**Common Browser Runtime**

**什么是Common Browser Runtime(CBR)？**

**Common Browser Runtime**是面向Windows桌面应用的***互联网浏览服务运行时***，其目标是在桌面应用系统的进程内创建一个完整的、全功能的现代浏览器运行环境，使得桌面应用程序可以**充分调度本地资源的同时，具备互联网浏览器的全部功能**，确保底层基础上适应现代互联网环境的发展、变化。

从应用软件基本结构上看，每一款软件都依赖一种或者多种运行时环境，典型的运行时环境有.NET、Java、COM、VC Runtime、……，这些运行时的职责是给开发者提供必要的有足够健壮性保障的软件基础服务。一旦一类运行时得以建立，那么软件的生产力就会大幅度提升。做为一个新角色，Common Browser Runtime的使命不同于.NET、Java、COM等运行时，CBR专注于Web与桌面应用之间的衔接、互操作，赋予桌面软件Web特征。

**从Web Browser到Common Browser Runtime**

从历史上看，微软Visual Studio 2008发行的时候，是桌面软件的黄金阶段，那个时候，Google正在内部酝酿开发Chrome浏览器，2008-2018十年的时间，Web迎来了其书写历史的时刻，Google通过自己的浏览器，极大的影响着互联网世界。由于支撑互联网内容的Web页面是基于文本的，编程模式是基于脚本的，这些与依赖经典运行时的软件体系形成了鲜明的差异，基于页面、脚本的体系更容易生产，门槛相对底，适应软件需求的方式更加灵活、快速；而依赖于传统运行时的软件体系，需要经过编译、连接、二进制打包，然后才可能分发到具体用户，这种鲜明的对比，给生产力带来极大的反差。毋庸置疑，Web模式的软件与桌面模式的软件是两个极端，互联网更加青睐于Web页面为基础的应用模式，这种模式在柔性以及内容构造方面显示出巨大的优势。**软件领域只有有限的几款浏览器，却支撑起一个丰富多彩的页面世界**，形成鲜明对照的是，桌面软件种类繁多，每一款都有自己的领地，有自己独特的内容表现，被严重的碎片化了，而浏览器为基础的应用，表现出极大的通用性，以及良好的兼容性，**支撑互联网页面的基础相对有限，但是内容却近乎无限，相比之下桌面应用系统显得相形见绌，**其根本原因是因为桌面软件在运行时环节缺失了Web基因。

我们认为，浏览器框架属于基础性软件服务，如果将其限制于浏览器本身，是对软件生产力的一种禁锢，Web内容浏览能力不应是浏览器的私有特征，这意味着，软件基础运行时系列里面缺失的一环就是桌面软件应该具有浏览器基因，进而拥有自己的页面体系。

**Common Browser Runtime与Chromium开源项目**

CBR的核心部分建立在Chromium开源项目基础之上，与其他Chromium Based浏览器不同，我们将其编译为一组可以被其他应用动态加载的dll组件。我们对Chromium的Browser架构做了必要的调整，使其可以与常规桌面应用的消息机制、UI体系相互融合。对于Chromium的Renderer环节，我们侧重于V8以及DOM的扩展，使得每一个本地UI组件拥有一个渠道体现为Web页面的新元素，从这个角度看，我们给传统DOM提供了一个内容丰富的“超集”，使得用户可以用常规的页面方式构造具有桌面特征的新型页面。

