

# 开源课程期末作业

小组成员：

卢哲钰51215903011，负责第二部分

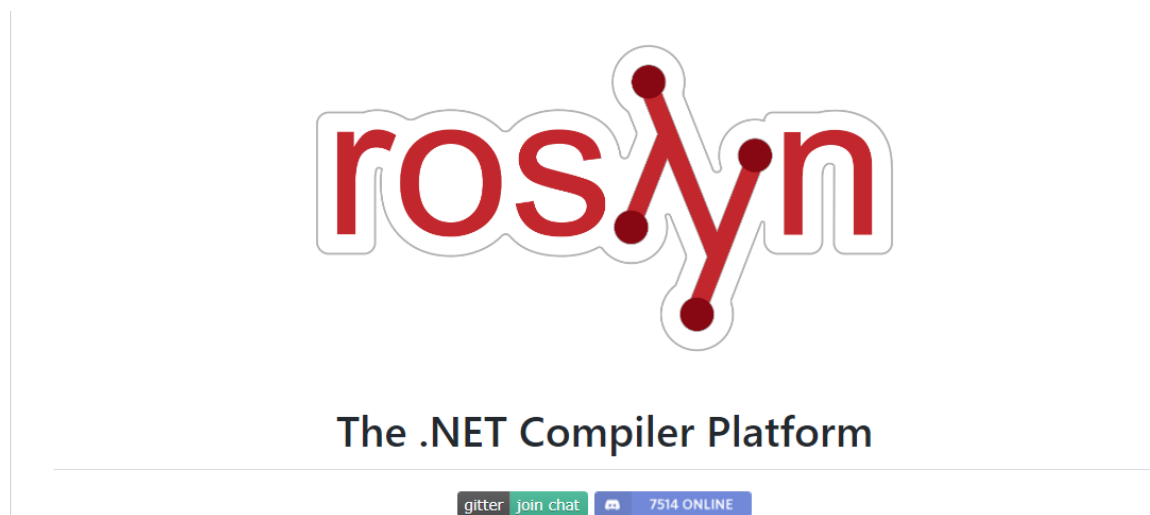
李雪51215903030，负责第二部分

阎松 51215903029，负责一、三部分

## 一、项目的基本背景和发展历程介绍

通过项目仓库的查阅，以及搜索引擎上项目的相关资料，给出项目一些基本的介绍：

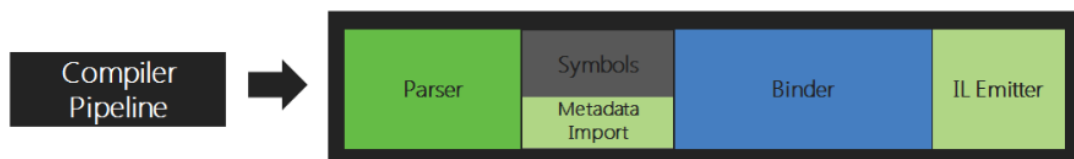
技术类型：



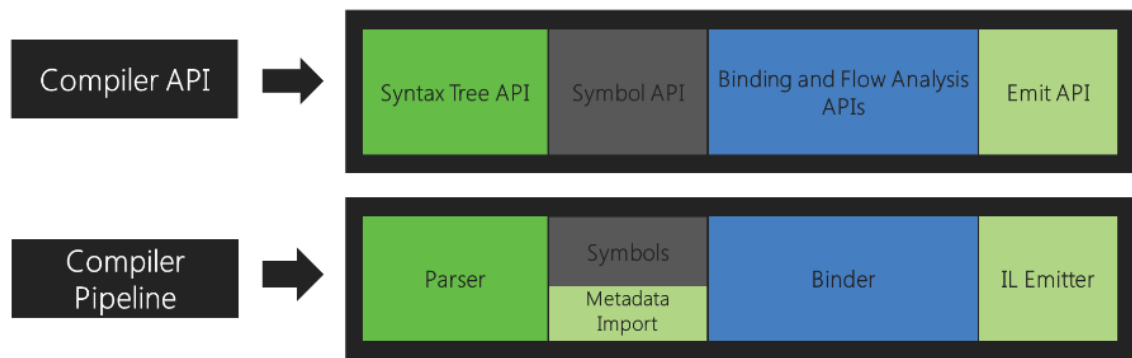
.NET 编译器平台（“Roslyn”）为开源 C# 和 Visual Basic 编译器提供了丰富的代码分析 API。Roslyn 是 C# 和 Visual Basic 编译器的开源实现，具有用于构建代码分析工具的 API 接口。

编译器按照结构化规则处理您编写的代码，这些规则通常与人类阅读和理解代码的方式不同。对编译器使用的模型有基本的了解，对于理解构建基于 Roslyn 的工具时使用的 API 至关重要。

