Why are there so many implementations of Object Pooling in Roslyn?

Asked 9 years, 6 months ago Modified 8 years, 10 months ago Viewed 8k times

38

The ObjectPool is a type used in the Roslyn C# compiler to reuse frequently used objects which would normally get new'ed up and garbage collected very often. This reduces the amount and size of garbage collection operations which have to happen.

The Roslyn compiler seems to have a few separate pools of objects and each pool has a different size. I want to know why there are so many implementations, what the preferred implementation is and why they picked a pool size of 20, 100 or 128.

1 - SharedPools - Stores a pool of 20 objects or 100 if the BigDefault is used. This one is also strange in that it creates a new instance of PooledObject, which makes no sense when we are trying to pool objects and not create and destroy new ones.

```
// Example 1 - In a using statement, so the object gets freed at the end.
using (PooledObject<Foo> pooledObject = SharedPools.Default<List<Foo>>
().GetPooledObject())
{
    // Do something with pooledObject.Object
}

// Example 2 - No using statement so you need to be sure no exceptions are not
thrown.
List<Foo> list = SharedPools.Default<List<Foo>>().AllocateAndClear();
// Do something with list
SharedPools.Default<List<Foo>>().Free(list);

// Example 3 - I have also seen this variation of the above pattern, which ends
up the same as Example 1, except Example 1 seems to create a new instance of
the IDisposable [PooledObject<T>][3] object. This is probably the preferred
option if you want fewer GC's.
List<Foo> list = SharedPools.Default<List<Foo>>().AllocateAndClear();
try
{
    // Do something with list
}
finally
{
    SharedPools.Default<List<Foo>>().Free(list);
}
```

2 - ListPool and StringBuilderPool - Not strictly separate implementations but wrappers around the SharedPools implementation shown above specifically for List and StringBuilder's. So this re-uses the pool of objects stored in SharedPools.

```
// Example 1 - No using statement so you need to be sure no exceptions are
thrown.
StringBuilder stringBuilder= StringBuilderPool.Allocate();
// Do something with stringBuilder
StringBuilderPool.Free(stringBuilder);

// Example 2 - Safer version of Example 1.
StringBuilder stringBuilder= StringBuilderPool.Allocate();
try
{
    // Do something with stringBuilder
}
finally
{
    StringBuilderPool.Free(stringBuilder);
}
```

3 - PooledDictionary and PooledHashSet - These use ObjectPool directly and have a totally separate pool of objects. Stores a pool of 128 objects.

```
// Example 1
PooledHashSet<Foo> hashSet = PooledHashSet<Foo>.GetInstance()
// Do something with hashSet.
hashSet.Free();

// Example 2 - Safer version of Example 1.
PooledHashSet<Foo> hashSet = PooledHashSet<Foo>.GetInstance()
try
{
    // Do something with hashSet.
}
finally
{
    hashSet.Free();
}
```

Update

There are new object pooling implementations in .NET Core. See my answer for the C# Object Pooling Pattern implementation question.

c# .net garbage-collection roslyn

Share
Improve this question
Follow

edited Feb 14, 2016 at 14:09

asked Jun 3, 2015 at 10:55
Muhammad Rehan Saeed
38.3k ● 47 ● 216 ● 326

3 Considering Microsoft always resisted the concept of object pools in .NET because they always said that GC of gen0 objects is very fast, it is an interesting turnaround :-) – xanatos Jun 4, 2015 at 8:47

1 A compiler isn't a real-time app where you don't want pauses... And it isn't like SO where hundred of users will connect concurrently. They wanted to optimized it because they didn't want it to be slower than the older compiler and they used an Object Pool... But it doesn't mean I need to like what they did. – xanatos Jun 4, 2015 at 9:16

2 And I would have done the same thing they did :-) And I would have felt smart *and* dirty at the same time :-) – xanatos Jun 4, 2015 at 9:38

5 @xanatos A command line compiler doesn't care about pauses. A compiler that's built into an IDE does. – svick Jun 5, 2015 at 22:00

