鱼眼视图的源代码应用

——Visual Studio编辑器扩展

软件工程 齐笑尘 指导教师 邬江

【摘要】 软件开发是一项高度复杂的劳动，为了提高编写程序、开发软件的效率，改善开发人员的体验，来自各个领域的许多理论和技术被应用到编程工具和软件开发环境当中，其中应用较多的理论来自人机交互领域。本设计基于信息可视化方向的鱼眼视图理论，通过设计实现主流的软件开发环境Visual Studio编辑器的扩展应用，探究鱼眼视图扩展在该开发环境中的可行性与实际效果。根据需求，设计选择了开发环境本身提供的辅助效果较少的C/C++语言。在设计过程中受到较多技术上的限制，发现Visual Studio提供的扩展性无法完全满足参考文献中鱼眼视图的设计。利用开发环境提供的支持，最终的扩展实现了鱼眼视图主要的框架和基本功能。得到的结论是鱼眼视图的源代码应用与编程语言本身特性有关，在Visual Studio平台下针对C/C++语言的鱼眼视图的完美实现有一定难度。如果开发平台可以提供更加强大的用户界面接口，同时配合相关语言的前端解析支持，鱼眼视图会在Visual Studio平台上提供给用户一个体验更好的浏览编辑代码环境。

【关键词】 鱼眼视图 VisualStudio 扩展

【Abstract】Software development is of high complexity. In order to improve the productivity of programming and software development, theories and techniques from many fields has been applied to the programming tools and integrated development environments(IDE), most of which are from the field of human-computer interaction. This project is based on the Fisheye Visualization theory, and through designing and implementing an extension for Visual Studio editor to explore the possibility and actual effect for applying fisheye to this IDE. The research and development process got quite a few constraints from the techniques available, and the expected fisheye effect is not fully supported by the IDE extensive functionality. The final application includes mainframe of the fisheye design and basic functions. To conclude, the fisheye view effect for source code is closely related to the programming language. If Visual Studio could provide more powerful user interface extensibility and corresponding programming language forefront parser, the fisheye view would be more helpful for browsing and editing code.

【Keywords】FisheyeView VisualStudio Extension

目录

[1 引言 4](#_Toc324453901)

[1.1 源代码应用鱼眼视图的背景 4](#_Toc324453902)

[1.2 现有的一些鱼眼视图研究 5](#_Toc324453903)

[1.3 本课题的研究范围及应解决的主要问题 5](#_Toc324453904)

[2 鱼眼视图理论 5](#_Toc324453905)

[2.1 鱼眼视图的核心思想 5](#_Toc324453906)

[2.2 兴趣指数（Degree of Interest） 6](#_Toc324453907)

[2.3 源代码展示中的具体问题 6](#_Toc324453908)

[2.3.1 源代码的结构抽象 6](#_Toc324453909)

[2.3.2 源代码展示空间的利用 7](#_Toc324453910)

[2.3.3 视觉焦点 7](#_Toc324453911)

[2.3.4 交互时的阈值 7](#_Toc324453912)

[2.3.5 兴趣指数各部分的权重 8](#_Toc324453913)

[2.3.6 实际情况下算法的效率 8](#_Toc324453914)

[3 编程语言相关性 8](#_Toc324453915)

[3.1 编程语言特点与鱼眼视图实现 8](#_Toc324453916)

[3.2 抽象语法树（Abstract Syntax Tree） 8](#_Toc324453917)

[3.2.1 源代码的树形抽象 8](#_Toc324453918)

[3.2.2 编程语言特点与抽象语法树的生成 9](#_Toc324453919)

[3.2.1 一种简单的替代方法 9](#_Toc324453920)

[3.3 鱼眼视图应用的需求 9](#_Toc324453921)

[4 VS2010编辑器的扩展性与鱼眼视图的设计 9](#_Toc324453922)

[4.1 VS2010的扩展性 9](#_Toc324453923)

[4.1.1 概述 9](#_Toc324453924)

[4.1.2 Visual Studio的主要扩展方式 10](#_Toc324453925)

[4.1.3 编辑器扩展接口 10](#_Toc324453926)

[4.2 鱼眼视图的设计 11](#_Toc324453927)

[4.2.1 原型 11](#_Toc324453928)

[4.2.2 显示布局 12](#_Toc324453929)

[4.2.3 交互操作 12](#_Toc324453930)

[5 扩展应用的实现 13](#_Toc324453931)

[5.1 应用架构 13](#_Toc324453932)

[5.2 分模块实现 14](#_Toc324453933)

[5.2.1 源代码解析器 14](#_Toc324453934)

[5.2.2 鱼眼视图的生成 16](#_Toc324453935)

[5.2.3 代码行效果标签 18](#_Toc324453936)

[5.2.4 响应交互操作 18](#_Toc324453937)

[5.2.5 全景图显示区（Overview） 18](#_Toc324453938)

[5.2.6 焦点区域（FocusArea）以及各组件的交互 20](#_Toc324453939)

[6 结论 20](#_Toc324453940)

1. 引言
   1. 源代码应用鱼眼视图的背景

软件开发（编写程序）是一项复杂的智力密集型劳动，是至今为数不多的机器不能够有效地代替人完成的工作之一。随着硬件的升级，软件系统规模越来越大，复杂程度也越来越高，编写程序与维护程序的成本也随之加大。

尤其是大型软件的开发，往往需要耗费大量人力和时间。即使软件一般都会有优良的设计和实现，开发人员也具备丰富的经验，但是，由于代码的数量很大、结构极其复杂，浏览和理解代码对开发人员来说是一个挑战，如果没有相应的展示增强工具，开发效率会受到严重挑战。我在实习工作中参与了一个大型商业软件的开发，使用Visual Studio经常会面对单文件几千行源代码，代码的浏览效果不尽如人意。对于较长的代码，开发人员无论是浏览还是添加新的代码都需要灵活地跳转代码的位置，同时也希望能够浏览到尽量多和重要的代码信息来节约时间。

和源代码展示相关的理论研究领域是信息可视化(Information visualization)，该领域主要关注大规模信息的可视化展示，比如软件中的文件和源代码、图书馆和文献系统数据库、网络与因特网。其中，鱼眼视图(Fisheye views)是一种比较成熟的理论，具有突出重点、节省展示空间的特性。最早提出信息展示和鱼眼视图结合理论的是George W. Furnas，他的设定的鱼眼视图中，通过比较代码行的重要程度来决定突出显示重要代码或者隐藏非重要代码。