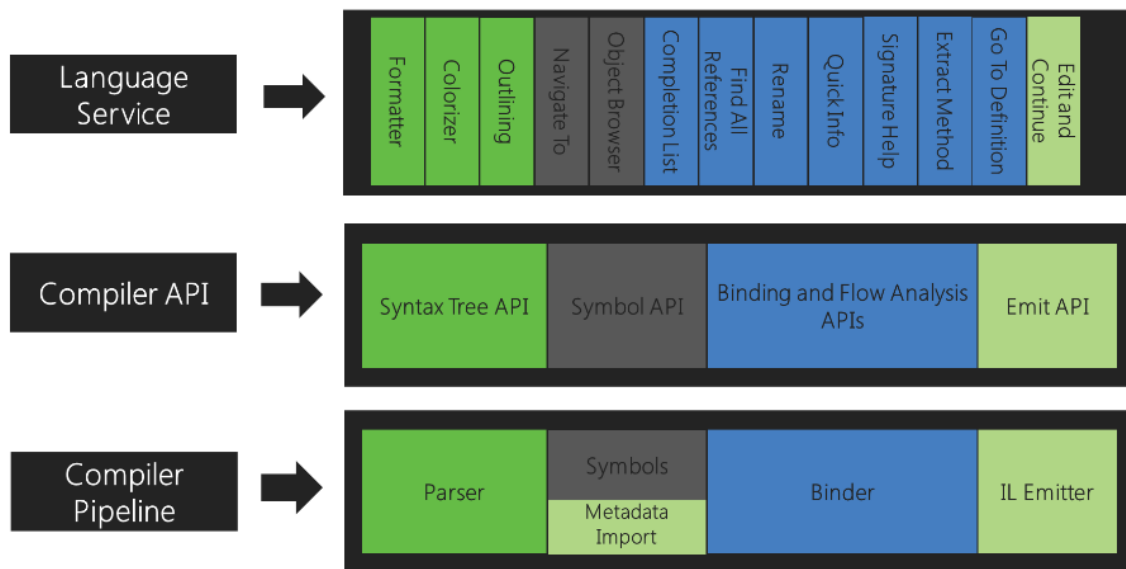
.NET 编译器平台 SDK 通过提供反映传统编译器管道的 API 层，向消费者公开 C# 和 Visual Basic 编译器的代码分析。



该管道的每个阶段都是一个单独的组件。首先，解析阶段将源文本标记化并解析为遵循语言语法的语法。其次，声明阶段分析源和导入的元数据以形成命名符号。接下来，绑定阶段将代码中的标识符与符号进行匹配。最后，发射阶段发出一个程序集，其中包含编译器构建的所有信息。



与这些阶段中的每一个相对应，.NET 编译器平台 SDK 公开了一个对象模型，允许访问该阶段的信息。解析阶段暴露了语法树，声明阶段暴露了分层符号表，绑定阶段暴露了编译器语义分析的结果，发出阶段是产生IL字节码的API。



每个编译器将这些组件组合在一起作为一个端到端的整体。

这些 API 与 Visual Studio 使用的相同。例如，代码大纲和格式化功能使用语法树，**对象浏览器**和导航功能使用符号表，重构和**转到定义**使用语义模型，**编辑和继续**使用所有这些，包括 Emit API。

## API 层

.NET 编译器 SDK 由多个 API 层组成：编译器 API、诊断 API、脚本 API 和工作区 API。

### 编译器 API

编译器层包含与在编译器管道的每个阶段（句法和语义）公开的信息相对应的对象模型。编译器层还包含编译器单次调用的不可变快照，包括程序集引用、编译器选项和源代码文件。有两种不同的 API 分别代表 C# 语言和 Visual Basic 语言。这两个 API 的形状相似，但针对每种单独的语言进行了高保真度的定制。该层不依赖于 Visual Studio 组件。

### 诊断 API

作为其分析的一部分，编译器可能会生成一组诊断，涵盖从语法、语义和明确的赋值错误到各种警告和信息诊断的所有内容。Compiler API 层通过可扩展的 API 公开诊断，允许将用户定义的分析器插入编译过程。它允许与编译器定义的诊断一起生成用户定义的诊断，例如由 StyleCop 等工具生成的诊断。以这种方式生成诊断具有与 MSBuild 和 Visual Studio 等工具自然集成的优势，这些工具依赖于诊断来获得体验，例如根据策略停止构建并在编辑器中显示实时曲线和建议代码修复。

## 脚本 API

托管和脚本 API 是编译器层的一部分。您可以使用它们来执行代码片段并累积运行时执行上下文。C# 交互式 REPL（读取-评估-打印循环）使用这些 API。REPL 使您能够将 C# 用作脚本语言，在编写代码时以交互方式执行代码。

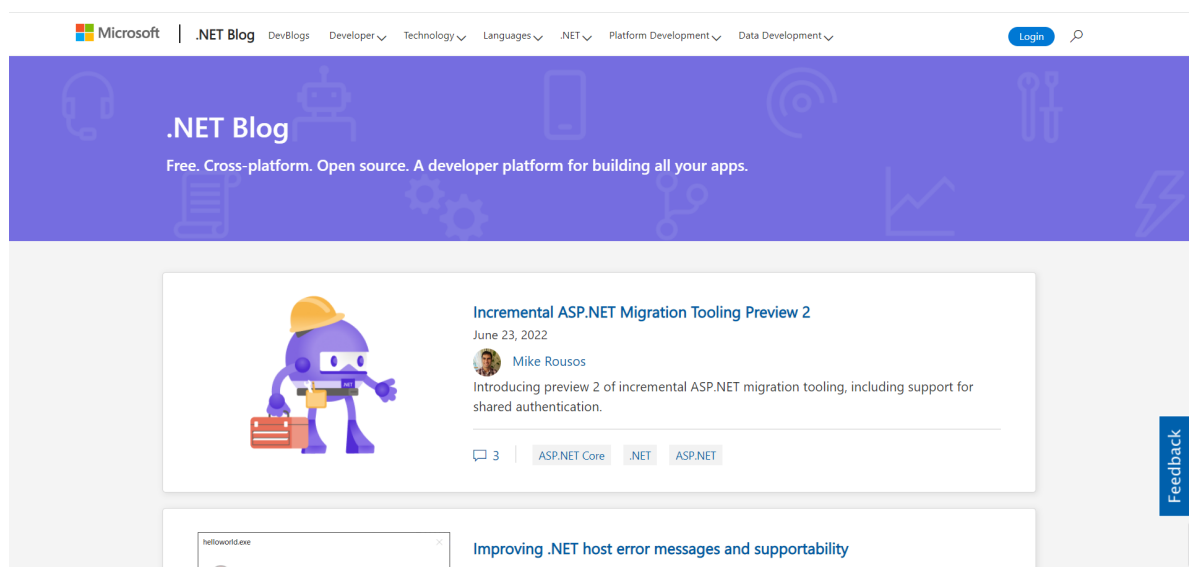
## 工作区 API

Workspaces 层包含 Workspace API，它是对整个解决方案进行代码分析和重构的起点。它帮助您将解决方案中有关项目的所有信息组织到单个对象模型中，让您可以直接访问编译器层对象模型，而无需解析文件、配置选项或管理项目到项目的依赖关系。

