processes, using all the available hardware threads on a machine. There are a few problems with this, however.

First, dividing the work into equal-size pieces isn't always easy or obvious. In this case, the file size for different years varies widely, so some processes will finish much earlier than others. Even if they pick up further work, the whole run is dominated by the longest file. A better approach, although one that requires more work, is to split the input into fixed-size chunks and assign each chunk to a process.

Second, combining the results from independent processes may need further processing. In this case, the result for each year is independent of other years and may be combined by concatenating all the results, and sorting by year. If using the fixed-size chunk approach, the combination is more delicate. For this example, data for a particular year will typically be split into several chunks, each processed independently. We'll end up with the maximum temperature for each chunk, so the final step is to look for the highest of these maximums, for each year.

Third, you are still limited by the processing capacity of a single machine. If the best time you can achieve is 20 minutes with the number of processors you have, then that's it. You can't make it go faster. Also, some datasets grow beyond the capacity of a single machine. When we start using multiple machines, a whole host of other factors come into play, mainly falling in the category of coordination and reliability. Who runs the overall job? How do we deal with failed processes?

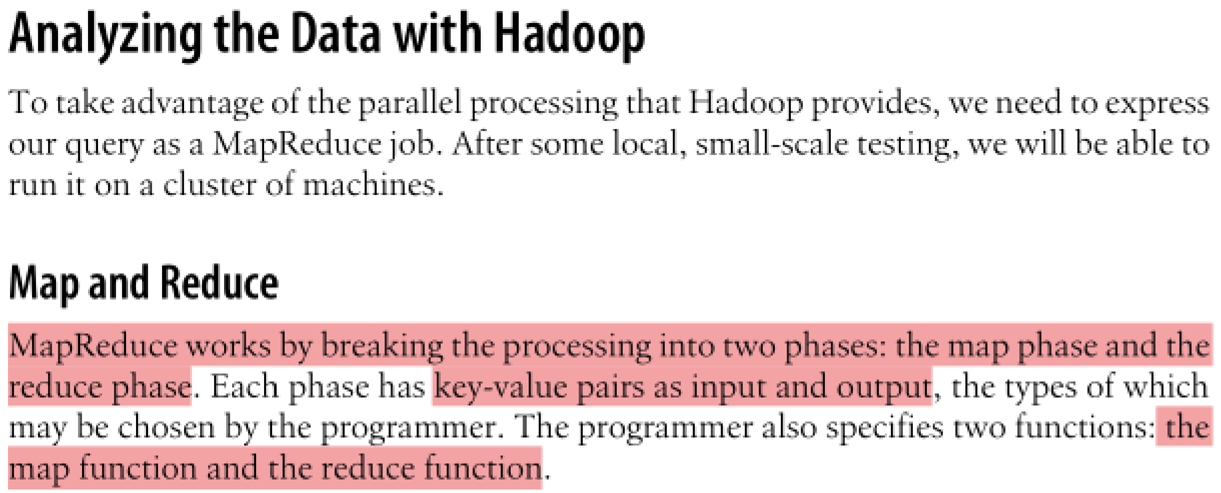
So, though it's feasible to parallelize the processing, in practice it's messy. Using a framework like Hadoop to take care of these issues is a great help.

Map and Reduce

MapReduce将进程分成两个阶段

每个阶段都有一个键值对作为输入和输出。

编程的时候需要指定两个函数，分别是map函数和reduce函数。



Analyzing the Data with Hadoop

To take advantage of the parallel processing that Hadoop provides, we need to express our query as a MapReduce job. After some local, small-scale testing, we will be able to run it on a cluster of machines.

Map and Reduce

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function.

