Understanding hash codes

devblogs.microsoft.com/oldnewthing/20050831-17

August 31, 2005

Raymond Chen

On more than one occasion, I’ve seen someone ask a question like this:

> I have some procedure that generates strings dynamically, and I want a formula that takes a string and produces a small unique identifer for that string (a hash code), such that two identical strings have the same identifier, and that if two strings are different, then they will have different identifiers. I tried String.GetHashCode(), but there were occasional collisions. Is there a way to generate a hash code that guarantees uniqueness?

If you can restrict the domain of the strings you’re hashing, you can sometimes squeak out uniqueness. For example, if the domain is a finite set, you can develop a so-called perfect hash which guarantees no collisions among the domain strings.

In the general case, where you do not know what strings you are going to be hashing, this is not possible. Suppose your hash code is a 32-bit value. This means that there are 2^{32} possible hash values. But there are more than 2^{32} possible strings. Therefore, by the pigeonhole principle, there must exist at least two strings with the same hash code.

One little-known fact about the pigeonhole principle is that it has nothing to do with pigeons. The term “pigeonhole” refers to a small compartment in a desk into which items such as papers or letters are distributed. (Hence the verb “to pigeonhole”: To assign a category, often based on gross generalization.) The pigeonhole principle, then, refers to the process of sorting papers into pigeonholes, and not the nesting habits of members of the family *Columbidae*.

Raymond Chen

Follow