此外，Workspaces 层还展示了一组 API，这些 API 在实现代码分析和重构工具时使用，这些工具在 Visual Studio IDE 等宿主环境中运行。示例包括查找所有引用、格式化和代码生成 API。

该层不依赖于 Visual Studio 组件。


下图是他们的开发博客。



## 版本发布历史：

The Roslyn .NET compiler在其github网站上面共发布58次releases，但目前是更新到了.NET 7.0版本，大的版本更新有7次。

Releases 58

 .NET 6.0.1 **Latest**  
on 15 Dec 2021

+ 57 releases

- 最新的一次更新是2022年6月14日：.NET 7.0.0 Preview 5 和 .NET SDK 7.0.100-preview.5.22307.18 版本可供下载。最新的 7.0 版本始终列在.NET 7.0 Releases中。

.NET 7将是当前版本，支持 18 个月，从 2022 年 11 月到 2024 年 5 月 14 日。Microsoft在多个操作系统上支持它。

.NET 7 Preview 5 现已推出，其中包括对 ASP.NET Core 的许多重大新改进。

以下是此预览版中新增功能的摘要：

- 1.JWT 身份验证改进和自动身份验证配置；
- 2.Minimal API 对参数列表简化的参数绑定支持。

## .NET 7 Releases

Date	Release
2022/06/14	<a href="#">7.0.0 Preview 5</a>
2022/05/10	<a href="#">7.0.0 Preview 4</a>
2022/04/13	<a href="#">7.0.0 Preview 3</a>
2022/03/15	<a href="#">7.0.0 Preview 2</a>
2022/02/17	<a href="#">7.0.0 Preview 1</a>

- 2021年12月15日正式发布了.NET 6.0.1版本，之前的迭代是从2021年2.18到2021年12.15。.NET 6 是LTS 版本，将支持三年，从 2021 年 11 月到 2024 年 11 月。它在多个操作系统上受 Microsoft 支持。

.NET 6 版本包括对 macOS 和 Windows Arm64 操作系统的支持。

## .NET 6 Releases

Date	Release
2022/06/14	<a href="#">6.0.6</a>
2022/05/10	<a href="#">6.0.5</a>
2022/04/12	<a href="#">6.0.4</a>
2022/03/08	<a href="#">6.0.3</a>
2022/02/08	<a href="#">6.0.2</a>
2021/12/14	<a href="#">6.0.1</a>
2021/11/08	<a href="#">6.0.0</a>
2021/10/12	<a href="#">6.0.0 RC 2</a>
2021/09/14	<a href="#">6.0.0 RC 1</a>
2021/08/10	<a href="#">6.0.0 Preview 7</a>
2021/07/14	<a href="#">6.0.0 Preview 6</a>
2021/06/17	<a href="#">6.0.0 Preview 5</a>
2021/05/25	<a href="#">6.0.0 Preview 4</a>
2021/04/08	<a href="#">6.0.0 Preview 3</a>
2021/03/11	<a href="#">6.0.0 Preview 2</a>
2021/02/17	<a href="#">6.0.0 Preview 1</a>

- 2021年3.10日进行了最后一次.NET5.0.4的更新。.NET 5于 2020 年 5 月 10 日发布。该版本是许多贡献者的协作努力。.NET 5 于 2022 年 5 月 10 日终止支持| 它在多个操作系统上得到 Microsoft 的支持。

## .NET 5 Releases

Date	Release
2020/05/10	Out of Support
2022/05/10	<a href="#">5.0.17</a>
2022/04/12	<a href="#">5.0.16</a>
2022/03/08	<a href="#">5.0.15</a>
2022/02/08	<a href="#">5.0.14</a>
2021/12/14	<a href="#">5.0.13</a>
2021/11/08	<a href="#">5.0.12</a>
2021/10/12	<a href="#">5.0.11</a>
2021/09/14	<a href="#">5.0.10</a>
2021/08/10	<a href="#">5.0.9</a>
2021/07/13	<a href="#">5.0.8</a>
2021/06/08	<a href="#">5.0.7</a>
2021/05/25	<a href="#">5.0.300</a>
2021/05/11	<a href="#">5.0.6</a>
2021/04/06	<a href="#">5.0.5</a>
2021/03/09	<a href="#">5.0.4</a>
2021/03/02	<a href="#">5.0.200</a>
2021/02/09	<a href="#">5.0.3</a>
2021/01/12	<a href="#">5.0.2</a>
2020/12/08	<a href="#">5.0.1</a>
2020/11/10	<a href="#">5.0.0</a>