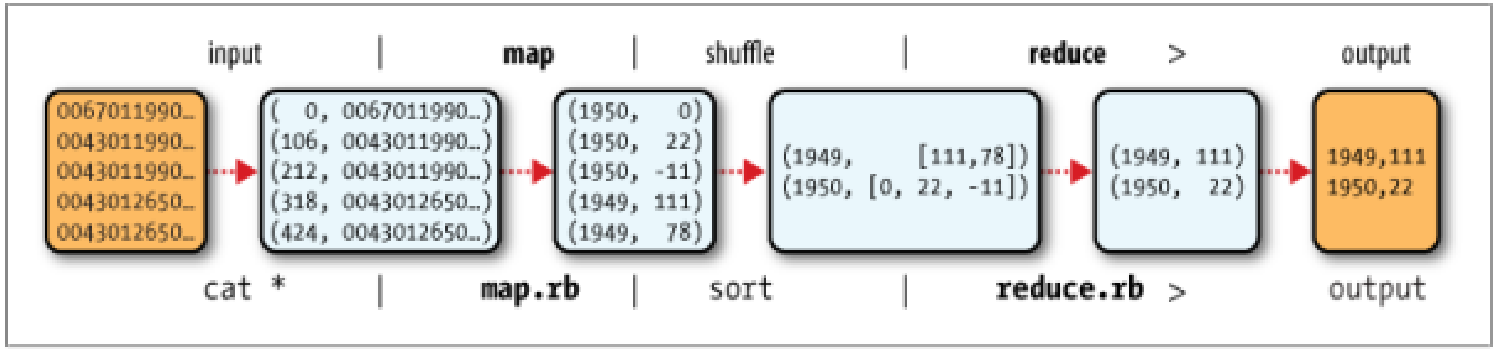
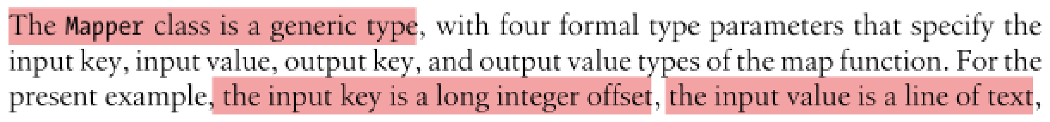