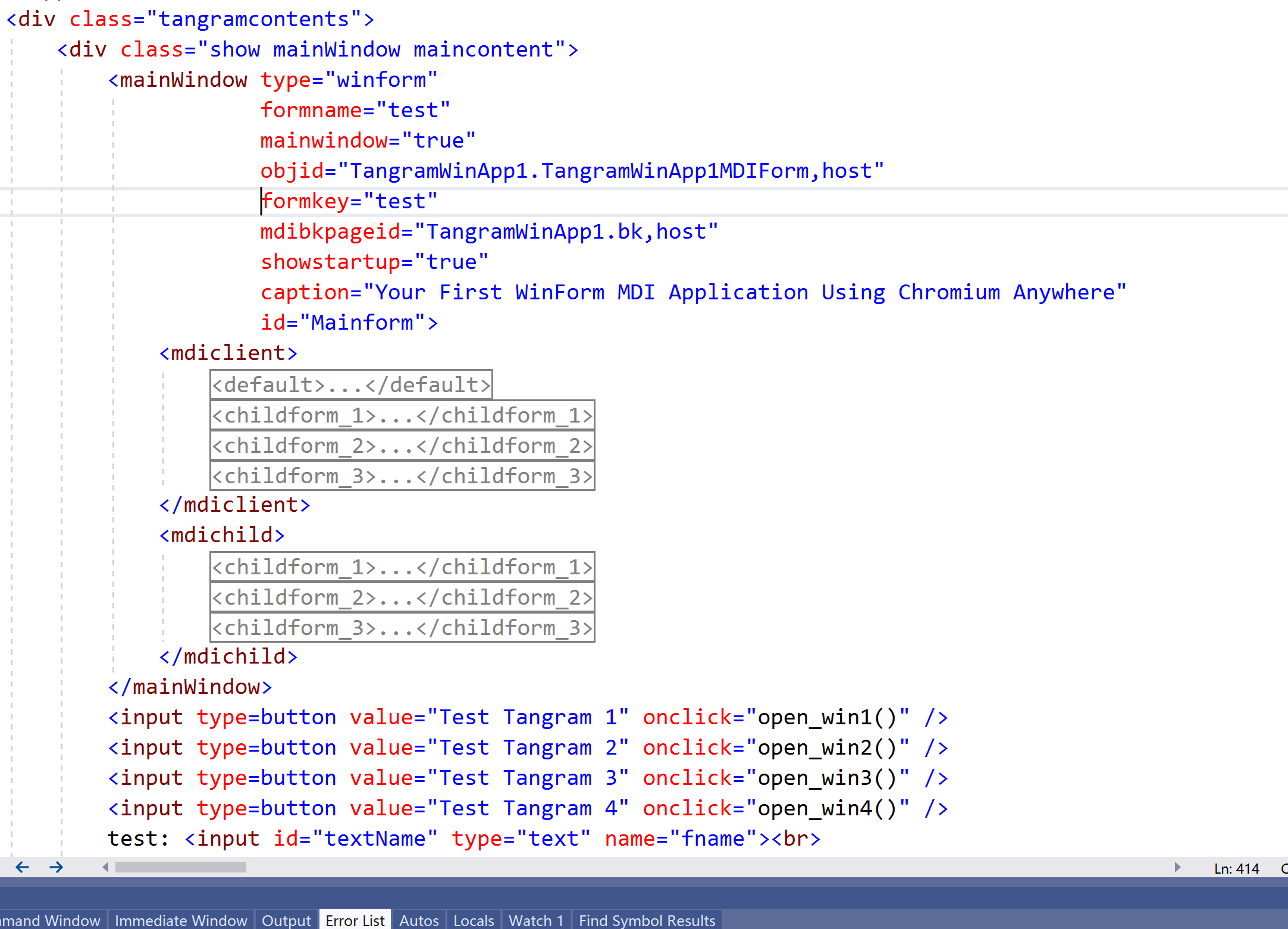
由于Chromium是开源架构，从CEF（Chromium Embedded Framework）开始，这个工程派生出一些面向应用的可编程框架，例如CEF Sharp等等。这些框架的目标是使得应用程序可以将Chrome浏览器窗口做为GUI界面的一部分。从开发角度看，为了在Web与应用之间形成良好的互操作，所有类CEF架构都屏蔽了Chromium的“SandBox”机制，这一点使得这类方案与Chromium的宗旨形成了偏差。这类方案保留了Chromium多进程架构、页面渲染，舍弃了Renderer进程的沙箱约束，从网络安全方面看，这类设计不适合提供通用浏览服务。我们认为不限于渲染页面，Chromium Project的方方面面都具有被应用软件充分利用的价值。CBR保留了Chromium开源项目全部功能，并为每一个应用程序在其进程空间内提供了一个浏览器运行态。为了充分利用浏览器资源，我们提供了一套面向开发者的BOM API，一种基于JavaScript的对象链接机制以及基于DOM的页面描述机制，以确保软件具备“页面化服务”的能力。就是说每一项具体的功能，可以体现为一种“页面”。那么，标准页面固有的屏障能否被突破？从浏览器多进程架构上看，出于安全性考虑，页面由Renderer进程渲染充分利用操作系统沙箱技术，这一点使得Java、COM、.NET等运行时与Renderer进程合理的隔离，同时这些Runtime相关的UI技术也就在Renderer环节被屏蔽了，不能成为页面的一部分。那么有没有办法合理的克服约束使得兼顾成熟组件技术的同时兼顾网络安全？CBR通过独特的技术途径，保留沙箱机制的同时避开了这些限制，其结果是形成一种新型Web页面。