**主要贡献者的构成（国家、区域和组织等）：**

Used by 4.3k



Contributors 525



+ 514 contributors

The Roslyn .NET compiler 有4.3K的人在使用，然后有514位贡献者。

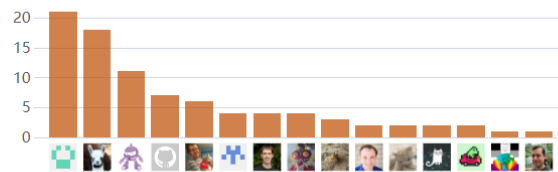
近一周的PR请求情况是：

2022 年 6 月 21 日 – 2022年6 月 28 日

周期：1周

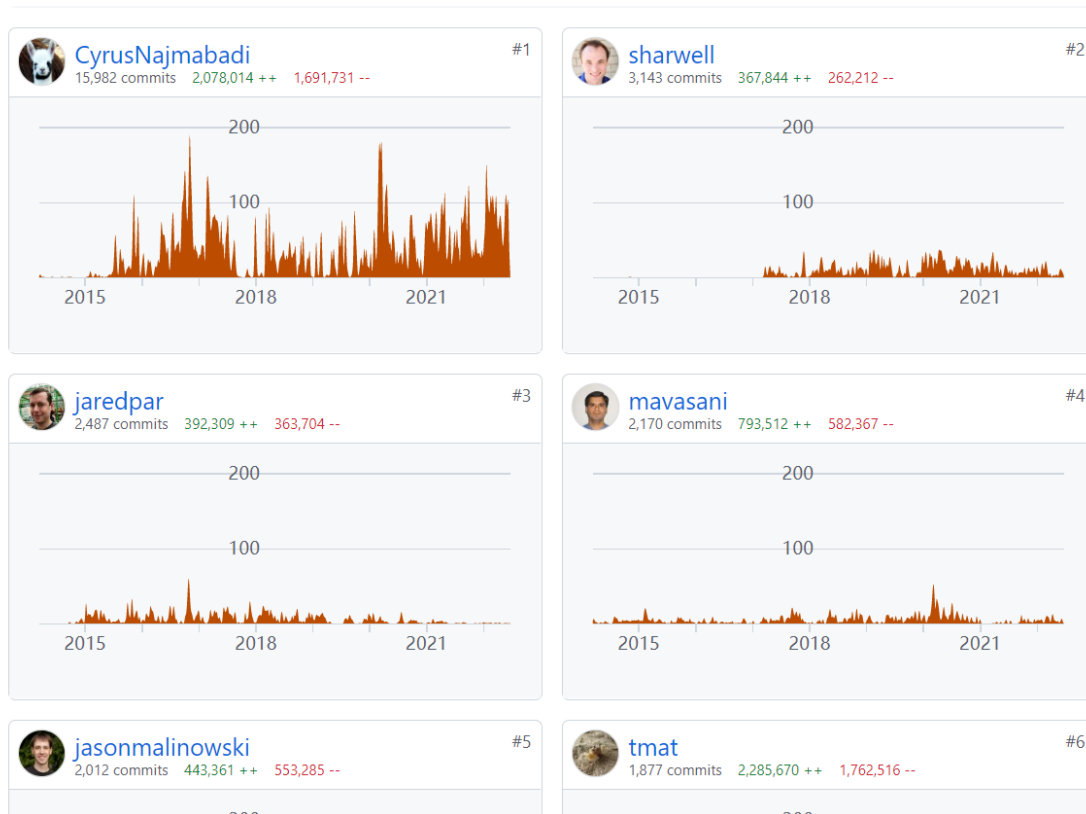


不包括合并，21 位作者 将 52次提交推送到主分支，92次提交推送到所有分支。在 main 上，有638 个文件 发生了变化，增加了 34,887个文件，删除了7,570个文件。



18人 合并了50 个 拉取请求

CyrusNajmabadi 有15,982次commits,是最多的一个贡献者。



sharwell有3,143次 commits，来自加利福尼亚州圣地亚哥。在Microsoft工作。

jaredpar有2,487次commits，也是一位主要贡献者。同样在Microsoft工作。

mavasani有2,170次commits。同样在Microsoft工作。

其他的贡献者或多或少的进行了1-2,000次以内的commits。

## CI/CD 的使用：

CI/CD 是一种持续开发软件的方法，可以不断的进行构建、测试和部署代码迭代更改。这种迭代有助于减少基于错误或失败的版本进行开发新代码的可能性。使用这种方法，从新代码开发到部署，可以减少人工干预甚至不用干预。

达到持续的方法主要是：持续集成，持续交付，持续部署。

CI (Continuous Integration)：持续集成，也就是当每一次更改的代码被推送到远程分支后，可以创建一组脚本来自动地构建和测试这些更改，确保这些更改可以通过一些基本的准则，减少引入错误的机会。

CD：

Continuous Delivery：持续交付，在持续集成的基础上更进一步，当每一次更改的代码落库后，不仅会构建和测试，也会进行部署，但是部署需要人工干预，手动的有目的进行部署。

Continuous Deployment：持续部署，持续集成之外的另一个步骤，类似于持续交付。不同之处在于，它不是手动部署应用程序，而是将其设置为自动部署。不需要人为干预。

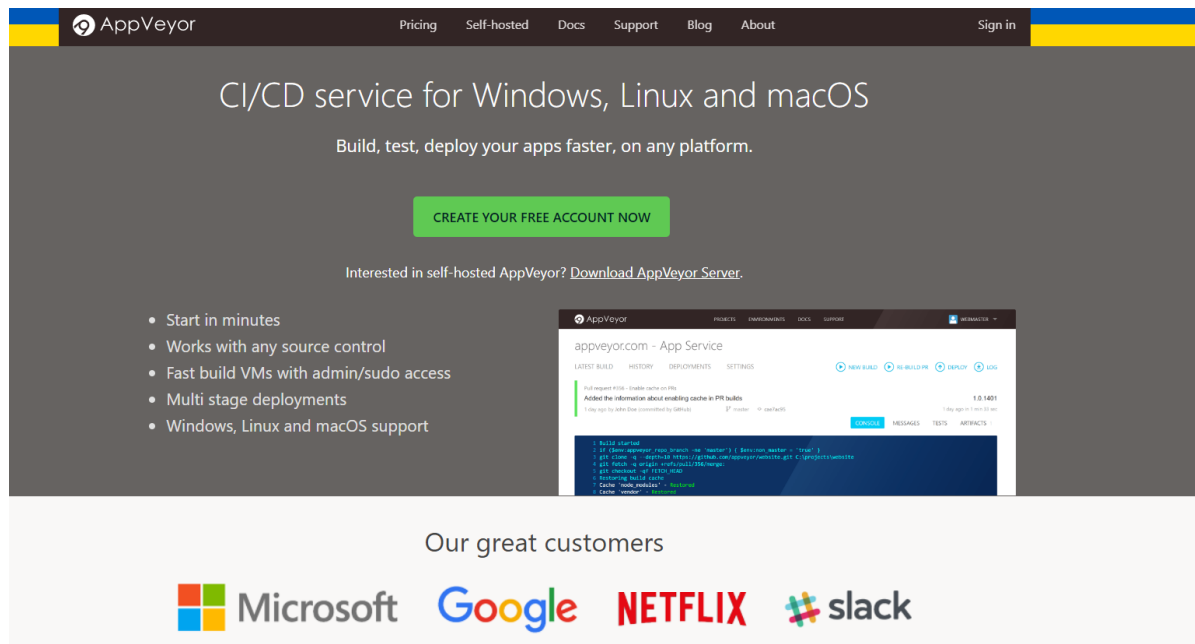
在项目迭代过程，可能有一个专门负责 CI/CD 的人员，但当想做一些静态代码检查，依赖检查，图片大小检查等事情的时候，就自己需要了解 CI/CD，编写特定 Pipeline Job。