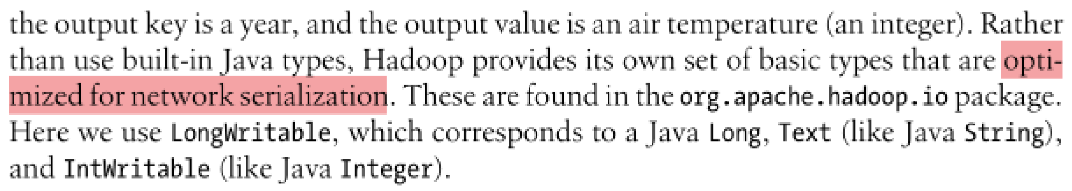
Figure 2-1. MapReduce logical data flow

MapReduce过程

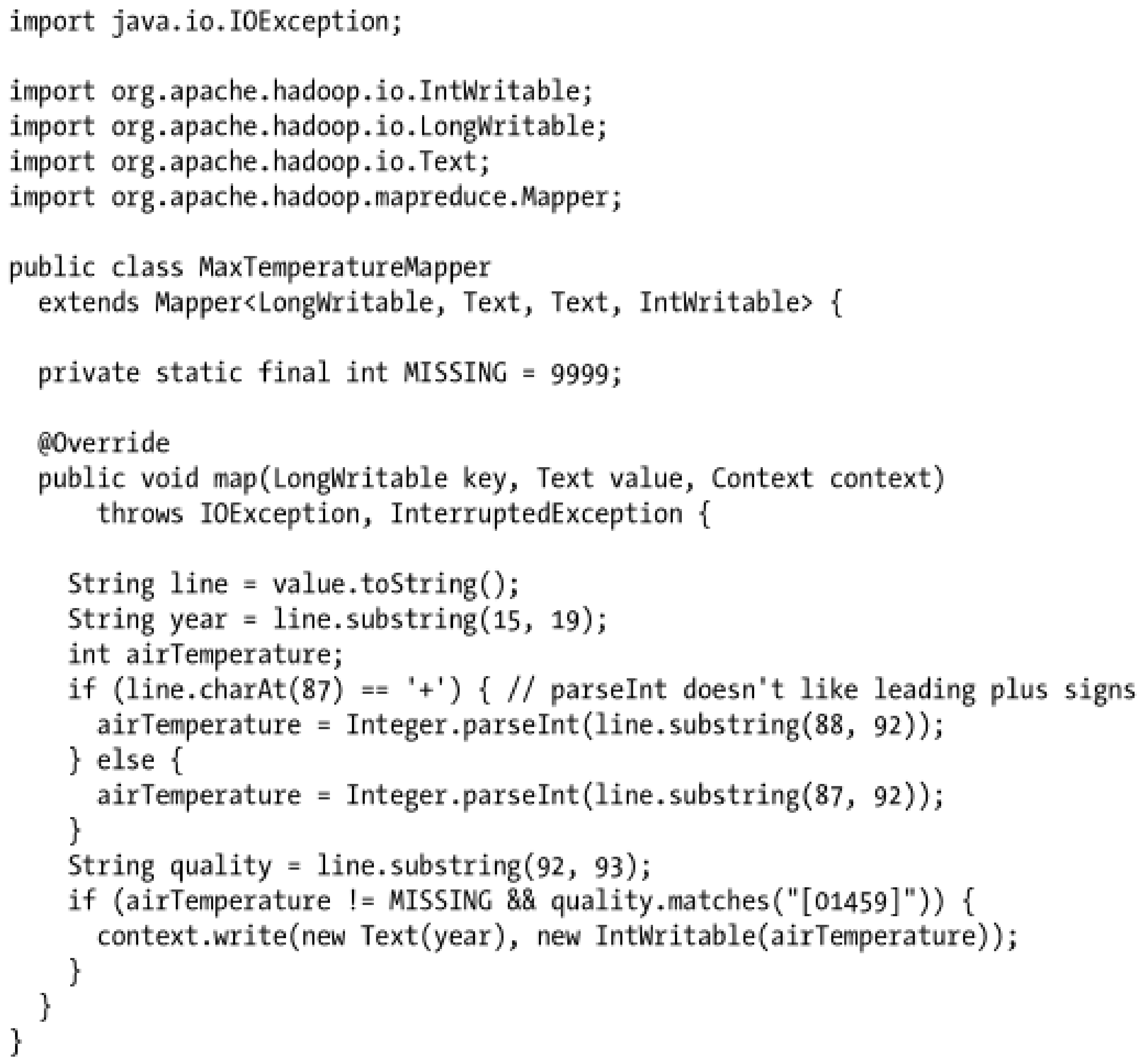
Map类的书写


The Mapper class is a generic type, with four formal type parameters that specify the input key, input value, output key, and output value types of the map function. For the present example, the input key is a long integer offset, the input value is a line of text,

Mapper类是一个基类，这里继承了Mapper类

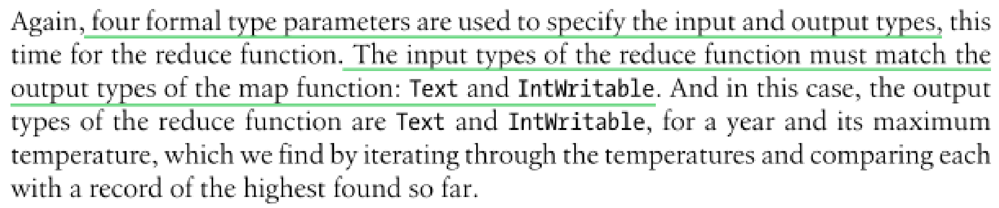

the output key is a year, and the output value is an air temperature (an integer). Rather than use built-in Java types, Hadoop provides its own set of basic types that are optimized for network serialization. These are found in the org.apache.hadoop.io package. Here we use LongWritable, which corresponds to a Java Long, Text (like Java String), and IntWritable (like Java Integer).

