几种无损数据压缩算法的探讨及在 JAVA Web 程序中的应用

肖武德
（甘肃广播电视大学定西市分校，甘肃 定西 743000）

摘　要：随着知识爆炸式的发展，数据压缩在计算机领域起到举足轻重的作用。Huffman 编码是一种常用的压缩方法，其原理是将使用次数多的代码转换成长度较短的代码，而使用次数少的可以使用较长的编码，并且保持编码的唯一可解性。LZ77 算法通过使用已经出现过的相应匹配数据信息替换当前数据从而实现压缩功能。LZW 算法是基于 LZ77 思想的一个变种。由于专利权原因，LZW 没有得到像 LZ77 一样的流行。DEFLATE 方法是 LZ77 算法与 Huffman 编码的组合，具有 LZ77 与 Huffman 编码的优势。DEFLATE 方法具有开源通用高压缩率的优势，因此，得到了广泛的应用。将采用 DEFLATE 方法的 GZIP 压缩应用到 B/S 架构的企业应用中可以减小网络传输的数据量，进而提高系统的整体性能。

关键词：无损压缩；Huffman 编码；LZ77；DEFLATE；LZW；GZIP JAVA
中图分类号：TP751.1

1　引言

自从有计算机以来，它的数据存储和传播能力一直在不断的发展，到目前已经达到了非常强大的地步。然而，这一时期也是人类的知识爆炸式发展的一个时期，很难衡量他们哪一个的速度更快。但是我们总能听到电脑用户抱怨磁盘空间不足，花费太长的时间下载需要的文件，从人们在使用计算机的过程中可以看到我们仍然期望计算机存储数据和传播数据的能力不断提高。当然我们可以升级计算机硬件，来提高电脑的性能，同时，计算机厂家不断涌现的新型号也意味着这一策略的庞大成本。在同样的硬件条件下，采用数据压缩可以存储更多的数据、获得更高的传输性能。另外，便捷终端、微型设备的出现更是需要很好的压缩算法来支持。比方说，遍布于数码录音笔、数码相机、数码随身听、数码摄像机等各种数字设备中的音频、图像、视频信息，就必须经过有效的压缩才能在硬盘上存储或是通过 USB 电缆传输。主要讨论几种常见的无损压缩算法，分析其原理，并通过对比给出其优缺点。最后给出 GZIP 在 Java Web 程序中的一种应用。

2　Huffman 编码

Huffman 编码是一种常用的压缩方法。是 1952 年为文本文件建立的，其基本原理是频繁使用的数据用较短的代码代替，很少使用的数据用较长的代码代替，每个数据的代码各不相同。这些代码都是二进制码，且码的长度是可变的。如：有一个原始数据序列，ABACCDAA 则编码为 A(0)，B(10)，C(110)，(D111)，压缩后为 0100110111011100。

Huffman 编码是无损压缩当中最好的方法。它使用预先二进制描述来替换每个符号，长度由特殊符号出现的频率决定。常见的符号需要很少的位来表示，而不常见的符号需要很多为来表示。哈夫曼算法在改变任何符号二进制编码引起少量密集表现方面是最佳的。然而，它并不处理符号的顺序和重复或序号的序列。而且产生霍夫曼编码需要对原始数据扫描两遍，第一遍扫描要精确地统计出原始数据中的每个值出现的频率，第二遍是建立霍夫曼树并进行编码，由于需要建立二叉树并遍历二叉树生成编码，因此数据压缩和还原速度都较慢。

2.1　Huffman 编码的压缩原理

Huffman 编码是一种可变长编码方式，是由美国数学家 David Huffman 创立的，是二叉树的一种特殊转化形式。编码的原理是：将使用次数多的代码转换成长度较短的代码，而使用次数少的可以使用较长的编码，并且保持编码的唯一可解性。Huffman 算法的最根本的原则是：累计的（字符的统计数字 * 字符的编码长度）为最小，也就是权值（字符的统计数字 * 字符的编码长度）的和最小。

