**Common Browser Runtime**

Common Browser Runtime是面向Windows桌面应用的浏览器运行时，其目标是在桌面应用系统的进程内创造一个完整的、全功能的现代浏览器运行环境，不同于主流web浏览器，这个运行时包容标准的浏览器功能，同时支持一个更大的页面DOM元素集合以及一个执行力更强的JavaScript引擎。

Common Browser Runtime的意义

仔细揣摩“应用程序”这个概念，应用程序除了包含开发者直接编写的代码之外，其实还包括一个运行基础，在.NET应用开发领域，这个运行基础通常称为.NET Framework。通常意义下的应用开发其实是工作在运行基础之上，通过编写高水平的代码来创造应用的价值。但应用程序却受到了本身运行基础的限制，例如NET应用会受制于NET Framework的限制，而Web应用会受制于浏览器技术的限制。我们认为下一代应用开发应当在这个层面拥有更大的自由度，开发者应当有机会选择使用更多的运行基础，为此我们将焦点转移到了浏览器之上。传统意义上的浏览器是面向终端用户的独立应用程序，但实际上随着这些年的技术发展，浏览器已经成为了一个包容海量Web技术资源的运行环境。而我们只是在此基础上稍微向前多迈了一步，真的将浏览器转换成了一个普通开发者可以直接利用的运行基础。

通过添加一个浏览器运行基础到一般应用程序中，开发者不再被迫使用桌面技术来实现他们的所有任务，而是能够有选择的利用浏览器技术来解决他们的问题。

服务交互方式的进化

从客户端角度来看，Web应用采用了页面的方式来与服务进行交互，这种交互方式非常的灵活和高效。而传统的桌面应用开发仍然采用非常底层的HTTP请求的方式来与服务进行沟通。而通过为桌面应用程序提供一个浏览器运行环境，也将这种基于页面的服务交互方式带入了桌面应用之中，这是一种质的飞跃。

**Common Browser Runtime与Common Language Runtime**

Microsoft 的CLR是一个历史悠久的应用软件支撑体系，做为现代软件工业的一个重要组成部分，以其为基础的应用软件不计其数。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，我们考虑将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。

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

做为浏览器运行时，CBR必须依托一款现代浏览器架构，通过慎重考虑，我们最终定位于Chromium开源框架，与所有的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库。我们对Chromium的Browser架构做了必要的修改，使其可以与常规桌面应用的UI体系相互融合，具体表现在：

1. 在应用系统的进程内，Browser可以体现为独立的浏览器窗口；
2. 可以做为其他窗口的子窗口

对于Chromium的Renderer环节，我们侧重于V8以及DOM的扩展，进而，每一个CLR的UI组件可以体现为html上的新型DOM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”，使得用户可以以常规的页面方式形成具有应用特征的新型页面

**Common Browser Runtime = Chromium + CLR + “Built-in Application Model”**

从软件结构上看，CBR就是“Chromium + CLR + Application Object 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。

CBR的宗旨是依托桌面软件技术，构造一个更加丰富的Web世界。事实上，现代浏览器也是一种桌面软件，只不过其内部集成了标准的DOM与标准的JS引擎。通过观摩一些典型的现代桌面应用，我们发现，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“内容生态”，每一个页面都可以看成应用系统的一个功能延申。互联网时代强调软件服务，而“浏览器机制”，恰好是提供内容服务的一种保障。形式的集成浏览器，是一种“堆砌”，意义不大。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。CBR的目标之一就是让桌面软件具备“页面化服务”的能力，就是说，软件系统的每一个具体的功能，或许就是一种“页面表现”，那么，标准页面固有的屏障能否被突破？这一点是CBR面对的一个挑战，从现代浏览器的多进程架构上看，出于系统安全性的考虑，页面是在另外一个进程渲染的，传统的桌面组件成为页面的一部分，是一种操作系统级别的技术限制，因为，操作系统的“沙箱”机制使得渲染进程不可能支持桌面组件，那么有没有办法克服这种操作系统级别的约束？CBR通过独特的技术途径，保留沙箱机制的同时克服了这些限制，其结果是一种新型的Web页面应运而生。

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，然而，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

由于CBR的存在，每个桌面组件除本身的固有Nativate属性外，额外派生了其“Web属性”，这种对偶匹配，是传统意义下的软件开发机制无法具备的。从本质上看，每个对象都有其web对应物，这种“对偶效应”，使得传统桌面软件在web环节产生了一个对偶系统，这个对偶系统相当于桌面系统的“Web镜像”，CBR使得这种对应成为可能。

**CBR与软件的“柔性”**

绝大多数桌面软件是基于编程语言实现的，基本都有编译、连接最后形成二进制包的过程，与之对照的是，web系统大体是“页面”+“脚本”的模式，这种方式的特点是基于文本以及脚本的，有“编排”的特征，大多数过程原则上不用编译，随着修改随之生效，过程充满着“柔性”，与此相比，桌面端基本是“刚性”的。

从软件的功能体系看，柔性易于控制、调整，所以会部署于服务端，与本地合理的“隔离”，这样非常适合管理、控制，这一点恰恰是桌面端的短板，受制于编程语言的限制，桌面端的软件系统似乎在结构上缺失了一些东西，因此缺乏必要的“柔性”，从发展的角度看，“刚柔并济”才是和谐之道，CBR致力于为软件柔性提供支撑，只要你的软件需要这种“柔性”，CBR就是为你准备的。

大多数被称之为引擎的系统，基本都有其Form系统，VBA就是一个典型的案例，其特点是，每个Office文档，都可以在文档级别拥有自己的一组Form。而常规的应用系统，一旦编译结束，动态的Form就是一个奢侈品，几乎是不可能的，因为一个二进制的东西，不知道如何处理动态，也无法处理其动态结构，由于缺乏文档的支持，动态form的结构不知道存在哪里。CBR支持的页面，可以看做是应用系统的一种内置文档，因此，动态存储成为可能，这个时候，服务端似乎是应用系统的一个存储系统、仓库，提供了无限的可能。