RapiD’s GZ demo ver 0.1

Develop Manual

1. 概述

本次PJ代码大约为5500行，是我写过的最长的PJ（囧）。

1. 开发体会

本次PJ经历了数次重构，最大的一次重构在HTC的帮助下加入了泛型，删除了几乎所有的强制类型装换，本来希望藉此提升代码的运行速度，但是效果非常不明显，而且大量使用的泛型导致程序架构变得不灵活，再加上初期的过度设计，导致后期优化代码的时候十分乏力，最终程序的运行速度十分捉急，而又无力优化。可以说这是本次PJ获得的最大的教育，设计过过度带来的问题不仅仅使程序变得十分巨大臃肿，而且在优化上带来极大的阻力。

之后数次算法优化无果后，我开始尝试在已经有的代码上增加多线程。由于huffman树算法多线程存在特殊性（分块后不同块之间的算法是可以并行的，但是最后块之间的输出顺序不能改变），给多线程的设计带来很大的难度，最后不得不使用大量的同步锁来保证代码并行的正确性，但是如此带来的Overhead十分巨大，导致最终多线程的效果非常差。

其他代码的部分实现起来都是很快的，基本在一次实现过后只存在很小的bug，在简单调试之后都可以发现。

1. 运行效果评估

在运行速度方面，任何格式的文件几乎都没有什么区别，因为huffman算法本身不依赖于文件的具体格式来进行压缩。在最终测试中，Test文件夹的压缩/解压时间为 23s/24s,大多数时间花费在压缩图片和音频上。广播体操的时间为43s/49s,相比于零散文件，建立文件索引和拼接的时间呗省去了，效率有略微的提升。在运行速度方面和7z几乎一样。

在压缩率上，首相对于小于100b的文件，因为建立索引会占据额外空间，压缩结果往往是比原来大的。对于正常大小的文件，文本文件的压缩率最好，在67%左右，相比于7z的10%，还是差很远。对于音频，视频，图片等文件，因为这类文件已经有过压缩，压缩率一般在90%以上，7z在这上面的表现类似，但是依然会稍微领先。对于普通的二进制文件，压缩率一般在75%以上，同比与7z的45%左右，差距很大。

1. 其他说明

由于代码篇幅太大，关于代码的reference由javadoc生成，在此不再赘述。