# 第二代 MurmurHash

Austin Appleby在2008年发布了一个新的散列函数——MurmurHash。其最新版本大约是lookup3速度的2倍（大约为1 byte/cycle），它有32位和64位两个版本。32位版本只使用32位数学函数并给出一个32位的哈希值，而64位版本使用了64位的数学函数，并给出64位哈希值。根据Austin的分析，MurmurHash具有优异的性能，虽然Bob Jenkins 在《Dr. Dobbs article》杂志上声称“我预测[MurmurHash ]比起lookup3要弱，但是我不知道具体值，因为我还没测试过它”。MurmurHash能够迅速走红得益于其出色的速度和统计特性。

参考 Jedis java版本实现: redis.clients.util.MurmurHash

# 第三代: CityHash 和 SpookyHash

2011年，发布了两个散列函数，相对于MurmurHash ，它们都进行了改善，这主要应归功于更高的指令级并行机制。Google发布了CityHash（由Geoff Pike 和Jyrki Alakuijala编写），Bob Jenkins发布了他自己的一个新散列函数SpookyHash（这样命名是因为它是在万圣节发布的）。它们都拥有2倍于MurmurHash的速度，但他们都只使用了64位数学函数而没有32位版本，并且CityHash的速度取决于CRC32 指令，目前为SSE 4.2（Intel Nehalem及以后版本）。SpookyHash给出128位输出，而CityHash有64位，128位以及256位的几个变种

# 比较

从我所了解的情况来看，这篇文章中所提到的所有散列函数从统计学角度来看已经足够好。需要考虑的一个因素是CityHash/SpookyHash的输出超过了64位，但是对于一个32位的hash table来说这输出太多了。其他应用可能会用到128或256位输出。

如果你用的是32位机器，MurmurHash看起来是最明显的赢家，因为它是唯一一个快于lookup3的32位原生版本。32位机器很可能可以编译并运行City和Spooky，但我预计它们的运行速度和在64位机器上运行的速度比起来要慢的多，毕竟32位机器需要模拟64位的数学运算。

在64位机器上，由于没有更深层次的基准，也很难说哪种算法是最好的。比起City我更倾向于Spooky，因为City的运行速度需要依赖于CRC32指令，毕竟这种环境并不是任何机器上都有的。

另一个需要考虑的是对齐和非对齐的访问。Murmur散列（不像City或者Spooky）是一个仅能进行对齐读取的变种，因为在很多架构上非对齐的读取会崩溃或者返回错误的数据（非对齐的读取操作在C中是未定义的行为）。City和Spooky都强调使用memcpy()把输入数据复制到对齐的存储结构中；Spooky使用一次memcpy()操作一个块（如果ALLOW\_UNALIGNED\_READS未定义），City使用一次memcpy()操作一个整型！在可以处理非对称读取的机器上（像x86和x86-64），memcpy将被优化，但我在我的小ARM上做了一个测试，发现如下：

#include <stdint.h>

#include <string.h>

int32\_t read32\_unaligned(const void \*buf) {

int32\_t ret;

memcpy(&ret, buf, 4);

return ret;

}

编译这段低效的代码（在x86上是一个单独的操作）：

0: b500 push {lr}

2: 2204 movs r2, #4

4: b083 sub sp, #12

6: 4601 mov r1, r0

8: eb0d 0002 add.w r0, sp, r2

c: f7ff fffe bl 0

10: 9801 ldr r0, [sp, #4]

12: b003 add sp, #12

14: bd00 pop {pc}

结论是，如果你需要32位或者仅仅是对齐读取的话，Murmur散列看起来依旧是最好的选择。City散列和Spooky散列在x86-64上看起来更快，但我更倾向于认为它们是特定用于那个架构的，因为我不知道是否有其他既是64位又允许非对其读取的架构。

# 其他参考

https://www.cnblogs.com/hzorac/p/5399042.html