这里用到的不是Java中的内建类型，而是被封装了的类型。
这些类型优化了网络传输的序列化。


```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class MaxTemperatureMapper
  extends Mapper<LongWritable, Text, Text, IntWritable> {

  private static final int MISSING = 9999;

  @Override
  public void map(LongWritable key, Text value, Context context)
      throws IOException, InterruptedException {

    String line = value.toString();
    String year = line.substring(15, 19);
    int airTemperature;
    if (line.charAt(87) == '+') { // parseInt doesn't like leading plus signs
      airTemperature = Integer.parseInt(line.substring(88, 92));
    } else {
      airTemperature = Integer.parseInt(line.substring(87, 92));
    }
    String quality = line.substring(92, 93);
    if (airTemperature != MISSING && quality.matches("[01459]")) {
      context.write(new Text(year), new IntWritable(airTemperature));
    }
  }
}
```

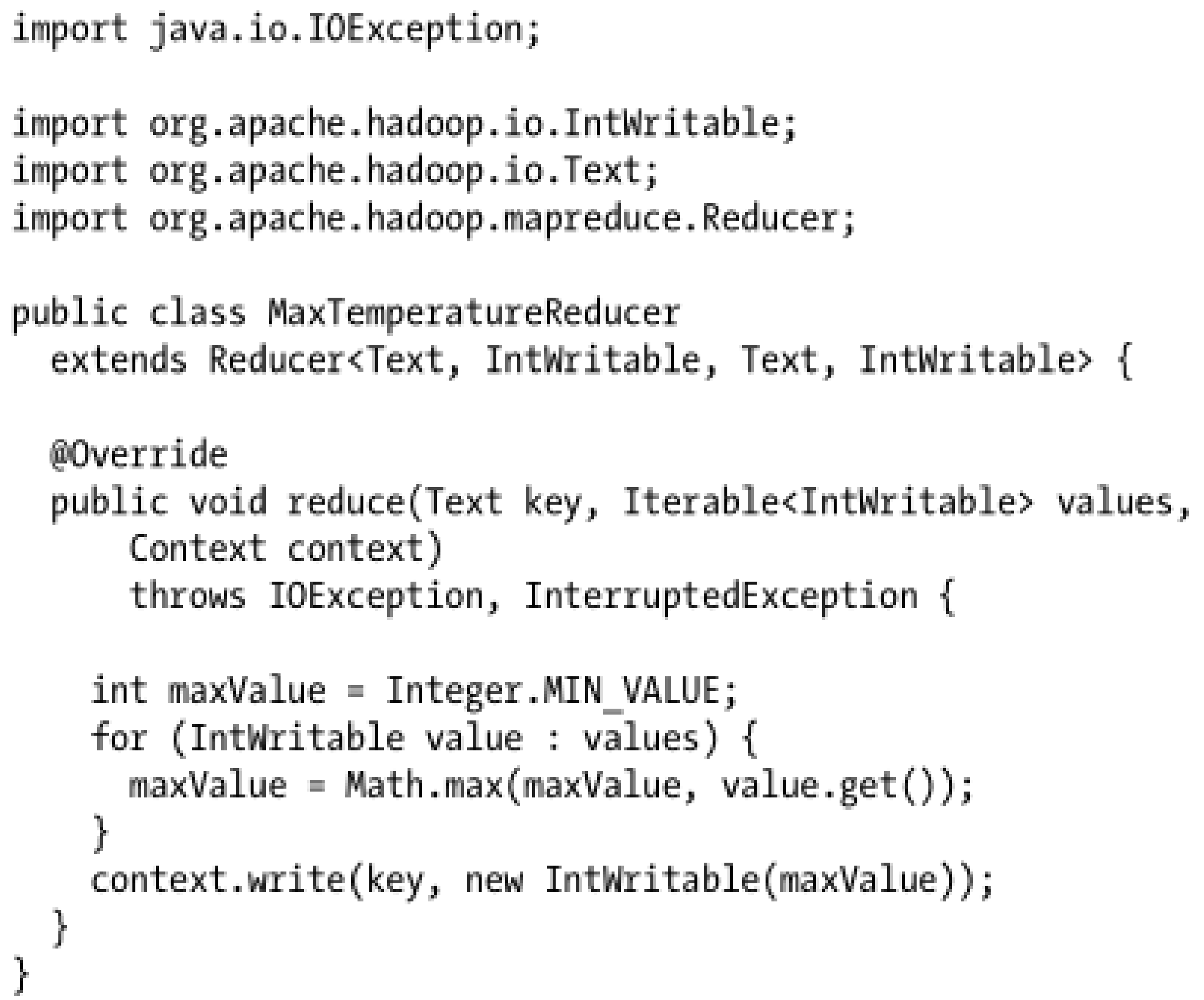
Reduce类的书写

同样集成Reducer类，重写reduce方法。



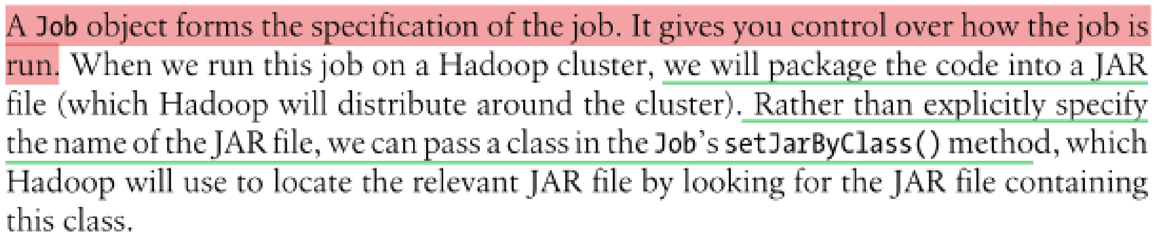
Again, four formal type parameters are used to specify the input and output types, this time for the reduce function. The input types of the reduce function must match the output types of the map function: `Text` and `IntWritable`. And in this case, the output types of the reduce function are `Text` and `IntWritable`, for a year and its maximum temperature, which we find by iterating through the temperatures and comparing each with a record of the highest found so far.

