今儿看到两位同学在讨论GetHashCode方法的执行，貌似他们的对象等同性判断挺复杂的，而又想把同样的逻辑写在GetHashCode中，然后激烈的讨论着用什么**[算法](http://lib.csdn.net/base/datastructure" \o "算法与数据结构知识库" \t "http://blog.csdn.net/cnhk1225/article/details/_blank)**让GetHashCode返回的整数更具唯一性。

我觉得简而言之GetHashCode的作用就是：尽量用最快的时间对对象进行初步判断。当然这里时间的快慢和判断的深度没有具体要求，只要没有走极端就可以（比如太费时间，或者判断深度太浅）。因此没必要吧GetHashCode搞得太复杂！

还有人错误的认为字典的存储是完全靠GetHashCode的结果，显然这是不对的，GetHashCode仅返回一个int怎能胜任所有结果呢？

来看这个例子，这样一个类，他的GetHashCode返回整数数据的余2的结果。（仅为做示例，很显然这个GetHashCode的执行很有效率但是比较深度也太差了）

class a{ public int Id { get; private set; } public a(int i) { Id = i; }

public override bool Equals(object obj) { Console.WriteLine("Equals"); if (obj == null || GetType() != obj.GetType()) { return false; }

return Id == ((a)obj).Id; }

//返回余2的结果 public override int GetHashCode() { Console.WriteLine("GetHashCode"); return Id % 2; }}

接着执行代码：

var o1 = new a(1); //GetHashCode返回1var o2 = new a(2); //GetHashCode返回0var o3 = new a(3); //GetHashCode返回1

var dic = new Dictionary<a, object>();dic.Add(o1, 123);Console.WriteLine("分隔符");Console.WriteLine(dic.ContainsKey(o2));Console.WriteLine("分隔符");Console.WriteLine(dic.ContainsKey(o3));

程序输出：

GetHashCode分隔符GetHashCodeFalse分隔符GetHashCodeEqualsFalse

可以看到，当GetHashCode可以直接分辨出不相等时，Equals就没必要调用了，而当GetHashCode返回相同结果时，Equals方法会被调用从而确保判断对象是否真的相等。所以，还是那句话：GetHashCode没必要一定把对象分辨得很清楚（况且它也不可能，一个int不可能代表所有的可能出现的值），有Equals在后面做保障。GetHashCode仅需要对对象进行快速判断。

最后你可能有些疑惑为什么不直接用Equals非得搞个GetHashCode在前面先判断一下？这个是由于Equals方法必须把两个对象搞清楚是等于还是不等于，所以可能效率不是最优的（况且Object.Equals通常包含类型的转换，这个可以参考IEquatable或IEqualityComparer，他们支持泛型），而GetHashCode不需要绝对弄清楚是否相等所以可以优化下效率。举个最简单的例子，比较两个人是不是完全一样（一样的话代表是他的克隆人），Equals会一个细胞接一个细胞得比较，而GetHashCode可以通过判断性别，长相，声音……快速得进行判断。所以先用GetHashCode会很快的判断出许多不同的人，当然如果GetHashCode返回True（遇到了双胞胎或者克隆人），再用Equals进行彻底的比较。