

Calendar.getInstance mítosz

Fejlesztői beszélgetéseken sokszor elkerül, hogy idmérésre a Calendar használata kerülend, mivel túl sok erőforrást igényel a példányosítás, illetve a használata. A fejlesztők ebből inkább jobban szeretik a System osztály `currentTimeMillis` metódusát használni, amely egyszer UNIX időbélyeget ad vissza. A tesztelés egyszer, mérjük meg, hogy az egyes módszerek mennyi időt vesznek el a program életéből. A mérési keretrendszer egyszer, ciklusban lekérdezzük egy tömbbe az időbélyegeket:

```
long[] result = new long[10000000];
for (int count = 0; count < 10000000; count++)
{
    Calendar cal = Calendar.getInstance();
    result[count] = cal.getTimeInMillis();
}
System.out.println("" + (result[9999999] - result[0]) + " ms");
```

A másik módszer egyszerűbbnek néz ki, de a végeredmény azonos lesz:

```
long[] result = new long[10000000];
for (int count = 0; count < 10000000; count++)
{
    result[count] = System.currentTimeMillis();
}
System.out.println("" + (result[9999999] - result[0]) + " ms");
```

Hívjuk meg többször egymás után a fenti programrészleteket, hogy a JVM is fel tudjon pörögni, és nézzük meg az utolsó pár hívás átlagát:

- Ötször mért 10 millió `System.currentTimeMillis()` hívás átlaga 12000 ms, egy hívás 1200 ns
- Ötször mért 10 millió `Calendar.getInstance()` hívás átlaga 21000 ms, egy hívás 2100 ns

A mérés szerint a Calendar használata 75 százalékkal lassabb, mint a System hívás, ám ez a különbség érthető a példányosítás okán, hiszen a Calendar (vagyis egy `GregorianCalendar`) példány is egy `System.currentTimeMillis()` hívás alapján állítja be a belső változóit. Java 5.0 óta elérhető egy új metódus is a System osztályban, amely egymilliószor pontosabb mérést tesz lehetővé: a `nanoTime`. Igazából csak a lehetősége van meg, hogy nano másodperc felbontással mérjük eltelt időt, jelenleg nincs olyan architektúra, amely támogatná egy ilyen felbontású időmérést, bár a processzorok órajele már meghaladta a szükséges 1GHz értéket.

Nos, mérjük újra és számoljuk ki ismét az utolsó pár mérési eredmény átlagát:

```
long[] result = new long[10000000];
for (int count = 0; count < 10000000; count++)
{
    result[count] = System.nanoTime();
}
System.out.println("" + (result[9999999] / 1000000 - result[0] / 1000000) + " ms");
```

Az eredmény:

- Ötször mért 10 millió `System.currentTimeMillis()` hívás átlaga 12000 ms, egy hívás **1200 ns**
- Ötször mért 10 millió `Calendar.getInstance()` hívás átlaga 21000 ms, egy hívás **2100 ns**
- Ötször mért 10 millió `System.nanoTime()` hívás átlaga 14000 ms, egy hívás **1400 ns**

A `nanoTime` metódus mintegy 15-20 százalékkal lassabb, mint a `currentTimeMillis` metódus, ám a felbontása még átlagos gépen is használhatóbb eredményt ad. Ezt alátámasztandó, nézzük meg a letárolt értékek első tíz elemét `System.currentTimeMillis()` hívás esetén:

```
1247913669850
1247913669850
1247913669850
1247913669850
1247913669850
1247913669850
1247913669850
1247913669850
1247913669850
1247913669850
```

Ugyanez `System.nanoTime()` hívás esetén:

74858347480430
74858347481896
74858347483363
74858347484830
74858347486296
74858347487763
74858347489230
74858347490627
74858347492093
74858347493490

Láthatjuk, hogy az utóbbi esetben szépen tükrözdik a 1400 ns ideig tartó hívás, míg az elbbi esetben azt láthatjuk, hogy az amúgy 1200 ns ideig tartó hívásból kell 800 darab, míg más értéket ad vissza a metódus.