**Common Browser Runtime与Web页面**

互联网时代 “浏览器机制”应该是提供内容服务的一种有效途径。应用系统的内置模型如果能够成为页面驱动的一部分，那么应用的模式就会变得“弹性十足”，这样就有可能将“应用软件”提升为“软件服务”，这一点正是互联网发展的一种趋势。当一个浏览器可以运行在应用程序的进程空间内的时候，页面会产生哪些变化？一个新的页面会是什么样的形态？会与传统的标准页面有什么样的差异？以.NET应用为例。众所周知，每个.NET应用都有自己的一组Form、UserControl，也可能包含一组WPF对象。通常情况下，这些组件仅能用于应用程序本身，开发者必须明确的写程序代码，才可能在具体的功能上体现出这些元素。如果应用程序支持Common Browser Runtime，那么如下页面：

|  |  |
| --- | --- |
| **图片包含 屏幕截图  描述已自动生成** | 图片包含 屏幕截图  描述已自动生成 |

就有可能是程序运行环节的一部分。如图，我们看到一个“熟悉”的全新的浏览界面。之所以熟悉，是因为这是一个标准的浏览器窗口，如果说这是一件全新的事情，是因为浏览器窗口呈现的Web页面是程序内部的.net组件与Web页面合成的“新页面”。现在，事情出现了新的变化，开发者可以如下发生写Web页面：



我们看到，这里描述了一个div结构，里面包含了一个“mainWindow”、“mdiclient”，还有其他元素……，这种页面化的结构，会如何展现给用户？实际上，上述页面会在实际运行的时候产生如下运行时画面：

图片包含 屏幕截图

描述已自动生成

**（CBT意义下的Web页面，不一定体现为一个浏览器窗口，页面可以是后台页面）**

从传统桌面端看，这是一个WinForm窗体，一般来说，基于.NET FrameWork代码形成的窗体，被编译到程序内部，才可能被用户激活；从CBR的视角看，WinForm等对象可以用Web页面重新规划，一个二进制的WinForm对象，可以看作是一个DOM结构，可以在不同的页面里面体现出完全不同的新的动态结构。而其中所包含的子控件，其行为可以用脚本重新定义，以适应不同的场景，就是说，通过CBR，一个本地的二进制UI对象，可以形成任意多个页面上的副本，具备很多种形态、属性，这些副本存储在服务端，通过URL获取，这种UI隔离，其优越性是传统基于本地的桌面模式完全无法比拟的。CBR创造了本地UI的服务端隔离机制，使得应用系统的机动性得以充分的挖掘。

**从CBR的角度看.NET Framework Class对象**

CBR需要在客户端技术与Web技术之间实现一种均衡，使得客户端的对象技术可以与Web技术形成最好的搭配。从技术角度看，类似.NET、Java、COM等成熟的Runtime已经为客户端技术奠定了非常成熟的基础，每一种运行时都会有自己的对象体系，CBR不是要淘汰这些，而是要充分、合理的运用。我们首先考虑.NET的类对象，其他运行时对象可以有相类似的途径。一般说，用户实现的一个.NET Class是一个Object对象，首先这类对象可以有具体的代码进行“实例化”，对.NET开发者而言，CBR给出一个具体的方法“BindObjToWebPage”，通过这个方法，可以将一个程序对象提升为一个Web页面对象，其他运行时有类似的途径。开发者可以直接在代码的合适环节将对象传递给这个方法。这个.NET静态方法形式如下：



