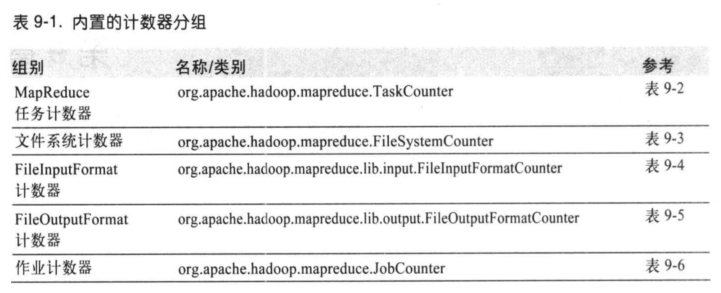
# 3.1 计数器

1.**计数器是收集作业统计信息的有效手段之一**，用于**质量控制或应用级统计**。计数器还可辅助诊断系统故障。**如果需要将日志信息传输到map或reduce任务，更好的方法通常是看能否用一个计数器值来记录某一特定事件的发生**。对于大型分布式作业而言，使用计数器更为方便。除了因为获取计数器值比输出日志更方便，还有根据计数器值统计特定事件的发生次数要比分析一堆日志文件容易得多。

## 3.1.1 内置计数器

1.**Hadoop 为每个作业维护若干内置计数器**，以**描述多项指标**。例如，某些计数器记录已处理的字节数和记录数，使用户可监控已处理的输入数据量和已产生的输出数据量。

2.这些内置计数器被划分为若干个组，如下表。



3.**各组要么包含任务计数器**(在**任务处理过程中不断更新**)，**要么包含作业计数器**(在作**业处理过程中不断更新**)。

### 任务计数器

1.在**任务执行过程中，任务计数器采集任务的相关信息**，**每个作业的所有任务的结果会被聚集起来**。例如，**MAP\_INPUT\_RECORDS计数器统计每个map任务输入记录的总数**，**并在一个作业的所有map任务上进行聚集**，使得**最终数字是整个作业的所有输入记录的总数**。

2.**任务计数器由其关联任务维护**，并**定期发送给application master**。因此，**计数器能够被全局地聚集**。**任务计数器的值每次都是完整传输的**，**而非传输自上次传输之后的计数值，从而避免由于消息丢失而引发的错误**。另外，**如果一个任务在作业执行期间失败，则相关计数器的值会减小**。

3.**虽然只有当整个作业执行完之后计数器的值才是完整可靠的**，**但是部分计数器仍然可以在任务处理过程中提供一些有用的诊断信息**，以便由 Web界面监控。例如，PHYSICAL\_MEMORY\_BYTES、VIRTUAL\_MEMORY\_BYTES和COMMITTED\_HEAP\_BYTES**计数器显示特定任务执行过程中的内存使用变化情况**。

4.内置的任务计数器包括再MapReduce任务计数器分组中的计数器以及在文件相关的计数器分组中的计数器。（详见Hadoop权威指南第四版Page269）

### 作业计数器

1.**作业计数器由application master维护**，**因此无需在网络间传输数据**，这一点与包括“用户定义的计数器”在内的其他计数器不同。**这些计数器都是作业级别的统计量**，**其值不会随着任务运行而改变**。例如，TOTAL\_LAUNCHED\_MAPS统计在作业执行过程中启动的map任务数，包括失败的map任务。（详见Hadoop权威指南第四版Page271）

# 3.2 排序（重点）

## 3.2.1 部分排序

1.默认情况下，MapReduce根据输入记录的键对数据集排序。键的排列顺序是由RawComparator控制的：

1. 若属性mapreduce.job.output.key.comparator.class已经显式设置，或者通过Job类的setSortComparatorClass()方法进行设置，则使用该类的实例。
2. 否则,键必须是WritableComparable的子类,并使用针对该键类的已登记的comparator。
3. 如果还没有已登记的comparator，则使用 RawComparator。RawComparator将字节流反序列化为一个对象，再由WritableComparable的compareTo()方法进行操作。