在实际项目中，CI 主要是提交或合并代码的时候触发，负责一些基本规则的检查，如果检查遇到失败，那么回滚或修改代码后再提交或合并，降低代码风险；CD 主要手动的触发，在CI的基础上，还负责功能检查，如果功能符合验收标准，那么就可以交付或部署。



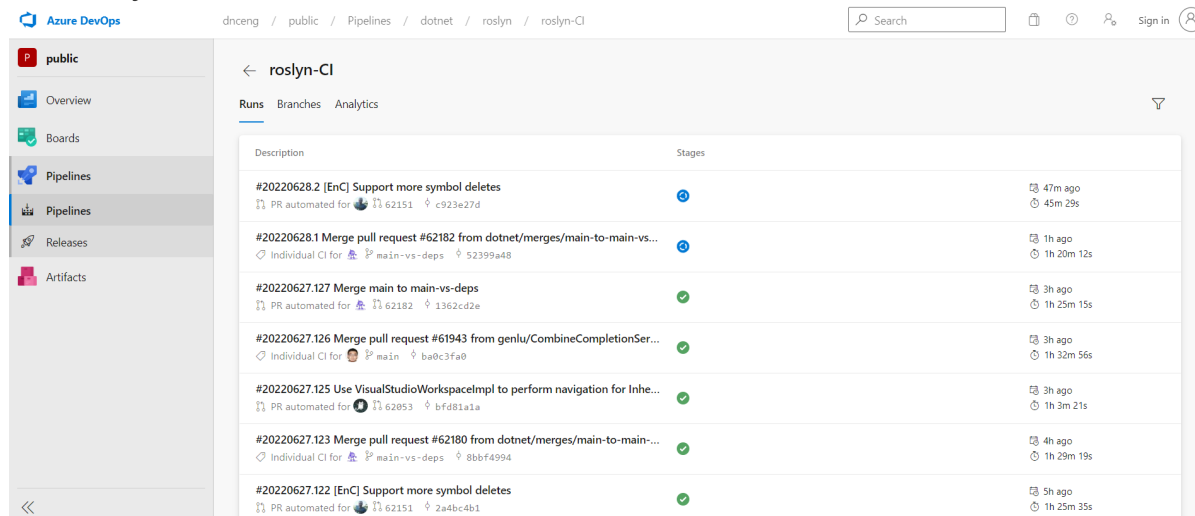
通过一个工具，我们可以看到其使用情况。

在Github上增加了一个第三方的插件—Appveyor，来简单的实现了CI/CD 操作,通过注册账号,然后各种配置以后,可以实现每次向Github提交,会自动编译，然后生成报告。



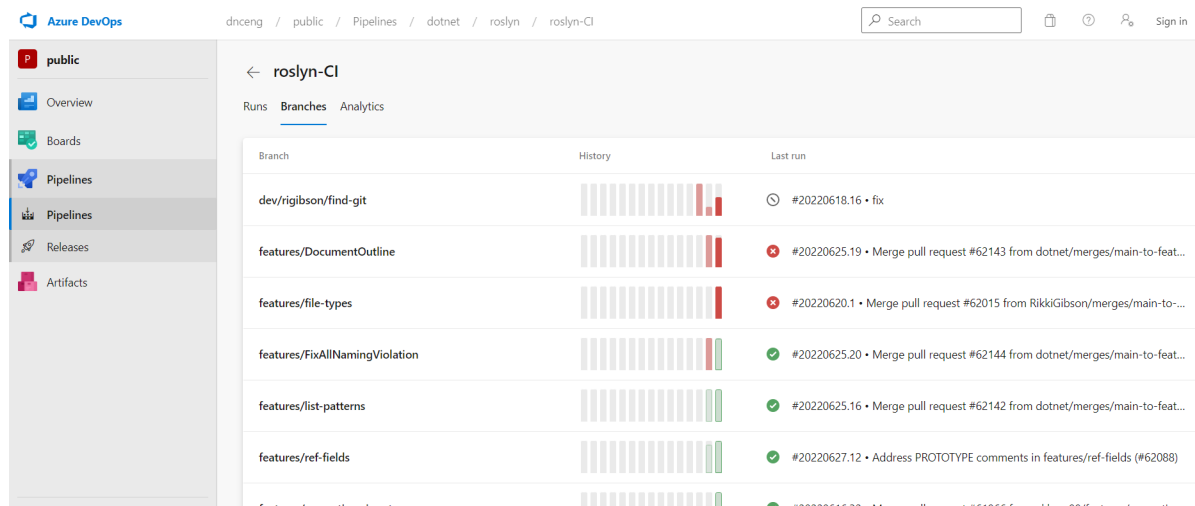
The screenshot shows the AppVeyor website. At the top, there's a navigation bar with links for Pricing, Self-hosted, Docs, Support, Blog, About, and a Sign in button. The main heading is "CI/CD service for Windows, Linux and macOS" with a subtext "Build, test, deploy your apps faster, on any platform." Below this is a prominent green button that says "CREATE YOUR FREE ACCOUNT NOW". Further down, there's a link "Interested in self-hosted AppVeyor? Download AppVeyor Server." and a list of features: "Start in minutes", "Works with any source control", "Fast build VMs with admin/sudo access", "Multi stage deployments", and "Windows, Linux and macOS support". To the right of the list is a screenshot of the AppVeyor web interface showing a build log. At the bottom, there's a section titled "Our great customers" featuring logos for Microsoft, Google, NETFLIX, and slack.