* 1. 现有的一些鱼眼视图研究

鱼眼视图其实是受自然界生物的启发，也符合人类视觉效果的一般习惯，即离的越近的物体和主观上越关注的物体越容易察觉到。人类模拟鱼眼制造出了鱼眼镜头产生的就是这样夸张的效果，在有限的展示范围内放大易于捕捉的和希望捕捉到的画面，同时缩小甚至略去难捕捉的和没有兴趣捕捉的画面。

在信息可视化领域，我们接触的较多的都是鱼眼视图在一般人机交互界面的应用。比如Mac OS X的Dock，一些手持设备的图片浏览器，一些软件的复杂菜单等。上述这些例子多为提供一种视觉效果，是作为一种吸引用户的手段出现的。而本文的主题，展示源代码的鱼眼视图应用，自从Furnas于1986年首次提出以来，在主流集成开发环境中的应用却没有预期中普遍。

* 1. 本课题的研究范围及应解决的主要问题

从程序开发人员的实际的角度来看，更高效地浏览、编辑源代码的需求应该比一般用户浏览工具栏图标和菜单项要迫切。本课题基于如下的情境：使用Visual Studio查看、编辑编写规范的C/C++源代码文件。这也是我在实习工作中的实际任务需求。

一方面，本课题需要学习鱼眼视图的基本理论，研究VS2010所提供的扩展开发环境，结合其特点来设计实现VS2010编辑器下的鱼眼视图扩展；另一方面，在实现过程中探究在该开发环境下鱼眼视图扩展应用的可行性与实用程度，提出该设计可能存在的一些问题，来解释鱼眼视图未能如预期应用到源代码展示中的原因。

1. 鱼眼视图理论
   1. 鱼眼视图的核心思想

计算机科学以及互联网技术的不断发展使得人们每天面对的信息量呈爆炸式增长，但与此同时用于展示、获取这些信息的交互设备并没有产生相应规模的进化。这主要还是受限于人本身获取信息的特点和能力。鱼眼视图理论就是在这种背景下产生的，提出该理论的是一位具有认知心理学背景的计算机科学家，George W. Furnas，他通过跨学科的研究阐释了鱼眼视图的原理和应用构想。

鱼眼视图在视觉上很像鱼眼镜头——距离视觉焦点近的部分被夸张放大，而距离焦点远的部分被缩小、减少细节甚至略去，但是仍然完整地展示一个范围。鱼眼视图提出的动机是为了在获取信息时能兼顾局部细节和整体结构。这种需求本质上符合人类认知事物的特点，其结构也与信息在人脑中的组织相似。举例来说，在一个公司内部，新员工会很快熟悉自己所在部门的同事，但是对于其他部门，开始只是对其领导有印象；地方性的报纸报道的信息，和本地相关的内容涵盖各个方面，比如本地蔬菜价格上涨，但是世界范围内只会选择诸如朝鲜发射火箭这样重要的内容报道。因此，在人类和外界的交互中，自然地就体现出鱼眼视图的特点。如果可以将信息以鱼眼视图的形式展示出来，用户会更容易理解整体并且高效地获取自己所需的部分。

* 1. 兴趣指数（Degree of Interest）

鱼眼视图通过引用一个称为“兴趣指数”（Degree of Interest）的量化标准作为计算标准。按照Furnas提出的公式：

DOIfisheye(x|.=y) = API(x) – D(x,y)

该值由两部分计算得出，其中优先重要性(A Priori Importance)是x点在视图中的绝对重要程度；D(x,y)是x点和视觉焦点y的距离。根据公式，兴趣指数和优先数量级呈正相关，与距离呈负相关。

我们将计算机中数据信息组织形式统一抽象为树形结构来具体应用上述计算方法。其实不难发现计算机系统中树形结构还是比较常见的：最容易想到的就是操作系统中的文件系统，除此之外还有目录和菜单等，对于结构化的编程语言也是以树形结构组织的。（源代码经过编译器前端解析，会生成抽象语法树，继而经过后端进行优化并生成机器码。）

在树形结构中，以两个节点的路径长度表示两点间距离，以某节点的深度（距离根节点的路径长度）表示其优先重要性，得到公式：

DOIfisheye(tree)(x|.=y) = -(dtree(x,y) + dtree(x,root))

通过该公式计算出树形结构中每个节点的兴趣指数之后，我们可以设定一个阈值来限制鱼眼视图最终展示的部分。比如，如果阈值为当前聚焦节点的兴趣指数，那么阈值筛选出的节点集合就是从当前节点到根节点的路径上的所有节点，也就是当前节点以及它的直系祖先；进一步如果缩小阈值的限制，我们将进一步得到当前节点的子节点、亲兄弟节点和表兄弟节点等。

* 1. 源代码展示中的具体问题

Furnas文中的举例比较有局限性，如果要将鱼眼视图应用到源代码展示尤其是当今的集成开发环境中，还有更多的问题待解决。

* + 1. 源代码的结构抽象

在Furnas论文列举的源代码展示例子默认将源代码按行划分，每行在树形结构中作为一个节点。图 1为Furnas论文中展示的源代码的鱼眼视图：

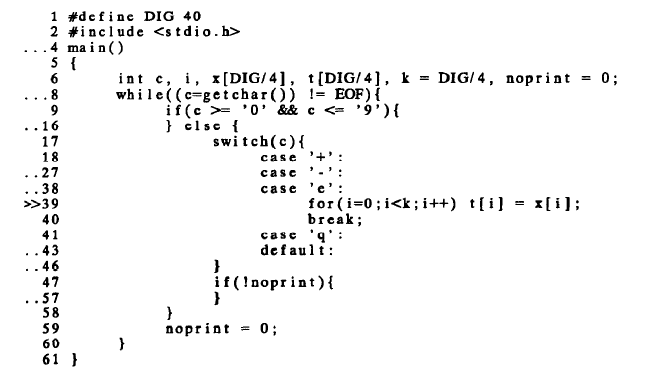


图 1

将源代码以行位单位构成一棵树是最直观的做法，但是实际应用还是需要和程序语言本身的特点结合起来，具体将在第三章讨论。

* + 1. 源代码展示空间的利用

在Furnas论文列举的源代码展示例子中，情况是比较理想的——程序较为简短、层次非常分明，展示结果能够完美地展示鱼眼视图的优势。但是实际的源代码可能较长，组织会非常复杂，在有限的编辑器展示空间内，未必能够理想地将局部细节和全局结构都展示出来。假设当前聚焦的代码行的优先重要性为一般，展示的两种极端情况为：力求展示整个文件，那么显示出的代码行都会是优先重要性较高的（比如函数名、类名），和聚焦行重要性相似的代码行（比如文本中的相邻行）无法显示；适当展示细节（至少展示出聚焦行的兄弟节点行），这样页面无法容纳整个文件的结构。有限的空间导致单纯基于兴趣指数和阈值生成的鱼眼视图缺乏灵活性。