3 I guess this is eventually about performance, so I believe the answer is basic and not what you want to hear. When you're really optimizing stuff, you don't really care about what's there; you just want it to be as bloody fast as possible. So, you find a hotspot, think of a possible way to optimize that, see if the existing code does exactly what you want it to do - and if it's not *exactly* what you want, you simply implement it. In the world of HELL (Highly Efficient Low Level code) there's no such thing as 'proper design'; everything is allowed, the end goal is the only thing that matters. – atlaste Jun 9, 2015 at 8:50

1 Answer

Sorted by: Highest score (default)

50

+50

I'm the lead for the Roslyn performance v-team. All object pools are designed to reduce the allocation rate and, therefore, the frequency of garbage collections. This comes at the expense of adding long-lived (gen 2) objects. This helps compiler throughput slightly but the major effect is on Visual Studio responsiveness when using the VB or C# IntelliSense.

> why there are so many implementations".

There's no quick answer, but I can think of three reasons:

1. Each implementation serves a slightly different purpose and they are tuned for that purpose.
2. "Layering" - All the pools are internal and internal details from the Compiler layer may not be referenced from the Workspace layer or vice versa. We do have some code sharing via linked files, but we try to keep it to a minimum.
3. No great effort has gone into unifying the implementations you see today.

> what the preferred implementation is

`ObjectPool<T>` is the preferred implementation and what the majority of code uses. Note that `ObjectPool<T>` is used by `ArrayBuilder<T>.GetInstance()` and that's probably the largest user of pooled objects in Roslyn. Because `ObjectPool<T>` is so heavily used, this is one of the cases where we duplicated code across the layers via linked files. `ObjectPool<T>` is tuned for maximum throughput.

At the workspace layer, you'll see that `SharedPool<T>` tries to share pooled instances across disjoint components to reduce overall memory usage. We were trying to avoid having each component create its own pool dedicated to a specific purpose and, instead share based on the type of element. A good example of this is the `StringBuilderPool`.

> why they picked a pool size of 20, 100 or 128.

Usually, this is the result of profiling and instrumentation under typical workloads. We usually have to strike a balance between allocation rate ("misses" in the pool) and the total live bytes in the pool. The two factors at play are:

1. The maximum degree of parallelism (concurrent threads accessing the pool)
2. The access pattern including overlapped allocations and nested allocations.

In the grand scheme of things, the memory held by objects in the pool is very small compared to the total live memory (size of the Gen 2 heap) for a compilation but, we do also take care not to return giant objects (typically large collections) back to the pool - we'll just drop them on the floor with a call to `ForgetTrackedObject`

For the future, I think one area we can improve is to have pools of byte arrays (buffers) with constrained lengths. This will help, in particular, the MemoryStream implementation in the emit phase (PEWriter) of the compiler. These MemoryStreams require contiguous byte arrays for fast writing but they are dynamically sized. That means they occasionally need to resize - usually doubling in size each time. Each resize is a new allocation, but it would be nice to be able to grab a resized buffer from a dedicated pool and return the smaller buffer back to a different pool. So, for example, you would have a pool for 64-byte buffers, another for 128-byte buffers and so on. The total pool memory would be constrained, but you avoid "churning" the GC heap as buffers grow.

Thanks again for the question.

Paul Harrington.

Share Improve this answer Follow

answered Jun 9, 2015 at 15:20

pharring

2,046 ●1 ●15 ●10

4 No, thank you Paul for the answer! This is what I love about StackOverflow, ask a question about some software and the developer turns up and answers it for you. I am looking into object pooling for my ASP.NET MVC Boilerplate project. – Muhammad Rehan Saeed Jun 9, 2015 at 15:41

1 Why cant we just get rid of this relic of the '90s??? Now it is open source we should do what Delphi does on mobile and objective-c does just free that particular object automatically. it is putting a Free() call when appropriate. – Joe Apr 18, 2016 at 16:52