下面是roslyn-CI。



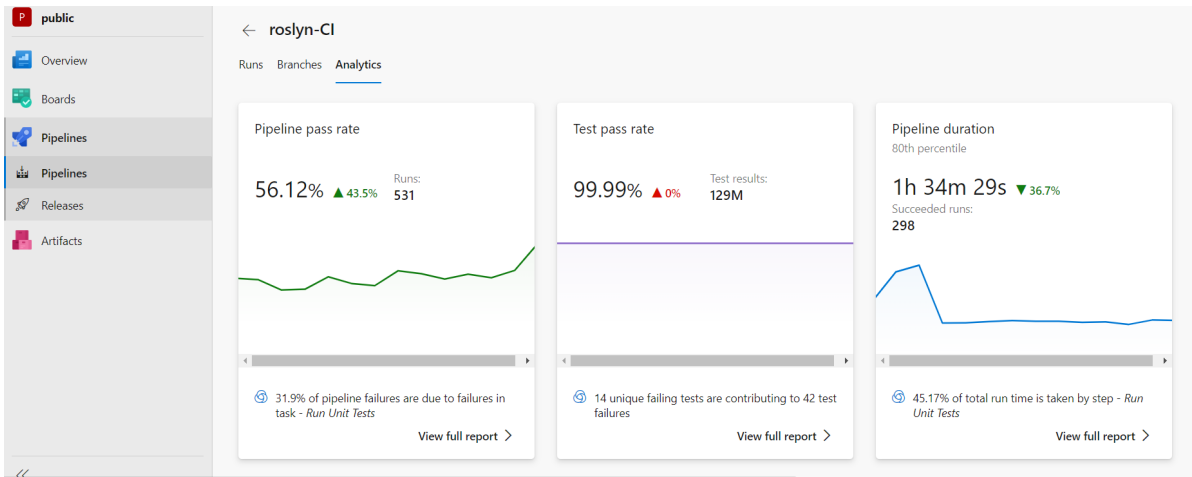
This screenshot shows the "Runs" tab of a pipeline named "roslyn-CI" in Azure DevOps. The left sidebar contains navigation links for Overview, Boards, Pipelines, Releases, and Artifacts. The main area displays a table of pipeline runs. Each row includes a description, a status icon (blue circle with a white 'i' for in progress, green checkmark for completed), and a timestamp. The runs are triggered by pull requests from various branches like 'main-vs-deps' and 'main'.

Description	Stages	Status	Time
#20220628.2 [EnC] Support more symbol deletes PR automated for 62151 c923e27d		In Progress	47m ago 45m 29s
#20220628.1 Merge pull request #62182 from dotnet/merges/main-to-main-vs-... Individual CI for main-vs-deps 52399a48		In Progress	1h ago 1h 20m 12s
#20220627.127 Merge main to main-vs-deps PR automated for 1362cd2e		Completed	3h ago 1h 25m 15s
#20220627.126 Merge pull request #61943 from genlu/CombineCompletionSer... Individual CI for main ba0c3fa0		Completed	3h ago 1h 32m 56s
#20220627.125 Use VisualStudioWorkspaceml to perform navigation for Inhe... PR automated for 62053 bfd81a1a		Completed	3h ago 1h 3m 21s
#20220627.123 Merge pull request #62180 from dotnet/merges/main-to-main-... Individual CI for main-vs-deps 8bbf4994		Completed	4h ago 1h 29m 19s
#20220627.122 [EnC] Support more symbol deletes PR automated for 62151 2a4bc4b1		Completed	5h ago 1h 25m 35s



This screenshot shows the "History" tab of the "roslyn-CI" pipeline. The left sidebar is the same as the previous screenshot. The main area displays a table with columns for Branch, History (represented by a bar chart), and Last run. The table lists various branches and their corresponding build status and timestamps.

Branch	History	Last run
dev/rigibson/find-git		#20220618.16 • fix
features/DocumentOutline		#20220625.19 • Merge pull request #62143 from dotnet/merges/main-to-feat...
features/file-types		#20220620.1 • Merge pull request #62015 from RikkGibson/merges/main-to-...
features/FixAllNamingViolation		#20220625.20 • Merge pull request #62144 from dotnet/merges/main-to-feat...
features/list-patterns		#20220625.16 • Merge pull request #62142 from dotnet/merges/main-to-feat...
features/ref-fields		#20220627.12 • Address PROTOTYPE comments in features/ref-fields (#62088)
features/semantic-enrichment		#20220616.32 • Merge pull request #61966 from alkhara90/features/semantic...



## 其他有价值的信息：

The screenshot shows the GitHub Issues page for the 'roslyn' repository. The page displays a list of open issues, with filters for 'is:issue is:open'. The issues are sorted by 'Open' date. The first issue is 'API proposal: Expose functionality of `SyntaxGenerator.TypedConstantExpression` via a better method', which is a 'Feature Request' and 'untriaged'. The second issue is 'Typo in API name: `ShouldShowItemsFromUnimportNamespaces`', which is an 'Area-IDE' issue and 'untriaged'. The third issue is 'Generated constructor with invalid parameter name', which is an 'Area-IDE' issue, a 'Bug', and 'help wanted'. The fourth issue is 'Local constant is considered unused when it's used as default parameter value for local function', which is an 'Area-Compilers' issue and 'untriaged'. The fifth issue is 'EE doesn't display `ref` fields as expected', which is an 'Area-Interactive' issue, a 'Concept-Design Debt', and 'Interactive-Debugging'. The sixth issue is 'Test ref fields in the Expression Evaluator', which is a 'New Language Feature - Ref Fields' issue.