如果你需要将一个对象映射到具体的页面上，你需要知道页面窗口的句柄，你自己的一个实例，以及需要一个该对象在页面上的名字。有这个方法，开发者就可以合理的选择对象做这样的对应关系，这里提供的途径是基于本地代码将具体对象映射到DOM环节，当然，可以在Web页面上通过JS对这类对象进行实例化。当这种映射完成的时候，具体互操作的条件就具备了。我们在Web页面上提供了基本的事件机制，用户可以在页面的如下环节做Web端的处理：

手机屏幕截图

描述已自动生成

实际的运行效果：

社交网站的手机截图

描述已自动生成

通过这样的途径，我们可以将几乎任何一个.NET对象绑定到具体的页面之上。原则上，开发者可以绑定一个具体的对象模型，设想一下，如果一组类对象，在客户端有一组实例、Web端也对应一组“对应物”，会是一种什么场景？这种“对偶”（发生在客户端主进程与Renderer进程），会营造软件的对偶原则，没有CBR的支持，软件可编程的另外一个环节要么不存在，要么相对薄弱，我们说，软件在Web端的可编程性应该一直是存在的，只是因为底层支撑环节的缺失，另外一部分一直没有被挖掘出来，一旦这一环节显示出来了，那么许多处理逻辑的原则就会不一样了，起码我们看到，我们可以在软件的另外一端用成熟的Web技术处理业务的可能性。

**从CBR的角度看.NET Framework UserControl对象**

从结构上看.NET、Java、COM等运行时，UI对象是这些成熟技术不可或缺的成分，在CBR的体系里面，这些充分依然是极为重要的。每一种运行时，都会有一种控件技术，在相关领域里面，这些应该是桌面编程的主题环节，我们依然用.NET Framework做为主干线索，以此说明CBR的策略，与COM、Java相关的主题不在本文里面赘述（可以借鉴）。控件对象，一直是GUI编程的中心话题之一，历史上有ActiveX控件、SWT、Java Widget、.NET UserControl等等。我们依然以.NET Framework为示范，其他领域可以平行的考虑。

毋庸置疑，.NET UserControl是一类内容极为丰富的对象，这类对象局部上有明显的UI结构，是桌面客户端非常有特色的一种成员。CBR首先要在html级别上充分考虑这类对象的价值。我们的目标是，用户用最常规的技术去构造UserControl，原则上不附加任何多余的规则，用户可以基于宿主Exe内部开发这类对象，也可以通过独立的组件库（dll）提供这类对象。对一个Web页面而言，由于沙箱隔离，页面与控件应该属于两个进程，这也是现代浏览器多进程架构的必然结构。

CBR意义下Web页面上的.NET UserControl

社交网络的手机截图

描述已自动生成

这里看到的是包含一个UserControl的Web页面。我们需要一个简单的书写规则，让用户方便的用页面技术将Web与.NET GUI直观的结合在一起。首先需要一种页面布局方案，例如，用户可以轻松的实现如下布局：

**从CBR的角度看.NET Framework WinForm对象**

Form、Window这类对象，当然是.NET、Java、COM等运行时支持的关键GUI成分，这些在CBR里面同样不会忽视，做为一个重要的环节，我们将Form、Window与浏览器窗口归结为一类带有独立“窗口行为”的对象，与控件技术不同，这些对象可以独立存在，而控件必须依赖宿主窗口。

**CBR与JavaScript**

从开发者的方面考虑，Web环节的JS，应该有一个合理的方式满足编程环节的需求

**CBR与其他编程语言的关系**

从桌面端看，桌面端的开发语言应该与页面端的基本语言存在一规范的互操作通道，我们希望这个环节能够体现出一种非常自然的关联，桌面端的语言环节可以有消息通道到达页面环节，这个通道可以承载合适的数据结构，能够合理的体现桌面端的请求，而页面端可以有类似的消息通道到达桌面端，，从编程环节看，基于消息的互操作机制，可以有效的将脚本语言与编程语言之间建立协作纽带。从JS方面看，可以建立一些回调策略，用于页面逆向操作桌面端。

**CBR是桌面对象技术、GUI技术与Web技术的一个大融合**

从某种意义上看，一个基于页面的描述机制，使得软件技术实现了从“端”到“云”的全方位的融合，这种融合体现为成熟客户端技术与成熟Web技术的和谐的互操作，由于跨越从端到云两个环节，所以“均衡”是非常重要的。如果客户端是整体，那么开发者面对的是单台机器这个作用域，如果将视野拓展到互联网，那么“端”仅仅是大环境的一个原子，所以全局的角度看，我们不难看出CBR勾划出的“幅度”。

