.NET Core 3.0 中默认启用了分层编译 (TC)。 此功能使运行时能够更适应地使用实时 (JIT) 编译器来实现更好的性能。

分层编译的主要优势是**提供两种实现实时的方法**，可在低质量快速层或高质量慢速层中编译。 **质量是指方法的优化程度。** 这有助于提高应用程序在从启动到稳定状态的各个执行阶段的性能。 禁用分层编译后，每种方法都以同一种方式进行编译，这种方式倾向于牺牲启动性能来保证稳定状态性能。

启用 TC 后，以下行为适用于应用启动时的方法编译：

- 如果方法具有预先编译的代码 (ReadyToRun)，将使用预生成的代码。
- 否则，将实时编译该方法。 一般来说，这些方法是泛型而不是值类型。
  - **快速 JIT 可以更快地生成较低质量（优化程度较低）的代码。** 在 .NET Core 3.0 中，默认为不包含循环的方法启用了快速 JIT，并且启动过程中首选快速 JIT。
  - **完全优化的 JIT 可生成更高质量（优化程度更高）的代码，但速度更慢(上面的两种方式快速JIT和完全优化JIT即是说明JIT后代码的性能也是有差别的，粗略的JIT相当于完全没有优化的机器码，而完全优化的JIT相当于会生成优化后的机器码速度更快。** 对于不使用快速 JIT 的方法（例如，如果该方法具有 MethodImplOptions.AggressiveOptimization 特性），则使用完全优化的 JIT。

对于频繁调用的方法，实时编译器最终会在后台创建完全优化的代码。 然后，优化后的代码将替换该方法的预编译代码。

通过快速 JIT 生成的代码可能会运行较慢、分配更多内存或使用更多堆栈空间。 如果出现问题，可以在项目文件中使用此 MSBuild 属性禁用快速 JIT：

XML

```
<PropertyGroup> <TieredCompilationQuickJit>false</TieredCompilationQuickJit> </PropertyGroup>
```

若要完全禁用 TC，请在项目文件中使用此 MSBuild 属性：

XML

```
<PropertyGroup> <TieredCompilation>false</TieredCompilation> </PropertyGroup>
```

提示

如果在项目文件中更改这些设置，则可能需要执行干净的生成以反映新的设置（删除 `obj` 和 `bin` 目录并重新生成）。

有关在运行时配置编译的详细信息，请参阅用于编译的运行时配置选项。