* + 1. 视觉焦点

虽然鱼眼视图提出的兴趣指数的两个组成部分：优先重要性和距离，可以在多种信息结构中找到对应的定义，但是计算这两者的基础还是要获取当前用户的视觉焦点，但是这一个问题在Furnas的论文中没有特别阐明，他只是在例子中指定了某一行作为焦点。同时需要考虑的还有用户交互的行为：仅浏览代码可能存在一种计算当前焦点的方法，但是除了浏览代码另外很重要的功能是编辑代码，用户会使用鼠标和键盘来控制光标，是否将此作为计算焦点会在很大程度上影响用户的体验。当前的很多开发环境还提供了其它一些编辑器相关的交互操作，如Visual Studio中对C#语言中region提供的大纲视图，用户可以选择隐藏或者展开代码块，这些操作也直接影响到视图的变化。

* + 1. 交互时的阈值

如果能够确定视觉焦点，兴趣指数就可以计算出来，下一步就是将兴趣指数与阈值进行比较。按照Furnas进行的演示，阈值可以限制鱼眼视图的规模，但是他并没有提出一种交互时的阈值策略。阈值反应了用户对于局部或全局的感兴趣程度，如果固定一个阈值，最严重的问题是我们无法事先知道源代码的综合优先重要性水平（代码行的深度），如果有些代码行（节点）的兴趣指数比较小甚至低于我们设定的阈值，那么这些代码行就永远无法被显示了；即使文件中兴趣指数极小的代码行可以被显示，但是距离其较近的一些代码行可能会由于在结构树种的路径距离较远而无法显示，这对于希望看到当前焦点行附近细节的用户来说无疑是不友好的。

* + 1. 兴趣指数各部分的权重

另外可以观察到随着阈值的减小，优先重要性本身就比较高的代码行更容易被显示，其次才是旁系的祖先或兄弟——这样显示出的结果总是一棵保留焦点和根节点之间主干的子树，逐渐显示出来的也是靠近这个主干的节点。焦点行的直系祖先的兴趣指数是和焦点行完全一样的，这可以理解为，既然焦点行被显示出来了，那么他的直系祖先行也默认要被显示出来。而在距离相等的情况下，优先重要性扮演了绝对重要的角色。比如，焦点行在文本中的相邻行在树中是其兄弟节点，其路径距离和焦点行与其祖父节点相同，但是兴趣指数却少了2点，这使得其在规模最小的鱼眼视图中无法被显示：很多情况下用户并不希望看到这种显示，在代码中我们往往不会仅满足于一行的细节，而是关注其相邻几行。因此Furnas提出的兴趣指数计算方法还可以通过增加不同部分的权重来得到改良。

* + 1. 实际情况下算法的效率

一个局限性体现在Furnas的例子是静态的，他也没有提出具体的计算算法，只是提出鱼眼视图的算法可以保持在一个较低的复杂度（复杂度是基于显示的结果而非整个树）。本课题将研究鱼眼算法在开发环境中的效率问题。

1. 编程语言相关性
   1. 编程语言特点与鱼眼视图实现

非结构化编程语言包含按顺序排列的命令或声明，通常每个都占用一行。每一行都有编号或者标签，这样程序中的任意行都可以被执行。非结构化程序设计引入了基本控制流的概念，比如循环、分支和跳转。尽管在非结构化模式中不存在过程，不过子程序还是可以使用的。和过程不同，子程序可能有多个入口和出口。 子程序中是允许直接跳转入或跳转出的。典型的非结构化编程语言有早期的BASIC、COBOL和汇编语言等。

我们平时接触较多的C/C++、Java都是结构化编程语言，也称过程化编程语言。其最大特点是采用子程序（Subroutine）、程式码区块（Block structures）、for循环以及while循环，来取代传统的 goto语句。希望借此来改善计算机程序的明晰性、品质以及开发时间，并且避免写出混乱的面条式代码。

显然，结构化编程语言不仅有更好的实现鱼眼视图的基础，而且实现的意义也更大。结构化编程语言一般是以树形结构组织的，而非结构化编程语言抽象出来更像是一种有环路的有向图，并不在Furnas提出的支持鱼眼视图应用的数据结构类型中。

* 1. 抽象语法树（Abstract Syntax Tree）
     1. 源代码的树形抽象

能够最准确地反映源代码抽象语法结构的是抽象语法树，这也是和代码文本关联较强的一种树形结构。抽象语法树上的每个节点都表示源代码中的一种结构。之所以说语法是“抽象”的，是因为这里的语法并不会表示出真实语法中出现的每个细节。比如，嵌套括号被隐含在树的结构中，并没有以节点的形式呈现；而类似于if-condition-then这样的条件跳转语句，可以使用带有两个分支的节点来表示。

抽象语法树是在编译器前端进行语法分析后得到的，在编译器后端在语义分析后会产生另外一种树形结构——分析树。但是和抽象语法树相比，分析树不适合作为生成鱼眼视图的“宿主”结构。鱼眼视图所展示的内容本身不一定是直观的树形结构，但是在计算时底层至少有相应的数据结构。正如源代码展示，展示项是代码行，代码行可以比较容易地映射到抽象语法树中的节点；但是对于分析树而言，其中的每一个节点对应的是源代码中的语法细节（比如一个操作符），因此同样是一段代码在编译过程中生成的树形结构，抽象语法树是最理想的生成鱼眼视图的基础。

