Tweaking our computation of the interval between two moments in time

devblogs.microsoft.com/oldnewthing/20050415-52

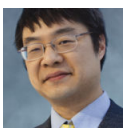
April 15, 2005

Raymond Chen

We can take our computation of the interval between two moments in time and combine it with the trick we developed for using the powers of mathematics to simplify multi-level comparisons to reduce the amount of work we impose upon the time/date engine.

```
static void PrintAge(DateTime bday, DateTime asof)
{
 // Accumulate years without going over.
 int years = asof.Year - bday.Year;
 if (asof.Month*32 + asof.Day < bday.Month*32 + bday.Day) years--;
 DateTime t = bday.AddYears(years);
 // Accumulate months without going over.
 int months = asof.Month - bday.Month;
 if (asof.Day < bday.Day) months--;
 months = (months + 12) % 12;
 t = t.AddMonths(months);
 // Days are constant-length, woo-hoo!
 int days = (asof - t).Days;
 SC.WriteLine("{0} years, {1} months, {2} days",
              years, months, days);
}
```

Observe that we avoided a call to the `AddYears` method (which is presumably rather complicated because years are variable-length) by replacing it with a multi-level comparison to determine whether the ending month/day falls later in the year than the starting month/day. Since no month has 32 days, a multiplier of 32 is enough to avoid an overflow of the day into the month field of the comparison key.



Raymond Chen

Follow