2.2　Huffman 树

首先统计各个符号在文件中出现的次数，然后

根据符号的出现次数,建立 Huffman 树,通过 Huffman 树得到每个符号的新的编码。对于文件中出现次数较多的符号,它的 Huffman 编码的位数比较少。对于文件中出现次数较少的符号,它的 Huffman 编码的位数比较多。然后把文件中的每个字节替换成他们新的编码。

建立 Huffman 树:

把所有符号看成是一个结点,该结点的值为它的出现次数。进一步把这些结点看成是只有一个结点的树。每次从所有树中找出值最小的两个树,为这两个树建立一个父结点,把这两个树和它们的父结点组成一个新的树,新树的值为它的两个子树的值的和。如此往复,直到最后所有的树变成了一棵树。我们就得到了一棵 Huffman 树。

通过 Huffman 树得到 Huffman 编码:

Huffman 树是二叉树,它的所有叶子结点就是所有的符号,它的中间结点是在产生 Huffman 树的过程中不断建立的。我们在 Huffman 树的所有父结点到它的左子结点的路径上标上 0,右子结点的路径上标上 1。现在我们从根节点开始,到所有叶子结点的路径,就是一个 0 和 1 的序列。我们用根结点到一个叶子结点路径上的 0 和 1 的序列,作为这个叶子结点的 Huffman 编码。

例如:有一个文件的内容如下:abbbbccccddde

统计各个符号的出现次数:

a(1),b(4),c(4),d(3),e(1)

建立 Huffman 树的过程如图 1 所示。

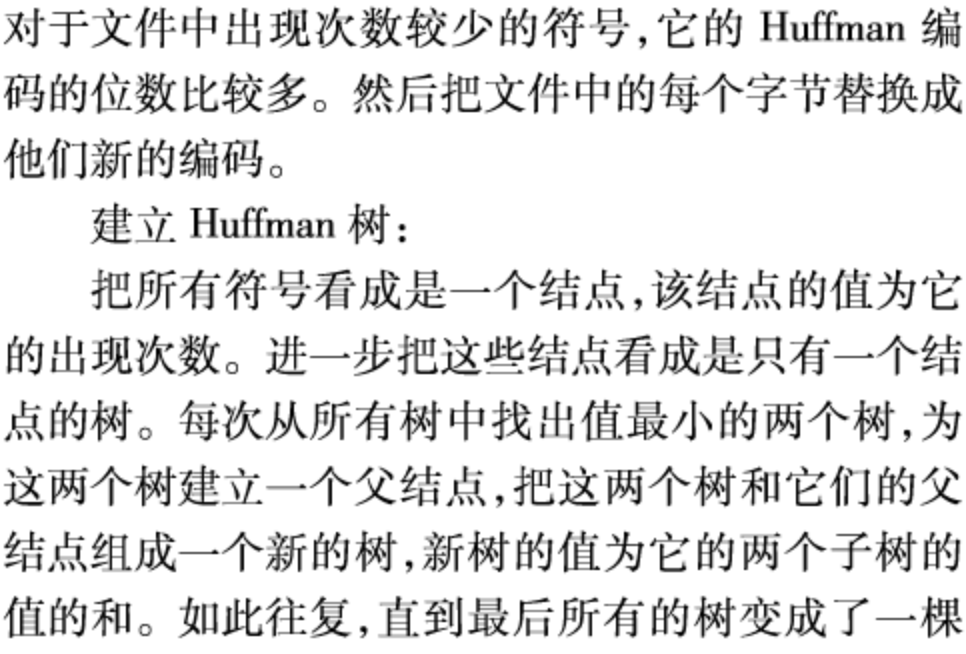

图 1 建立 Huffman 树的过程

通过最终的 Huffman 树,我们可以得到每个符号的 Huffman 编码。

a 为 110,b 为 00,c 为 01,d 为 10,e 为 111。

Huffman 树使用变长编码。对于变长编码,可能会遇到一个问题,就是重新编码的文件中可能会无法如区分这些编码。比如,a 的编码为 000,b 的编码为 0001,c 的编码为 1,那么当遇到 0001 时,就不知道 0001 代表 ac,还是代表 b。出现这种问题的原因是 a 的编码是 b 的编码的前缀。由于 Huffman 编码为根结点到叶子结点路径上的 0 和 1 的序列,而一个叶子结点的路径不可能是另一个叶子结点路径的前缀,所以一个 Huffman 编码不可能为另一个 Huffman 编码的前缀,这就保证了 Huffman 编码是可以区分的。