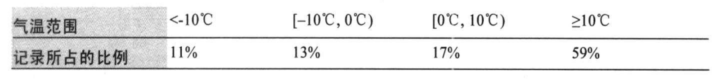
2.假设采用30个reducer来运行该程序，该指令产生30个已排序的输出文件。但是如何将这些小文件合并成一个有序的文件并非易事。例如，直接将纯文本文件连接起来无法保证全局有序。

## 3.2.2 全排序

1.使用MapReduce产生一个全局排序文件的最简单方法是使用一个分区。但在处理大型文件时效率很低，因为一台机器必须处理所有输出文件，从而完全丧失了MapReduce所提供的并行架构优势。

2.事实上仍有替代方案：**首先，创建一系列排好序的文件**；**其次，串联这些文件**；**最后，生成一个全局排序的文件**。**主要的思路是使用一个partitioner来描述输出的全局排序**。例如，可以为上述文件创建4个分区，在第一分区中，各记录的气温小于-10℃，第二分区的气温介于-10℃和0℃之间，第三个分区的气温在0℃和10℃之间，最后一个分区的气温大于10℃。

3.此方法的关键点是在于如何划分各个分区。理性情况下，各个分区所含记录数应该大致相等，使作业的总体执行时间不会受制于个别reducer。若前面分区方案相对大小如下：



会发现记录并没有均匀划分。只有深入了解整个数据集的气温分布才能建立更均匀的分区。

4.写一个MapReduce作业来计算落入各个气温桶的记录数，并不困难。

5.获得气温分布信息意味着可以建立一系列分布非常均匀的分区。但由于该操作需要遍历整个数据集，因此并不实用。**通过对键空间进行采样，就可较为均匀地划分数据集。采样的核心思想是只查看一小部分键，获得键的近似分布，并由此构建分区**。幸运的是，Hadoop已经内置若干**采样器**，不需要用户自己写。

## 3.2.3 辅助排序

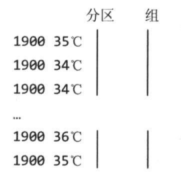
1.**MapReduce框架在记录到达reducer之前按键对记录排序**，**但键所对应的值并没有排序**。甚至在不同的执行轮次中，**这些值的排序也不固定，因为它们来自不同的map任务且这些map任务在不同轮次中的完成时间各不相同**。一般来说，大多数MapReduce程序会避免让reduce函数依赖于值的排序。但是，有时也需要通过特定的方法对键进行排序和分组等以实现对值的排序。

2.例如，**考虑如何设计一个MapReduce程序以计算每年的最高气温**。如果全部记录均按照气温降序排列，则无需遍历整个数据集即可获得查询结果——获取各年份的首条记录并忽略剩余记录。尽管该方法并不是最佳方案，但演示了辅助排序的工作机理。

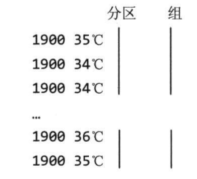
3.为此，**首先构建一个同时包含年份和气温信息的组合键，然后对键值先按年份升序排序，再按气温降序排列**：

1900 35℃1908 34℃1908 34℃...1901 36℃1901 35℃

4.如果仅仅是使用组合键的话，并没有太大的帮助，因为这会导致同一年的记录可能有不同的键，通常这种情况下记录并不会被送到同一个reducer中。例如，(1900，35℃)和(1900，34℃)就可能被送到不同的reducer中。**通过设置一个按照键的年份进行分区的 patitioner，可以确保同一年的记录会被发送到同一个reducer中**。但是，这样做还不够。因为 partitioner只保证每一个reducer接受一个年份的所有记录，而在一个分区之内，reducer仍是通过键进行分组的分区:



5.该问题的**最终解决方案是进行分组设置**。如果reducer中的值按照键的年份进行分组，则一个reducer组将包括同一年份的所有记录。**鉴于这些记录已经按气温降序排列，所以各组的首条记录就是这一年的最高气温**：