四个参数是用来指定输入输出类型的。

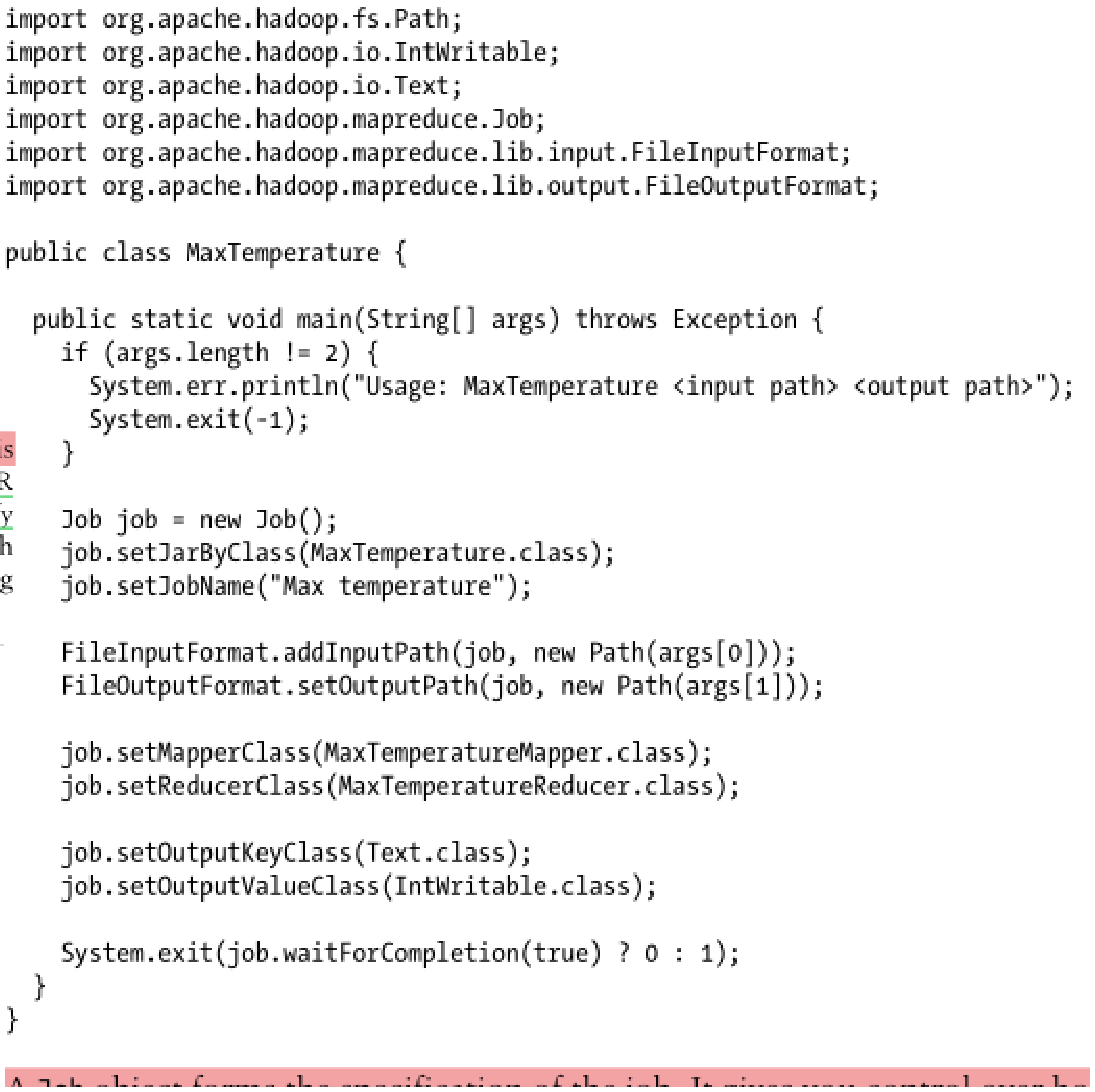


```java
import java.io.IOException;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class MaxTemperatureReducer
  extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values,
      Context context)
      throws IOException, InterruptedException {

    int maxValue = Integer.MIN_VALUE;
    for (IntWritable value : values) {
      maxValue = Math.max(maxValue, value.get());
    }
    context.write(key, new IntWritable(maxValue));
  }
}
```