2.3 使用 Huffman 编码进行压缩和解压缩

为了在解压缩的时候,得到压缩时所使用的 Huffman 树,我们需要在压缩文件中,保存树的信息,也就是保存每个符号的出现次数的信息。

压缩:读文件,统计每个符号的出现次数。根据每个符号的出现次数,建立 Huffman 树,得到每个符号的 Huffman 编码。将每个符号的出现次数的信息保存在压缩文件中,将文件中的每个符号替换成它的 Huffman 编码,并输出。

解压缩:得到保存在压缩文件中的,每个符号的出现次数的信息。根据每个符号的出现次数,建立 Huffman 树,得到每个符号的 Huffman 编码。将压缩文件中的每个 Huffman 编码替换成它对应的符号,并输出。

3 LZ77 算法

LZ77 算法是字符串匹配的算法。例如:在一段文本中某字符串经常出现,并且可以通过前面文本中出现的字符串指针来表示。当然这个想法的前提是指针应该比字符串本身要短。

例如,在上一段短语“字符串”经常出现,可以将除第一个字符串之外的所有用第一个字符串引用来表示从而节省一些空间。

一个字符串引用通过下面的方式来表示:

* 唯一的标记
* 偏移数量
* 字符串长度

由编码的模式决定引用是一个固定的或变动的长度。后面的情况经常是首选,因为它允许编码器用引用的大小来交换字符串的大小(例如,如果字符串相当长,增加引用的长度可能是值得的)。

3.1 LZ77 算法原理

LZ77 是 Abraham Lempel 在 1977 年发表的无损数据压缩算法。LZ77 算法是基于字典的编码器。LZ77 算法通过使用编码器或者解码器中已经出现过的相应匹配数据信息替换当前数据从而实现压缩

功能。这个匹配信息使用称为长度 - 距离对的一对数据进行编码,它等同于“每个给定长度个字符都等于后面特定距离字符位置上的未压缩数据流。”(“距离”有时也称作“偏移”)。

编码器和解码器都必须保存一定数量的最近的数据,如最近 2 KB、4 KB 或者 32 KB 的数据。保存这些数据的结构叫作滑动窗口,因为这样所以 LZ77 有时也称作滑动窗口压缩。编码器需要保存这个数据查找匹配数据,解码器保存这个数据解释编码器所指代的匹配数据。所以编码器可以使用一个比解码器更小的滑动窗口,但是反过来却不行。

3.2　LZ77 算法的基本流程

1)从当前压缩位置开始,考察未编码的数据,并试图在滑动窗口中找出最长的匹配字符串(使用哈希表),如果找到,则进行步骤 2,否则进行步骤 3。

2)输出三元符号组(off,len,c)。其中 off 为窗口中匹配字符串相对窗口边界的偏移,len 为可匹配的长度,c 为下一个字符。然后将窗口向后滑动 len +1 个字符,继续步骤 1。

3)输出三元符号组(0,0,c)。其中 c 为下一个字符。然后将窗口向后滑动 len +1 个字符,继续步骤 1。

结合实例来说明。假设窗口的大小为 10 个字符,刚编码过的 10 个字符是:abcdbbccaa,即将编码的字符为:abaeaaabaee。可以直接和要编码字符匹配的最长串为 ab(off = 0,len =2),ab 的下一个字符为 a,我们输出三元组:(0,2,a)。现在窗口向后滑动 3 个字符,窗口中的内容为:dbbccaaaba。下一个字符 e 在窗口中没有匹配,我们输出三元组:(0,0,e)。窗口向后滑动 1 个字符,其中内容变为:bbccaaabae。所以要编码的 aaabae 在窗口中存在(off = 4,len =6),其后的字符为 e,可以输出:(4,6,e)。这样,我们将可以匹配的字符串都变成了指向窗口内的指针,并由此完成了对上述数据的压缩。