* + 1. 编程语言特点与抽象语法树的生成

经过查阅、学习其他已有的源代码的鱼眼视图展示的文献，我发现已有的原型是基于Java和C#的，而没有C/C++相关的。集成开发环境为Java和C#提供了抽象代码树的接口支持，但是较少的有直接支持C/C++语言的。这两类编程语言比较突出的区别是编译过程，C/C++是编译型语言；Java与C#都不是编译型语言（Java是解释型，而C#是一种混合的形式）。Java与C#的一个共同特点就是中间语言（Java是Java字节码，C#是CIL），使用中间语言不仅仅可以实现平台无关性或是高级语言的互操作性，还有助于性能优化和代码安全性。相对比，C/C++没有类似的中间代码，因此我猜想C/C++的编译器前端对抽象语法树的支持不如Java、C#般完美，一方面有语言功能强大、语法设计庞大复杂的原因，但主要还是由于没有相应的中间代码。虽然微软Visual Studio开发组C++编译器前端的开发人员很早就提出了“让每一个人用上C++抽象语法树API”[[1]](#footnote-1)的目标，但是目前版本的Visual Studio中还是没有这方面的接口。虽然一些编译器，比如gcc、clang提供功能较强的抽象语法树支持，但是这些库因为没有提供managed code接口而无法直接在Visual Studio中被使用。

* + 1. 一种简单的替代方法

鉴于本课题将关注的C/C++语言在目标开发环境中没有十分有效的抽象语法树支持，我会选择一种简单的替代方法来模拟生成一个C/C++文件的抽象语法树。

这种方法是基于文本特征的，比如LISP中的括号、Python中的缩进等。在编写规范的源代码中通过这些文本特征还是可以生成一个足够鱼眼视图使用的树形结构的。针对C/C++语言可以使用的有花括号和缩进，考虑到在编辑代码过程中缩进对代码结构的影响较小（花括号在编辑时会经常发生不匹配的情况，这种不匹配是不可避免的，因而造成显示的不稳定或者错误），我将实现一个基于缩进判断的简易解析器，Furnas文中演示的代码也是以缩进来组织树形结构的。

* 1. 鱼眼视图应用的需求

基于实习期间主要接触的开发情境，本课题将开发一个Visual Studio 2010编辑器的扩展应用，主要功能是实现C/C++源代码文件的鱼眼视图显示。由于本课题倾向于探究该应用的实现可行性和实用性，因此前提假设是C/C++源代码文件的编写是符合规范的。

具体需要实现的效果应该具备Furnas论文中所展示的鱼眼视图的主要特征，为浏览代码提供便利。在实现鱼眼视图显示的同时应保证浏览的流畅度，即鱼眼视图的计算不应该占据过多的资源以致于影响编辑器和用户交互的性能。

1. VS2010编辑器的扩展性与鱼眼视图的设计
   1. VS2010的扩展性
      1. 概述

Visual Studio作为微软的产品，在开发者眼中也和微软其他产品一样——封闭性强而扩展性差。确实，和另外一个开源集成开发环境项目Eclipse相比，Visual Studio不会像Eclipse一样衍生出Rational和MyEclipse这样定制性极高的二次开发平台。虽然微软的文化和哲学决定了Visual Studio的扩展性和可定制性不如开源项目，但是我们要承认Visual Studio还是提供了一套供各个需求层次使用的的较完整的扩展接口。使用较多的Visual Studio扩展有Visual Assist X、NuGet Package Manager、MetalScroll等，微软也提供了一个在线市场一样的平台供开发者发布扩展应用，其中很多实用的应用都是免费的。

* + 1. Visual Studio的主要扩展方式

1. 宏

宏是组合到一起形成一个命令以自动完成某项任务的一系列命令和指令。微软在Office系列中就加入了宏功能，来便于批量化和自动化操作。宏在Visual Studio中是最简单的一种扩展方式。

1. 外接程序和向导

外接程序就是一般意义中的插件，由 COM 对象构成，可实现 Extensibility 接口，并通过 EnvDTE 程序集中包含的核心自动化对象模型来与 IDE 通信。 开发时主要以调用Visual Studio命令为主，开发语言是支持COM的语言。外接程序比宏的功能要强大，比如可以创建工具窗口、将加入的功能部署在工具栏中等。我在最初进行可行性分析时尝试了外接程序的功能，但是因为其对于外观效果是无法进行自行定制的，因此需要更强大的扩展接口来实现鱼眼视图的应用。

1. Visual Studio SDK

VS SDK是一个开发人员套件和平台。 它通过提供对更多接口的访问，使开发者能够更细致、更全面地控制 IDE，进而提供功能更强大的扩展应用。套件中包含了很多功能库，因此需要另外下载。其中又按功能模块分成了诸如文本编辑器、文档视窗、编程语言服务、项目管理等，几乎涵盖Visual Studio的所有功能大类。

另外微软还提供了一种Visual Studio Partner计划，加入该计划的开发者会获得更强大的技术支持。比如开发Visual Assist X的Whole Tomato就是其中一员。对于个人开发者来说仅仅是VS SDK本身应该在大多数情境下会满足需求了。

* + 1. 编辑器扩展接口

结合本课题的需求特点，我主要使用Visual Studio SDK中和代码编辑器相关的接口来进行应用的开发。编辑器扩展接口的最重要技术是Managed Extensibility Framework (MEF)——可控扩展框架[[2]](#footnote-2)。这个技术是.NET 4.0最新加入的，其主要目的提供一种更优化的框架供可扩展应用程序和组件的设计使用。从Visual Studio 2010开始微软倡导使用该技术进行扩展应用的开发，对于此前版本Visual Studio编辑器的扩展接口微软提供了适配器供新接口调用。

1. 可扩展性存在的问题

大型软件系统中提供可扩展性是十分必要的，系统可能会运行一些潜在的组件。一种简单的实现就是将这些组件作为源代码包括在应用程序中，然后通过代码直接调用它们。这种做法存在很多明显的缺陷。最重要的是，开发者无法在不修改源代码的情况下添加新组件，这一限制在 Web 应用程序中也许能够接受，但在客户端应用程序中行不通。同样存在问题的还有，开发者可能没有对组件的源代码的访问权，因为这些组件可能是由第三方开发的，而出于相同的原因，宿主程序也不允许第三方访问自己的代码。