有人提出：一个API方案，提供了这个方案，就能够更好的方法公开功能。

但这样做会导致：内部 API 由 Roslyn 本身的一些代码修复使用，这会阻止将这些代码修复移动到共享的分析器层。

因此这个提出的方案不可以。

另外，这个工作也用到了语法树。

“语法树”是一种由编译器 API 公开的基础的不可变数据结构。这些树表示源代码的词法和语法结构。它们有两个重要用途：

- 支持使用工具（如 IDE、加载项、代码分析工具和重构）查看和处理用户项目中源代码的语法结构。
- 支持使用工具（如重构和 IDE）以自然的方式创建、修改和重新排列源代码，而无需直接编辑文本。通过创建和操作语法树，可轻松使用工具创建和重新排列源代码。

## 二、项目的历史轨迹分析

见Sourcetrail.ipynb。

## 三、结合期中分析的归档项目，对比分析活跃/归档项目

# 期中期末项目数据对比分析

- 项目基础数据 (2.1/2.2/2.3) 的变化趋势

## 1. 每月新增 Star 和 Fork

Sourcetrail的Star 和 Fork：（项目刚开始是增长迅速，后来直到归档每月趋于下降）

☆ 12.7k stars  
👁 276 watching  
🔗 1.1k forks

The Roslyn .NET compiler (roslyn) 的Star 和 Fork：（保持平稳递进）

☆ 16.1k stars  
👁 1k watching  
🔗 3.6k forks

## 2. 每月打开 Issue 和 关闭 Issue

Sourcetrail每月打开和关闭Issue量：（项目开始缓慢增长，但每月打开Issue最高数量仅为62，每月关闭Issue最高数量为38.最后每月Issue打开和关闭数量几乎为0）

每月打开issue量

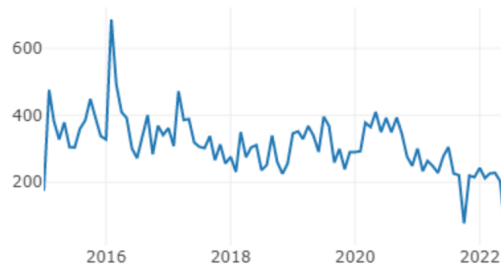


每月关闭issue量



The Roslyn .NET compiler (roslyn) 每月打开和关闭issue量：（项目开始增长速度较快，后段时间增长较慢，但每月打开Issue最高数量为493，每月关闭Issue最高数量为625.后来Issue打开和关闭数量都不为0）

每月打开issue量



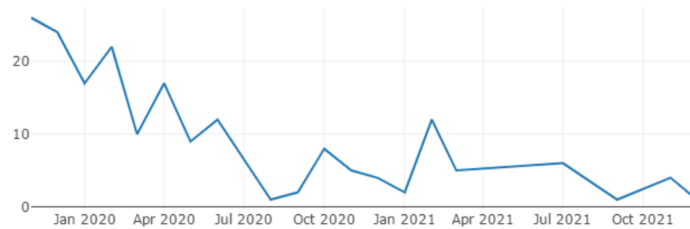
每月关闭issue量



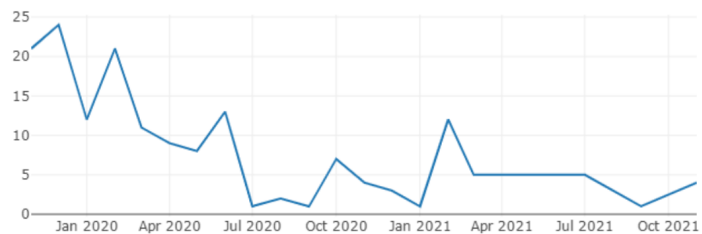
### 3. 每月打开 PR 和合入 PR

Sourcetrail 每月打开 PR 和合入 PR 量: (不断下降, 最后几乎为0)

每月打开PR量

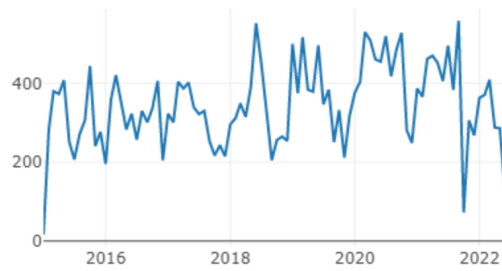


每月合入PR量



The Roslyn .NET compiler (roslyn) 每月打开 PR 和合入 PR 量: (数量不断波动, 但都不为0)