解压缩的过程十分简单,只要我们向压缩时那样维护好滑动的窗口,随着三元组的不断输入,我们在窗口中找到相应的匹配串,缀上后继字符 c 输出(如果 off 和 len 都为 0 则只输出后继字符 c)即可还原出原始数据。

3.3　LZ77 算法的编码方法

必须精心设计三元组中每个分量的表示方法,才能达到较好的压缩效果。一般来讲,编码的设计要根据待编码的数值的分布情况而定。

3.3.1　编码偏移值(off)

偏移接近窗口尾部的情况要多于接近窗口头部的情况,这是因为字符串在与其接近的位置较容易找到匹配串,但对于普通的窗口大小(例如:4096 字节)来说,偏移值基本是均匀分布的,可以用固定的位数来表示它。

需要的位数 bitnum = upper_bound(log2(MAX_WND_SIZE))

由此,如果窗口大小为 4096,用 12 位就可以对偏移编码。如果窗口大小为 2048,用 11 位就可以了。

复杂一点的程序考虑到在压缩开始时,窗口大小并没有达到 MAX_WND_SIZE,而是随着压缩的进行增长,因此可以根据窗口的当前大小动态计算所需要的位数,这样可以略微节省一点空间。

3.3.2　编码字符串长度(len)

匹配的字符串长度在多数时候时不会太大,少数情况下才会发生大字符串的匹配。显然可以使用一种变长的编码方式来表示该长度值。要输出变长的编码,该编码必须满足前缀编码的条件。Huffman 编码便可以在此处使用(但不是最好的选择。适用于此处的好的编码方案很多,比如 Golomb 编码)。

4　LZW 算法

LZW 算法是基于 LZ77 思想的一个变种。LZW 压缩效率较高。其基本原理是把每一个第一次出现的字符串用一个数值来编码,在还原程序中再将这个数值还成原来的字符串,如用数值 0x100 代替字符串" abccddeee" 这样每当出现该字符串时,都用 0x100 代替,起到了压缩的作用。至于 0x100 与字符串的对应关系则是在压缩过程中动态生成的,而且这种对应关系是隐含在压缩数据中,随着解压缩的进行这张编码表会从压缩数据中逐步得到恢复,后面的压缩数据再根据前面数据产生的对应关系产生更多的对应关系。直到压缩文件结束为止。LZW 是可逆的, 所有信息全部保留。LZW 属于无损压缩编码,该编码主要用于图像数据的压缩。对于简单图像和平滑且噪声小的信号源具有较高的压缩比,并且有较高的压缩和解压缩速度。由于专利权原因,LZW 没有得到像 LZ77 一样的流行。

5　DEFLATE 方法

DEFLATE 方法是 LZ77 算法与 Huffman 编码的组合。GZIP、ZIP 即采用这个方法。如前所述,LZ77

算法先将文件压缩表示为(唯一的标记,偏移量,字符串长度)三元组的方式,Huffman 编码再进一步对偏移量、字符串长度进行压缩。在 LZ77 算法中,需要利用哈希表对字符串进行匹配。

DEFLATE 方法集合了 LZ77 与 Huffman 编码的优势。并且不受任何专利权制约。它具有通用的开放源码无版权工业标准,因此得到了广泛的应用。

6 GZIP 压缩在 Java Web 程序中的应用

DEFLATE 方法用途十分广泛,采用它的 GZIP 压缩目前在 HTTP 压缩中非常流行。随着互联网和企业信息化的不断发展,越来越多的系统采用 B/S 的架构,使用 Java Web 程序或 J2EE 框架无疑是实现 B/S 架构最好的方法之一。随着系统规模的扩大,系统性能的优化也是软件工程师和用户关心的主要问题。解决这类问题,一般会想到的是数据库调优、程序代码优化或缓存处理,诚然这些方法可以提高系统的性能,这里要说的是另一种方式:在 Web 应用中采用 GZIP 压缩技术,减少网络数据的传输量。在 Java Web 程序中有两种方式运用 GZIP 压缩技术。