一种有难度的实现是：提供扩展点或接口来允许应用程序和其组件相分离。依据此模型，程序员可能会提供一个组件能够实现的接口，并提供一个 API 以使该接口能够与应用程序进行交互。在这种情况下程序依然缺乏自己发现组件的能力，需要通过配置文件来维护一个组件列表。由于没有定义好的接口，组件之间也较难实现通信。

1. MEF的功能特性

MEF提供了隐式发现组件的方法——在组件的声明中定义依赖项和提供的功能。MEF有一个组件容器对象来组织管理组件的导出与导入。这一方法解决了上面讨论的问题。由于 MEF 组件以声明方式指定其功能，因此在运行时可发现这些组件。这意味着，应用程序无需硬编码的引用或脆弱的配置文件即可利用相关组件。通过 MEF，应用程序可以通过组件的元数据来发现并检查组件，而不用实例化组件。因此，没有必要仔细指定应何时以及如何加载扩展。

除了组件提供的导出以外，组件还可以指定其导入，然后由其他组件填充这些导入。 这不仅使各组件之间的通信变为可能，而且使通信变得很容易，此外，还可以合理地分解代码。使用 MEF 编写的可扩展应用程序会声明一个可由扩展组件填充的导入，而且还可能会声明导出，以便向扩展公开应用程序服务。 每个扩展组件都会声明一个导出，而且还可能会声明导入。 通过这种方式，扩展组件本身是自动可扩展的。因此也有将MEF翻译成“自动发现式扩展框架”的。

* 1. 鱼眼视图的设计
     1. 原型

通过了解Visual Studio的扩展接口功能，鱼眼视图应用的可行性得到了初步验证。现有的可参考的原型一是Mikkel Rønne Jakobsen与Kasper Hornbæk在他们的Evaluating a Fisheye View of Source Code一文中基于Eclipse实现的Java源代码鱼眼视图扩展；另一个是Thomas Ren´e Sidor依靠简易C#解析器实现的鱼眼视图源代码浏览器。虽然依赖的环境和显示的目标语言相差较大，但是这两个应用在界面设计上看是相似的，图 2是Sidor开发的原型：

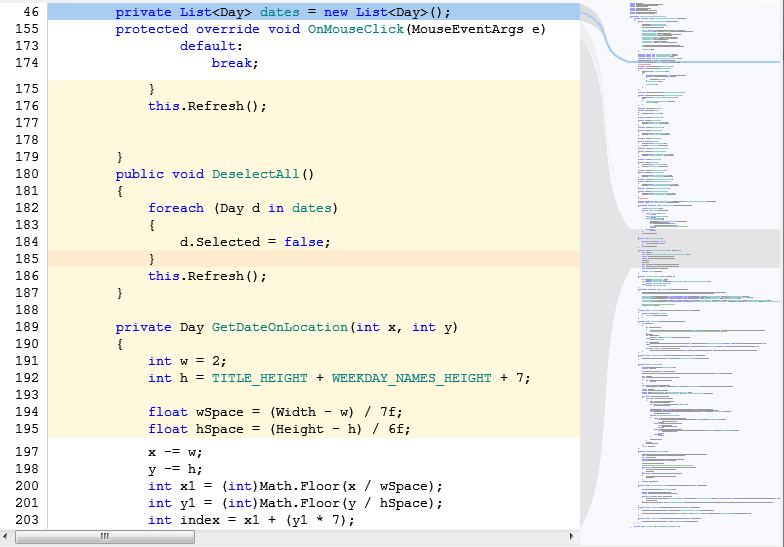


图 2

* + 1. 显示布局

本应用在展示布局的设计上借鉴已有的原型，将编辑器视窗分为三个区域（结合图1说明）：左侧较大的显示区域还是源代码展示区，源代码以行为单位显示或隐藏，构成鱼眼视图；右侧狭长的区域为源代码的“全景图”展示，该区域相当于把整个源文件缩印了一份显示；“鱼眼视图区”和“全景图区”中间的区域绘以曲线表示两者之间的关系，也就是左侧源代码在右侧全局结构中的位置。

特别值得指出的是，左侧的鱼眼视图区与Furnas在他的论文中提出的不同：该显示区域内中部为特别改进的“焦点视图”区域，即第175至195行（在图中以颜色区别），可以观察到，该区域内代码行号是连续的，也就是说每以行代码均被显示；与之对应的是其上下背景为白色的区域，在这两个区域内的代码行号不是连续的，有些代码行被隐藏——这才是传统意义上的鱼眼视图。

整体的显示效果相比较于Furnas论文中的设计，增加了对全局的展示（全景图区域），同时也加大了对局部细节的展示程度（焦点区域）。这种设计从根本上解决了在第二章中针对原论文提出的一些问题：首先就是对展示空间的利用（见2.3.2 ），在整个视图中显示了两种极端的情况，可谓“鱼和熊掌兼得”；进而阈值（见2.3.4 ）和兴趣指数公式（见2.3.5 ）对优先重要性较低的附近行的限制也就比较小了。

* + 1. 交互操作

该扩展的应用范围是编辑器，仅支持与编辑器交互的用户行为。按照显示布局，考虑到原始的滚动条占据空间，且使用不够直观，本应用略去了原始的滚动条。划分为两类：

1. 浏览类操作

　　包括鼠标滚轮的滚动、鼠标在右侧全景图的移动与点击。这类操作会将用户导向源文件中相应的位置。由于略去了原始滚动条，代替滚动条行为的是全景图。鼠标点击选中全景图中的微缩代码行会使源代码显示区域跳转至对应的代码行。

1. 编辑类操作

　　鼠标在源代码显示区域的点击、键盘的操作。除了编辑器本身对这类操作的响应，扩展应用会将视图进行相应的位移，以满足快速跳转的需求。

1. 扩展应用的实现
   1. 应用架构

借助于Visual Studio提供的MEF框架，本应用的架构也呈现出一种松耦合的特点。基于MEF的组件和VS的关系如图 3：

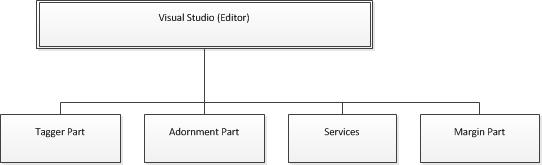


图 3

VS2010编辑器本身就是由MEF组件构成的，我们进行扩展开发出来的组件和VS2010自身的组件是一个性质的。VS2010内部会通过组件容器（ComponentContainer）来将当前组件目录（ComponentCatalog）中的组件执行组合。每一个称为组件可以组合（导入）其它组件的功能（其它组件通过声明Export提供功能）并且它也可以通过定义Export将其功能暴露给其它组件。图 4[[3]](#footnote-3)展示的是其抽象原理：