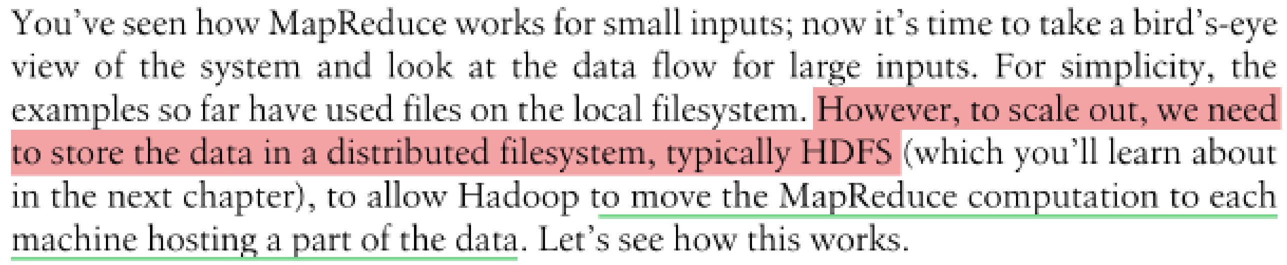
主体Job类的书写

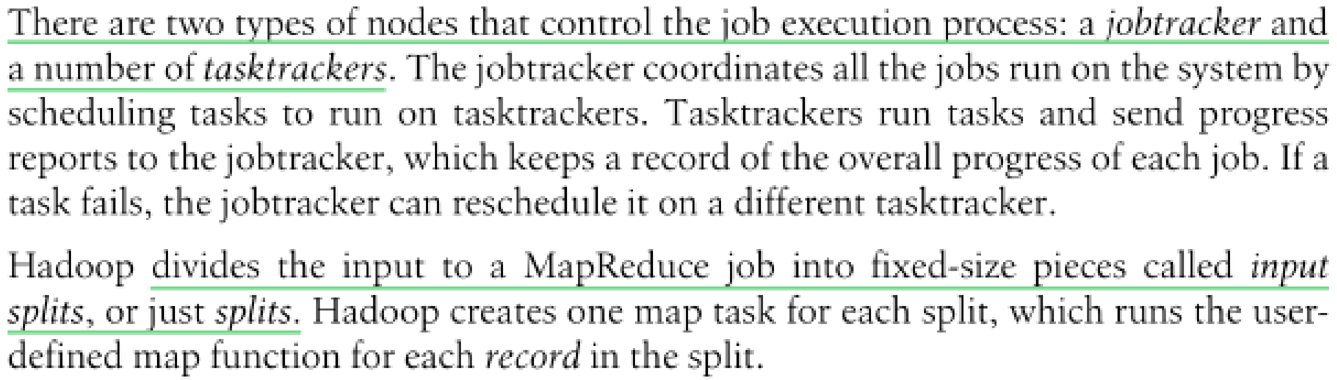

A Job object forms the specification of the job. It gives you control over how the job is run. When we run this job on a Hadoop cluster, we will package the code into a JAR file (which Hadoop will distribute around the cluster). Rather than explicitly specify the name of the JAR file, we can pass a class in the Job's setJarByClass() method, which Hadoop will use to locate the relevant JAR file by looking for the JAR file containing this class.