6.下面按值排序的方法做一个总结：

* 定义包括自然键和自然值得组合键
* 根据组合键对记录进行排序，则同时用自然键和自然值进行排序
* 针对组合键进行分区和分组时均只考虑自然键

# 3.3 连接（重点）

1.**MapReduce连接分为两种，如果连接操作由mapper执行，则成为“map端连接”**；**如果由reducer执行，则成为”reduce端连接”**。

2.**如果两个数据集的规模均很大**，**以至于没有哪个数据集可以被完全复制到集群的每个节点**，我们**仍然可以使用MapReduce来进行连接**，至于到底采用**map端连接还是reduce端连接**，**则取决于数据的组织方式**。最常见的一个例子便是用户数据库和用户活动日志(例如访问日志)。对于一个热门服务来说，将用户数据库(或日志)分发到所有MapReduce节点中是行不通的。

## 3.3.1 Map端连接

1.在**两个大规模输入数据集之间的map端连接会在数据到达map函数之前就执行连接操作**。为达到该目的，各**map的输入数据必须先分区并且以特定方式排序**。**各个输入数据集被划分成相同数量的分区，并且均按相同的键(连接键)排序**。**同一键的所有记录均会放在同一分区之中**。听起来似乎要求非常严格(的确如此)，但这的确合乎MapReduce 作业的输出。

2.**map端连接操作可以连接多个作业的输出**，**只要这些作业的reducer 数量相同**、**键相同并且输出文件是不可切分的**(例如，借助于小于一个HDFS 块、或进行gzip 压缩来实现)。在天气例子中，如果气象站文件以气象站ID部分排序，记录文件也以气象站ID部分排序，而且reducer 的数量相同，则就满足了执行map端连接的前提条件。

## 3.3.2 Reduce端连接

1.**由于reduce端连接并不要求输入数据集符合特定结构**，**因而reduce端连接比map端连接更为常用**。但是，**由于两个数据集均需经过MapReduce的shuffle过程**，**所以reduce端连接的效率往往要低一些**。**基本思路是mapper为各个记录标记源**，**并且使用连接键作为map输出键**，**使键相同的记录放在同一个reducer中**。以下技术能帮助实现reduce端连接。

2.多输入：**数据集得输入源往往有多种格式**，因**此可以使用MultipleInputs类来方便地解析和标注各个源**

3.辅助排序：如前所述，**reducer将从两个源中选出键相同的记录**，**但这些记录不保证是经过排序的**。然而，为了更好地执行连接操作，**一个源的数据排列在另一个源的数据前是非常重要的**。以天气数据连接为例，对应每个键，气象站记录的值必须是最先看到的，这样reducer能够将气象站名称填到天气记录之中再马上输出。虽然也可以不指定数据传输次序，并将待处理的记录缓存在内存之中，但应该尽量避免这种情况，因为其中任何一组的记录数量可能非常庞大，远远超出reducer的可用内存容量。

**为标记每个记录，我们使用第5章的 TextPair 类，包括键(存储气象站ID)和“标记”**。**在这里，“标记”是一个虚拟的字段，其唯一目的是用于记录的排序**，**使气象站记录比天气记录先到达**。一种简单的做法就是:**对于气象站记录，“标记”值为0;对于天气记录，“标记”值为1**。范例9-9和范例9-10分别描述了执行该任务的两个mapper类。

**reducer知道自己会先接收气象站记录**。因此从中抽取出值，并将其作为后续每条输出记录的一部分写到输出文件。

**上述代码假设天气记录的每个气象站ID恰巧与气象站数据集中的一条记录准确匹配**。**如果该假设不成立，则需要泛化代码，使用另一个TextPair将标记放入值的对象中**。reduce()方法在处理天气记录之前，要能够区分哪些记录是气象站名称，检测(和处理)缺失或重复的记录。