每月打开PR量



每月合入PR量



- 开发者数量 (2.4) 变化趋势

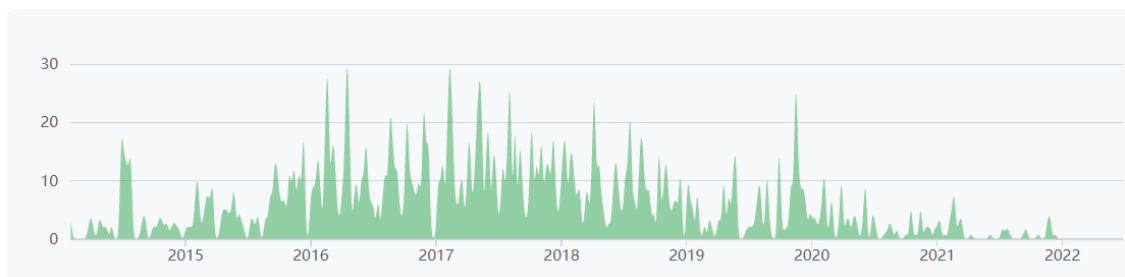
Sourcetrail 只有36位开发者。说明大家对这个项目并不感冒。

The Roslyn .NET compiler (roslyn) 目前有514位贡献开发者。以后还会有更多的开发者加入。

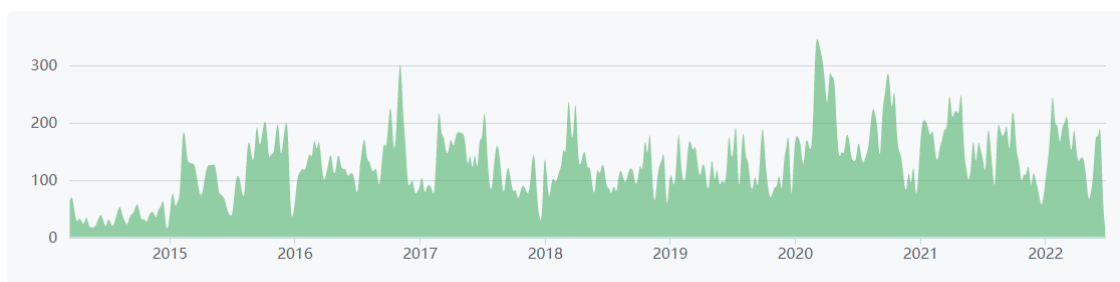
- 其他你感兴趣的对比方向

两个项目的延续时间，可以看到：

Sourcetrail 是从2014年开始，但到2021年就归档了。



The Roslyn .NET compiler (roslyn) 是从2015年发表了第一个版本，但一直到现在（2022年）还在延续。



# 项目发展到活跃/归档的主要影响因素及原因

---

最后，给出我们小组认为的项目发展到活跃/归档的主要影响因素及原因：

## 期中作业分析：归档项目-Sourcetrail

该项目由 Sourcetrail 的原始作者和维护者于 2021 年底存档。

在 2019 年秋季将 Sourcetrail 转换为开源项目后，他们的这个项目最初得到了很多关注，并且非常有动力继续开展该项目。在他们的前几个季度发布时保持了良好的时间表，但很快进展变得缓慢，最终不再对电子邮件、新问题和拉取请求（PR）做出响应。

有几个原因：

- 1.关闭他们的初创公司后，他们将每天的工作时间投入到新工作中。
- 2.然后在 2020 年春季，公司的老板从萨尔茨堡搬到了维也纳，这使得合作变得更加困难，因为他们的成员不再同处一地。
- 3.慢慢地，重新开始该项目的工作也成为一项挑战。毕竟，它一直是一个雄心勃勃的软件，具有许多技术依赖项，需要持续关注以保持最新版本的最新状态。
- 4.特别是考虑到 Sourcetrail 在多个平台上运行，支持多种语言和构建系统，并与不同的 3rd 方工具集成。总而言之，这使得维护和恢复变得相当困难。

最终，他们的创始者也失去了在语言分析和软件可视化领域工作的兴趣，并开始将重点放在新的领域。虽然他们所取得的成就感到自豪，但在了解了创建开发工具的现实挑战，并且知道它需要大量的工作，这些工作往往令人沮丧，而且根本不好玩。

由于所有这些原因，他们决定停止该项目。

但他们在2021年秋季发布 Sourcetrail 2021.4 的最终版本。之后，将网站离线并将存储库存档在 GitHub 上，这实际上冻结了git 存储库、问题列表和 wiki。但是，它仍然允许任何人查看存储库、分叉并下载任何托管版本。

究其原因，我认为是开源项目最需要的是大家的共同合作，一个人很难完成一个巨大的工作，因此，合作共赢是一个团队应该做的事情。

## 期末作业分析：活跃项目-The Roslyn .NET compiler（roslyn）

综上所述，该项目一直在活跃，而且就在刚刚还有新的CI/CD。这就足以证明该项目的活跃性。

90 Active pull requests


80 Active issues

40  
Open pull requests

41  
Closed issues

39  
New issues

Class	Number of Images
1	21
2	18
3	11
4	7
5	6
6	4
7	4
8	4
9	3
10	2
11	2
12	2
13	2
14	1
15	1
16	1
17	1
18	1
19	1
20	1

 Merge main to main-vs-deps  
#62182 merged 2 hours ago

- .NET Compiler Platform SDK 极大地降低了创建以代码为中心的工具和应用的门槛。它将在元编程、代码生成和转换、C# 和 Visual Basic 语言的交互式使用，以及在域特定语言中嵌入 C# 和 Visual Basic 等方面，创造许多创新的机遇。
- .NET Compiler Platform SDK 使你能够生成分析器和代码修补程序，用于发现和更正编码错误。分析器可理解语法（代码的结构）和语义，从而检测应更正的做法。代码修补程序建议一处或多处修复，以修复分析器或编译器诊断发现的编码错误。通常情况下，分析器和关联的代码修补程序一起打包在一个项目中。
- 分析器和代码修补程序通过静态分析来理解代码。它们既不运行代码，也未带来其他测试方面的好处。但是，它们可以指出经常导致 bug 的做法、无法维护的代码或标准原则冲突。
- 除了分析器和代码修补程序外，.NET Compiler Platform SDK 还允许你生成代码重构。它还提供了一组 API，可便于检查和理解 C# 或 Visual Basic 代码库。由于可以使用这一基准代码，因此能够利用 .NET Compiler Platform SDK 提供的语法和语义分析 API，更轻松地编写分析器和代码修补程序。从复制由编译器执行的分析的繁重任务中解放出来，可以将精力放在为项目或库查找常见编码错误并进行修复的更为重要的任务上。
- 这带来的一个小小的好处是，分析器和代码修补程序较小，在 Visual Studio 中加载时占用的内存比为了理解项目中的代码而编写自己的基准代码占用的内存少得多。利用编译器和 Visual Studio 使用的相同类，可以创建自己的静态分析工具。也就是说，团队可以使用分析器和代码修补程序，而不会对 IDE 的性能造成显著影响。