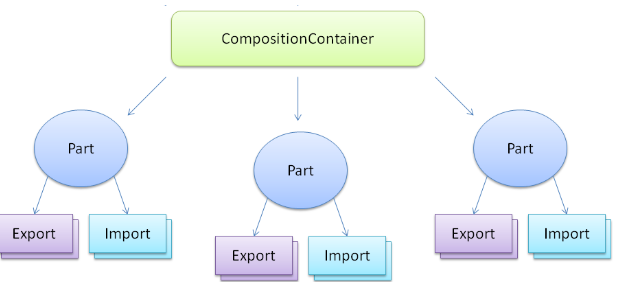


图 4

对于VS2010来说，它可以导入用户开发的符合其定义类型的组件，比如图 3中的IWpfTextViewMarginProvider（Margin part）就是VS2010定义的编辑器视图边缘，宿主将编辑器中的某些对象（比如ITextView，这是当前代码视图对象）传递给组件，其实现的预先定义的接口可以被VS2010调用。该组件还可以导入（英文为consume）其他组件，反过来亦可被其他组件导入。图 3中的Services也是VS2010提供的组件，只不过不是直接将组件对象暴露给其它组件。

图 3显示了构成本应用所需的几种组件，其中边缘（Margin）组件主要用于实现编辑器右侧的全景图；装饰（Adornment）组件用于显示编辑器左侧鱼眼视图的“焦点区域”；标签（Tagger）组件实现了不止一种，分别提供不同的代码显示效果。

除了基本的组件类，我将模拟生成抽象语法树的解析模块、鱼眼视图计算算法模块还有操作VS2010接口的工具模块进行了封装，为组件部分调用。

* 1. 分模块实现
     1. 源代码解析器

源代码解析的目的是模拟生成一个源文件的抽象语法树。本应用面向C/C++语言，采取基于缩进判断的策略来解析源文件。解析器得到的是ITextSnapshotLine对象集合，也就是当前代码行集合；由于缩进为0的代码行不止一行，所以经过解析产生的是多棵树，如果认为这多棵树的根节点存在一个共同的虚构父节点，也可以把产生的结果看成一棵树。

1. 计算每行代码的缩进

计算缩进是进行解析需要主要的准备工作。对于每一行源代码，使用正则表达式判断当前行非空白字符前的空白字符组成，再根据缩进量预设值（即制表符对应的空格符数）得出当前行的缩进值。在实现本应用时使用的制表符是以4个空格符计，符合常用的开发规范。以下函数实现的就是计算代码行缩进量的功能（图 5），其中以“rx”开头的变量是用来匹配非空白字符、制表符、空格符的正则表达式：

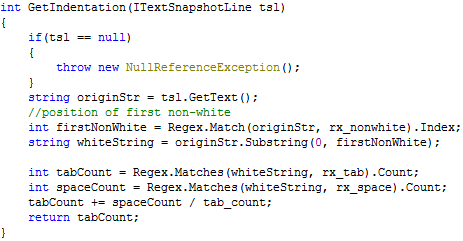


图 5

1. 代码行集合的解析

对于一个代码行集合，应用采取顺序（文本的自上向下）遍历方式，依次根据每行的缩进量和当前树结构的状态进行判断，决定该行代码在树中的节点位置，缩进量直接体现当前代码行的节点深度。解析过程参考对应的有限状态图（图 5）：

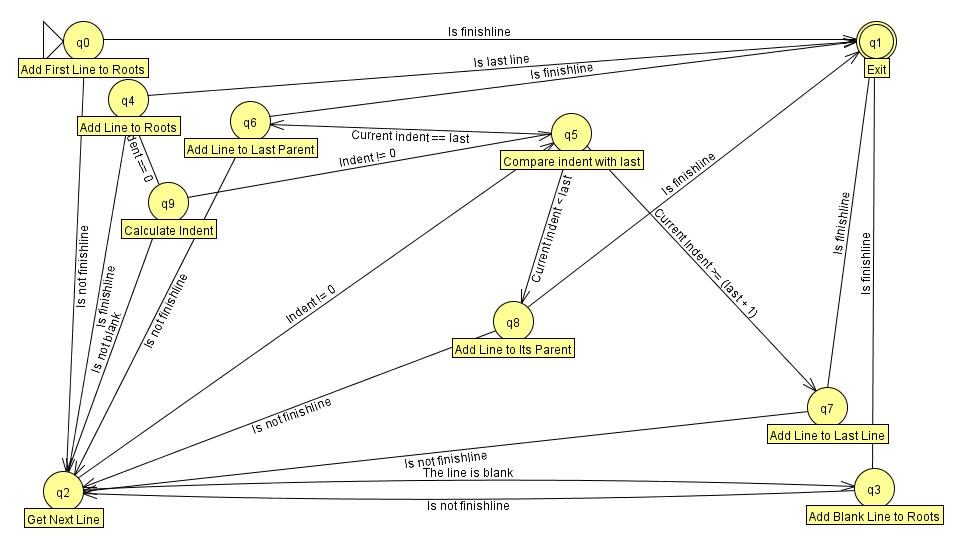


图 6

上图是使用JFLAP绘制的有限状态图，具体说明如下：

1. 起始状态（q0）程序默认得到代码行集合中的第一行，并将其加入到Roots（根节点集合）中，判断是否为最后一行：若是，退出（q1）；若不是，进入循环依次处理剩余的节点（q2）；
2. 先判断该行是否为空白行：若是，将其标记为空白，加入到Roots中（q3）；若不是，计算其缩进量（q9）准备下一步判断；
3. 判断缩进量是否为0，若是，将其加入到Roots中（q4）；若不是，和上一行代码比较缩进量（q5）；
4. 缩进量和上一行相比分三大类情况：缩进量和上一行相等，将当前行加入到上一行的父节点（q6）；缩进量比上一行小，向上回溯找到其父节点，将其加入（q8）；缩进量比当前行大1（大于或等于2的情况不属于预期，但是为了功能的正常将其归于一类），将其加入上一行作为子节点（q7）；
5. 进行完树的添加操作后判断当前行是否为集合的最后一项，若是，退出（q1）；若不是，继续处理下一行（q2）。
6. 生成树的数据结构