6.1 设置 Web 服务器

目前大部分 Java Web 服务器都支持 GZIP 压缩技术,Tomcat 服务器从版本 5.0 之后开始支持对输出内容压缩,下面是在 tomcat 6.0.20 环境下中开启 GZIP 的方法。

在 tomcat 的安装目录下找到 conf 目录,打开 server.xml 文件可以看到如下内容:

```
        ⋮
<Service name="Catalina">
    <Connector port="80" protocol="HTTP/1.1"
    connectionTimeout="20000" redirectPort="
    8443" />
        ⋮
</Service>
        ⋮
```

将内容修改为:

```
        ⋮
<Service name="Catalina">
        <Connector port="80" protocol="HTTP/
        1.1" connectionTimeout="20000" redirect-
        Port="8443" compression="on" compres-
        sionMinSize="2048" noCompression-
        UserAgents="gozilla, traviata" compres-
        sableMimeType="text/html, text/xml, text/
        javascript, text/css, text/plain"
        />
        ⋮
</Service>
        ⋮
```

6.2 使用 filter 方式

这种方式主要是对 HttpServletResponse 进行包装,获取它向客户端浏览器所有的输出,用 filter 进行 GZIP 处理,等处理完毕,再输出其内容到 HttpServletResponse 对象中。J2EE 框架中已经定义了一个 HttpServletResponseWrapper 类使得包装 HttpServletResponse 更加容易。

首先定义 Wrapper 用来包装 HttpServletResponse 对象:

```
public class Wrapper extends HttpServletResponse-
Wrapper
{
    ⋮
}
```

其次定义 filer:

```
public class GzipFilter implements Filter
{
    ⋮
}
```

filter 中处理 GZIP 压缩的主要代码如下:

```
public void doFilter(ServletRequest request,ServletRe-
sponse response,
        FilterChain chain) throws IOException,
ServletException
{
    HttpServletResponse resp = (HttpServletRe-
sponse)response;
    Wrapper wrapper = new Wrapper(resp);
    chain.doFilter(request,wrapper);
    ByteArrayOutputStream byteOutput = new Byte-
ArrayOutputStream(10240);
    GZIPOutputStream gzipOutput = new GZIPOut-
putStream(byteOutput);
    gzipOutput.write(wrapper.getResponseData());
    byte[] gzipData = byteOutput.toByteArray();
    resp.addHeader("Content-Encoding","gz-
ip");
```

(下转第 39 页)

(上接第 48 页)

```
        resp. setContentLength(gzipData. length);
        ServletOutputStream output = response. getOutputStream();
        output. write(gzipData);
        output. flush();

}
```

7 结束语

随着网络承载的信息的飞速增长,数据压缩必然会备受重视。本文比较了几种常见无损压缩算法,详细分析了使用广泛的 DEFLATE 方法,在本文的最后指出了 GZIP 压缩的一种应用。LZW 压缩技术有很多变体,例如常见的 ARC、RKARC、PKZIP 高效压缩程序,但由于 LZW 拥有专利,所以使用的范围受到限制。DEFLATE 同时使用了 LZ77 算法与哈夫曼编码,不受任何专利所制约。文章只讨论了数据压缩在某一方面的应用,事实上数据压缩已经在信息技术上广泛应用,并将在深度和广度上不断延伸。

参考文献:

[1] [LZ77] Ziv J, Lempel A. A Universal Algorithm for Sequential Data Compression[J]. IEEE Transactions on Information Theory, 23(3):337-343.

[2] A Method for the Construction of Minimum - Redundancy Codes[J]. David A. Huffman, 1952.

[3] 吴乐南. 数据压缩(第 2 版). 电子工业出版社, 2008.

[4] MARK ALLEN WEISS. 数据结构与算法分析—C 语言描述[M]. 机械工业出版社, 2004.

[5] 吴家安. 数据压缩技术及应用[M]. 科学出版社, 2009.

[6] THOMAS H CORMEN, CHARLES E LEISERSON. 算法导论(第 2 版)[M]. 机械工业出版社, 2006.