**每个桌面应用拥有一个自己的Web后端**

从更广泛的方面看，做为一个工业标准，.NET Framework提供了丰富的程序元素，这些元素创造了丰富多彩的.Net软件世界，然而其局限是缺乏html元素具有的可描述性，CBR消除这个限制，我们知道，互联网浏览器软件首先内部编译好一组二进制组件（CSS、DIV、li、……），然后制定页面书写规范形成页面，CBR给出了类似的原则，在CBR看来，页面里的DIV仅仅是一个C++实现的二进制对象，只不过在页面环节体现为一个<div>标签，类似的，WinForm等.NET对象应该在网页内被等价的实现为类似<div>的标签，通过这个途径，标准页面的DOM模型被放大了，一个全新的页面世界出现了。CBR构建了一个远远大于W3C标准的页面世界，如果不能很好的处理JS引擎，那么这些新页面就会变得极其乏味。从浏览器内部结构看，每一类标准的dom元素的事件都是被预先规划好的，所以页面脚本引擎可以很好的处理标准DOM元素的事件，这一点是Web处理交互行为的基础。.NET带来的挑战是，由于每个.Net对象都可能有自己的事件集合，新的页面结构JS能否处理.NET对象的事件？如果不能很好的处理这个问题，桌面软件系统的柔性，就无法淋漓尽致的展现。

CBR提供了JS处理.NET对象事件的通用规则，这个规则还考虑到现代JS程序员的编码习惯，我们允许在Renderer进程可以监听.NET对象的事件，提供了JS、.NET对象互操作的基本规范，页面开发者可以以如下的方式处理他们关注的.NET对象事件：

图片包含 屏幕截图

描述已自动生成

在CBR看来，一个.NET Framework对象，在Web页面上可以存在一个“对偶对象”与之匹配，来自.NET Framework的对象与CBR提供的页面对象之间的“彼此纠缠、互操作”，使得桌面开发环节增加了对应的“Web后端开发环节”，软件的刚性体现在客户端编程；软件的柔性表现在“Web后端的开发”，这样的搭配，提供了互联网时代软件开发的新模式。Web端的开发有利于软件逻辑的中心化控制，客户端开发可以充分发挥二进制组件的客户端体验，二者完美的结合，才能给出完美的应用系统。

**互联网浏览行为是一种共性，应该是桌面客户端的公共属性**

CBR的出发点是：一旦桌面软件支持CBR，那么浏览器的概念就会被泛化，桌面软件与Web软件的边界就会逐步消失。事实上，每个支持CBR的应用程序就是一个完整版本的Chromium浏览器，与Chromium不同的是，除标准浏览器窗口之外，CBR应用还支持其他类型的窗口，特别，允许第一个应用窗口是非浏览器窗口。CBR将浏览器的功能看做是默认功能，从这个意义上看，浏览器的概念被淡化了。当更多的应用系统直接具备网络浏览功能的时候，特殊的浏览器会逐渐消失。

**CBR = Chromium + CLR , CBR APP = CBR + Built-in App Model**

由于CBR是面向桌面应用的，所以其内置的浏览器模型必须与标准浏览器模型存在本质的差异，做为差异的一部分，我们将CLR做为CBR的一个组成部分，由此决定了CBR内部的JS引擎可以合理的与CLR互操作，为此我们在CBR内部实现了以下关键特色：

1. JS可以处理CLR对象的事件、委托；
2. JS可以做为.NET对象方法、属性的自然延申；
3. JS对象与.NET对象可以基于消息机制互操作；
4. JS引擎可以创建.NET UI元素，例如WinForm、UserControl以及WPF，并与之互操作，例如监听这些UI组件的各种事件。

从应用软件运行时的角度看， CBR就是“Chromium 开源框架”与.NET Framework合并的结果，从软件结构上看，CBR应用就是“Chromium + CLR + Built-in App Model”。在CBR的角度看，JS + “.NET语言”，给开发者在编码环节带来足够的弹性，例如，开发者可以合理的在“脚本代码”与“二进制代码”之间取得平衡，让二者各司其职，在软件的桌面开发环节，可以充分利用CLR的优势，而软件的“网络、Web环节”可以充分的利用“html+js”丰富的技术资源，这样的构思给软件开发带来极大的灵活性，更加容易适应互联网时代的各种需求。