为了提高计算鱼眼视图算法的效率，解析器生成的树形结构是双向的，也就是说每个节点（LineEntity）不仅包含子节点集合，也包含父节点的引用。同时存在一个类型是节点引用的数组，此数组按文本中代码行的顺序记录每一行代码对应的节点。整体上是一种空间换时间的策略。类图见图 7，emuParser类包含了解析器的方法还有计算鱼眼视图的方法：

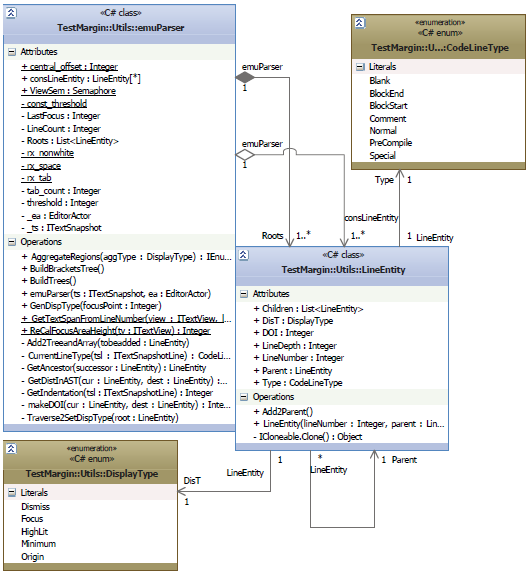


图 7

* + 1. 鱼眼视图的生成

1. 兴趣指数计算

Furnas兴趣指数公式中的优先重要性（节点深度）已经在上一节的解析过程中得到，剩下需要计算的只有节点间路径距离。源代码生成树的特点是同一深度的兄弟节点较多，但是树的高度较小。结合这一特点，计算路径距离的策略是找到两个节点的最深的公共祖先，再将两个节点相对于公共祖先的深度相加。由于生成树是双向的，因此寻找公共祖先的复杂度不高。实现这一计算的方法见图 8：

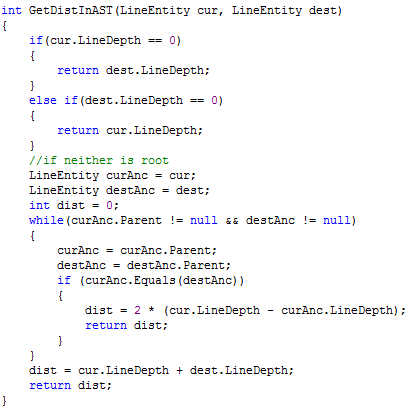


图 8

1. 焦点行的确定

本应用的设计虽然解决了2.3 节中提出的一些问题，但是焦点的确定还是要具体结合VS2010提供的接口来实现。基本策略是取当前视图中点位置的代码行。ITextView是编辑器代码视图的对象，它提供了一个当前被渲染的代码行集合（ITextViewLines）。根据此集合我先后尝试了两种确定焦点行的方法：一开始是获取集合的中间元素，这样得到的结果不够准确（焦点的准确性在本应用中非常重要），因为这个集合所含的代码行有些只是部分显示在视图上；最近又发现了集合对象提供的一种方法，是用来获取涵盖某个纵坐标值的代码行，因此将编辑区中心的纵坐标传递给这个函数就可以获得相对准确的焦点行。事实也证明后面的方法不会带来误差（见5.2.4 ）。

1. 计算阈值

对于此前提出的问题（见2.3.4 ），阈值的绝对化影响到较深的节点无法显示。现在虽然设计中虽然有了焦点区域来保证焦点临近的代码行能够被完整显示，但是如果阈值设定得不合适，焦点区域附近的很多较深节点对应的代码还是无法被显示。因此本设计尝试使用和当前焦点相关的阈值（T）算法：

Tfocus（x） = -dtree(x,root) + Tconst

其中x是当前焦点，阈值是当前焦点的优先重要性和一个预设阈值（Tconst）之和。因为在本设计中，展示全局信息的功能由全景图区承担，所以鱼眼视图区就没有展示全部文件的负担了。对于长文件，展示空间带来的限制比阈值带来的限制要明显，鱼眼视图可以将有限的展示空间分配给与当前焦点行相近的代码行。Tconst应该是允许用户进行设置的，在下面的涉及代码行显示效果的小节（5.2.3 ）我会阐述这样做的其它原因。

1. 代码行显示类型

为了将代码生成树和鱼眼视图代码行（数据与显示）进行分离，我没有在兴趣指数的计算过程中直接操作节点对应的代码行的显示，而是在计算并与阈值比较每个节点的兴趣指数后为节点赋一个显示类型。显示类型（DisplayType，见图 7）是一个枚举类型，每一种对应代码行可能的一种显示效果。通过递归遍历Roots，可以得到出每个节点的显示类型，之后在应用到显示时只需线性遍历引用这些节点的数组，根据其显示类型进行相应的操作。

* + 1. 代码行显示效果标签

* + 1. 响应交互操作
    2. 全景图显示区（Overview）

全景图是鱼眼视图的控制中枢，是本应用的一个特殊设计，功能上替代并且强化了原有的滚动条。类似的Visual Studio扩展应用是Metal Scroll，但是因为它支持的版本只有VS2005与VS2008，对本应用提供的借鉴价值不大（VS2010编辑器使用全新的扩展接口）。和全景图相关的类见图 6：