通过Job类来导出Jar包，main函数写在job类里。


```java
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class MaxTemperature {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperature <input path> <output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);
    job.setJobName("Max temperature");

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setReducerClass(MaxTemperatureReducer.class);

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

扩展至HDFS

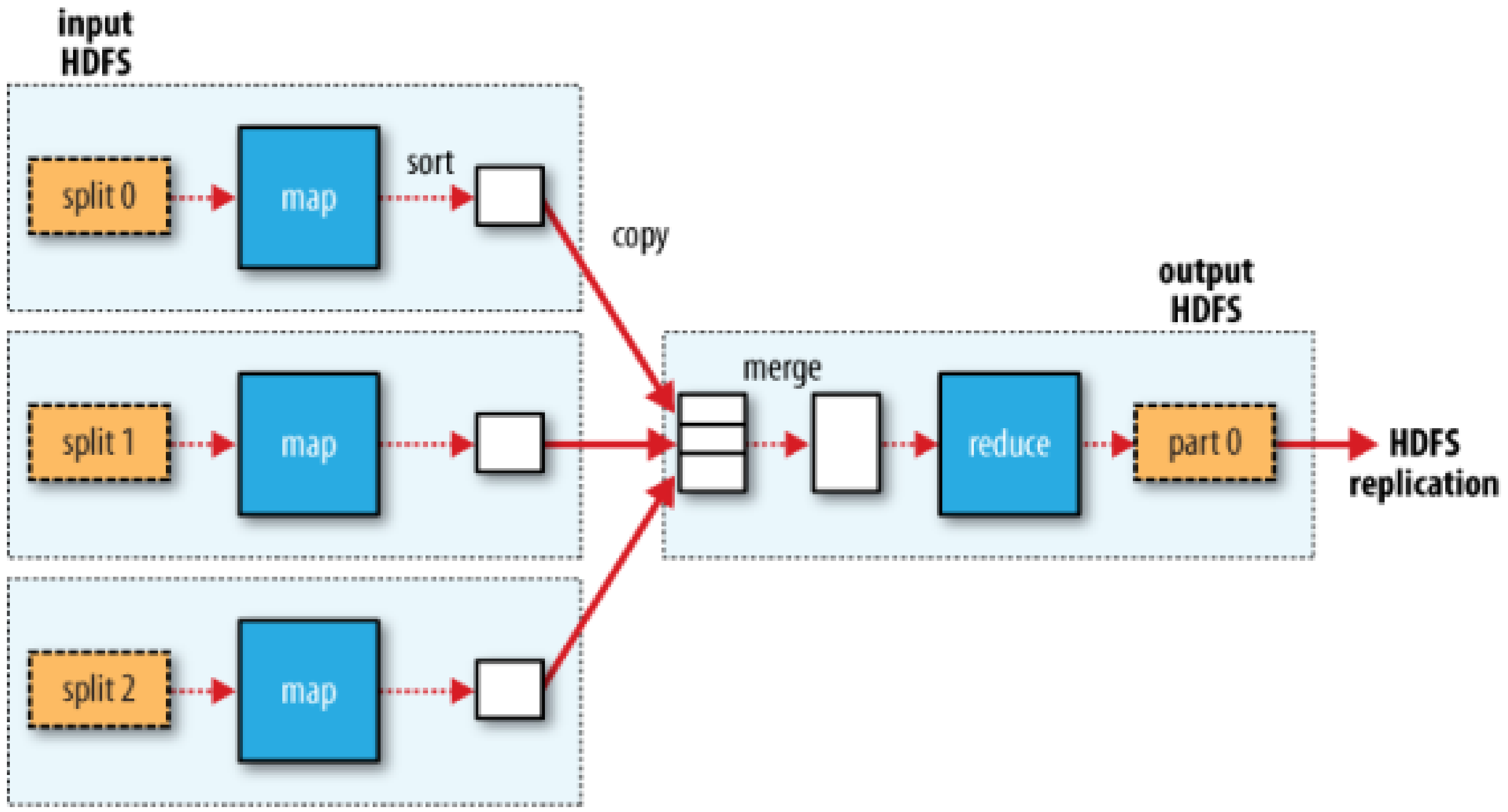

You’ve seen how MapReduce works for small inputs; now it’s time to take a bird’s-eye view of the system and look at the data flow for large inputs. For simplicity, the examples so far have used files on the local filesystem. However, to scale out, we need to store the data in a distributed filesystem, typically HDFS (which you’ll learn about in the next chapter), to allow Hadoop to move the MapReduce computation to each machine hosting a part of the data. Let’s see how this works.


There are two types of nodes that control the job execution process: a *jobtracker* and a number of *tasktrackers*. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a task fails, the jobtracker can reschedule it on a different tasktracker.

Hadoop divides the input to a MapReduce job into fixed-size pieces called *input splits*, or just *splits*. Hadoop creates one map task for each split, which runs the user-defined map function for each *record* in the split.

有两种类型的节点来控制工作的执行流程：
Jobtracker作为主节点。
TaskTracker管理每个计算节点上计算任务。

数据流



The number of reduce tasks is not governed by the size of the input, but is specified independently. In “The Default MapReduce Job” on page 225, you will see how to choose the number of reduce tasks for a given job.

Combiner Function

The combiner function doesn't replace the reduce function. (How could it? The reduce function is still needed to process records with the same key from different maps.) But

但是Combiner函数是来取代reduce函数的
Combin可以提高效率

```
(1950, 0)
(1950, 20)
(1950, 10)
```

And the second produced:

```
(1950, 25)
(1950, 15)
```

The reduce function would be called with a list of all the values:

```
(1950, [0, 20, 10, 25, 15])
```

with output:

```
(1950, 25)
```

Not all functions possess this property.[4] For example, if we were calculating mean temperatures, then we couldn't use the mean as our combiner function, since:

```
mean(0, 20, 10, 25, 15) = 14
```

but:

```
mean(mean(0, 20, 10), mean(25, 15)) = mean(10, 20) = 15
```