**Software as a Platform**

CBR做为一个应用运行时，最直接的效应是每个桌面应用可以拥有自己的Web后端，这一点为传统意义下的桌面应用软件提供了面向互联网的延申。CBR试图从页面化的角度，给应用程序提供一个全新的组织对象的能力，一般说来，无论你规划得多么充分，应用系统的模型总是不完备的，平台化意味着外界的干预，从数据结构、对象模型到运行机制都如此。应用系统的主干对象，是系统的核心，其行为如果完全依赖于二进制代码，那么平台化就会受到非常大的制约，我们认为，一个可充分定制化的后端，是软件平台化最基础的保障，软件架构者可以将明确的代码，用二进制的方式实现，那些不明确的环节，可以通过事件、委托、消息机制迁移到Web环节，例如，可以在一个对象的方法环节触发合适的消息，进而委托到Web端，这样软件适配具体的应用场景，就会因Web中心服务而形成最合理的匹配，这一点几乎是Web应用的专利。CBT的建立有可能导致应用软件平台化是一种常态，软件平台化的最关键因素是：

1. 软件的关键对象可以依据具体情况个性化；
2. 具备产生新业务应用能力；
3. 创建新对象的机制；
4. 合理的内容定制渠道。

CBR可以依托桌面软件技术，构造一个更加丰富的Web世界。如果每个桌面应用合理的包容一个“内置浏览器”，那么，应用程序本身就会支撑一个Web页面形成的“内容生态”，每一个页面都可以看成应用系统的一个功能。从软件结构上看，CBR是给桌面软件提供了一种“内容、功能”的补充机制，客观上，这一举措改变了软件的根本结构，具体表现在以下两个方面：

1. 语言机制上，JS成为软件内置的“编程引擎”，这一点类似于Office里面的VBA；
2. UI上，CBR给应用系统提供了任意多个“应用页面”以及无限个数的WinForm窗体，可以与VBAForm有异曲同工的效应。

一旦一个应用系统内置了“编程引擎”以及可再生的UI机制，那么这个应用就是一个开发平台。从更加一般化的意义上看，不是每个应用系统都适合平台化，然而，只要其对象模型合适，同时希望演化成一种服务，那么就具备建立自己应用生态的客观基础。CBR的出现，使得这种可行性具体化了。应用程序拥有自己的应用生态，意味着程序结构必须突破本地的局限。我们认为，建立生态的基本条件应该是：

1、合理的内部模型；

2、丰富的外部元素、素材支持；

3、合适的内容生成机制；

页面开发与那些基于常规运行时（.NET、Java等等）做开发是有相当大区别的，对今天的互联网而言，已经为基于页面的运营方式提供了大量的基础设施，因此，应用程序的页面化是利用现代互联网最合适的模式。每一款具体应用，都会经历从小到大、从内容单调、匮乏到内容丰满、多样化的阶段，CBR试图为应用系统在这些方面提供长远的保障，这一点是将其视为底层运行时的一个依据。从技术上看，“页面结构+.NET Framework”，已经给软件团队提供足够的内容制作空间，CBR所提供的不局限于此，还允许软件构造者提供更多的DOM元素，我们提供扩展规范，允许开发者提供.NET之外的扩展DOM技术，这类技术可以延续到Java甚至其他语言机制。我们认为，营造一个有拓展能力的DOM机制，是充分发挥WebPage作用的关键所在。

客观上CBR为桌面应用提供了基于文本结构组织应用程序对象、应用程序内容的基础，具体体现在：

1. 应用对象行为可以延续到Web页面；
2. 新的业务应用可以基于“脚本+扩展对象”用现代的Web技术实现；
3. 通过与成熟的类似.NET这样的运行时协作，给应用软件提供对象、内容级别的保障；
4. 文本化页面制作确保应用软件有自己的内容生成机制；

所有这些方面，都是传统桌面软件不具备的特征，也是“互联网浏览服务”的精髓。