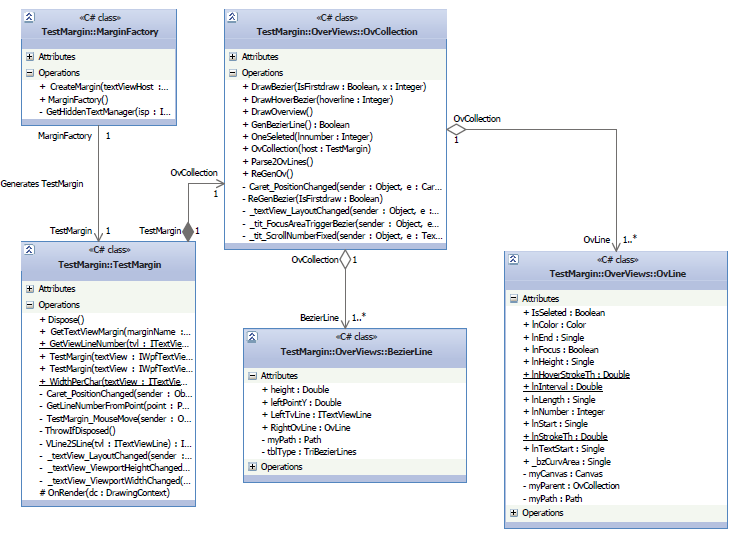


图 9

全景图本身是编辑器左侧的一个边缘对象（TestMargin），它继承了WPF的Canvas类和VS2010扩展接口定义的IWpfTextViewMargin对象。此对象由MarginFactory对象创建并作为组件导出给编辑器。其内部组合了一个OvCollection对象作为可显示对象的容器。全景图中包含两种可显示对象，一种是OvLine，即源文件代码行在全景图中的对应；另一种是BezierLine，用于将全景图和鱼眼视图区相关联。全景图区的实现主要分为以下三个部分：

1. 代码的微缩显示

全景图最基本的功能是将左侧源文件整体微缩显示出来。由于这个需求不具有一般性，Visual Studio没有提供直接将源文件快照的功能；即使得到了源文件的快照，交互功能还是要另外实现。全景图是利用WPF的绘图接口将从编辑器提供的文本视图对象中获取的源文件代码行抽象成线段绘制到面板上的。

原始代码行转化成微缩线段（OvLine）需要结合当前全景图和编辑区整体的尺寸来确定微缩线段的宽度，以下函数实现的就是计算原始代码行中字符宽度的功能（图 7）：

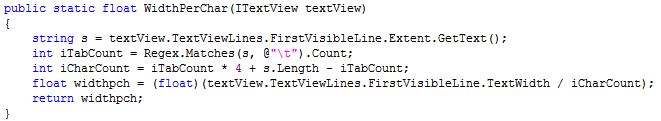


图 10

其余重要参数还包括微缩线段的行间距、全景图和原始显示的宽度比。最终线段的主要起始点由以下公式计算得出（图 8）：其中横坐标与代码行飞空白字符的起止位置相关，还要预留出绘制贝塞尔曲线的区域；纵坐标和代码行号与行间距相关，全景图区的代码行号与鱼眼视图区的一致，以便于两者正常交互。



图 11

微缩展示的一个重要挑战是代码行数过多无法正常显示，一些开发人员有竖屏编程的条件，但是目前主流开发的配置（以我在实习公司的观察）还是较大尺寸的宽屏显示器。考虑到这个问题，我对微缩线段的宽度（行间距）加以限制，防止因为代码行过多影响微缩线段的显示和交互效果。如果代码行过多导致行间距小于预设值，那么仅选择部分代码行绘制对应的微缩线段：比如每相隔1行或者2行进行绘制。这样，整个源文件的代码还是能够较全局地被展示出来。图12展示的是一个较长的源文件被略去一半代码行（奇数行）的全景图，图13展示的是一个较短源文件保留每一行的完整全景图，可见其效果差异不大。

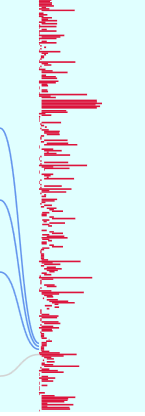
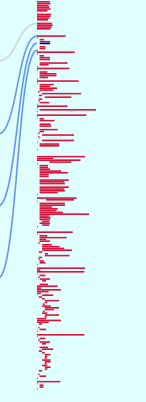
 

图 12 图 13

1. 贝塞尔曲线

常见的表示代码行相互联系是贝塞尔曲线，比如在版本管理系统Perforce版本对比程序、Google Code提供的版本对比页面等。全景图包含4条贝塞尔曲线，其中关联到鱼眼视图的焦点区域的一共3条（上下边界和中心），另外一条用于将鼠标停留的代码行映射到全景图上（以颜色和另外三条区分）。贝塞尔曲线也是由WPF提供的接口、根据指定的4个点绘出，其中第一个点是曲线的起点，第二、三个点是控制点，最后一个点为终点。如图12、图13中的展示：绘制时以鱼眼视图对应代码行的位置为起点，以改行对应的全景图微缩线段位置为终点，控制点都在位于贝塞尔曲线区域的中心位置。

1. 全景图的交互操作

全景图的交互操作包括鼠标微缩线段的操作和对编辑区域触发事件的响应。鼠标停留会将此微缩线段高亮，鼠标离开则使之复原。鼠标点击会以另一种颜色高亮微缩线段，并且产生和滚动条一样的滚动页面效果，页面直接跳转到发生点击事件的微缩线段对应的代码行。编辑区域操作造成的页面位移、鼠标停留也会通过贝塞尔曲线反映到全景图中。

WPF绘图方式有利于处理鼠标事件。每条微缩线段不仅是被绘制到Canvas上，而是作为一个对象被添加到Canvas的子对象集合中。每个绘制出的微缩对象都有鼠标相关的事件，因此不需要根据鼠标的位置来人工判断当前触发事件的对象。

* + 1. 焦点区域（FocusArea）以及各组件的交互

全景图是鱼眼视图显示的中枢

1. 结论

1. [Thoughts on the Visual C++ Abstract Syntax Tree (AST)](http://blogs.msdn.com/b/vcblog/archive/2006/08/16/thoughts-on-the-visual-c-abstract-syntax-tree-ast.aspx) -- Visual C++ Team Blog [↑](#footnote-ref-1)
2. [Managed Extensibility Framework Overview](http://msdn.microsoft.com/en-us/library/dd460648.aspx) -- MSDN Library [↑](#footnote-ref-2)
3. <http://mef.codeplex.com/> -- MEF开源项目 [↑](#footnote-ref-3)