**Native Client**

本文描述了Native Client（NaCl）的组成部分及它们的作用，并对NaCl涉及的其它知识点做了必要阐述。

Native Client是一种为在浏览器内安全高效地运行已编译的C和C++代码的沙盒，独立于用户的操作系统。Portable Native Client是对这项技术的扩展，它与架构无关，开发者能使用ahead-of-time(AOT)翻译技术，使得他们的代码只需编译一次，就能在任何架构的任何网站上使用。Native Client在没有牺牲web的安全性和可移植性的情况下，给现代web浏览器带来了本地代码才有的性能和底层控制力。

一个NaCl应用由许多可信和不可信NaCl模块组成，每个模块都在一个进程中单独运行。使用Native Client Software Development Kit( SDK)能创建使用NaCl并在跨平台的Chrome里运行的web应用。一个Native Client web应用由JavaScript、HTML、CSS和一个SDK所支持的语言编写的NaCl模块组成。

一个Native Client Web 应用的结构：

HTML和CSS: HTML文件通过embed标签告诉浏览器上哪儿找manifest（nmf 文件）。

Manifest：它标明要加载的模块和具体的选项。

Pexe(portable NaCl file)：这是个编译好的Native Client模块。它采用Pepper API。

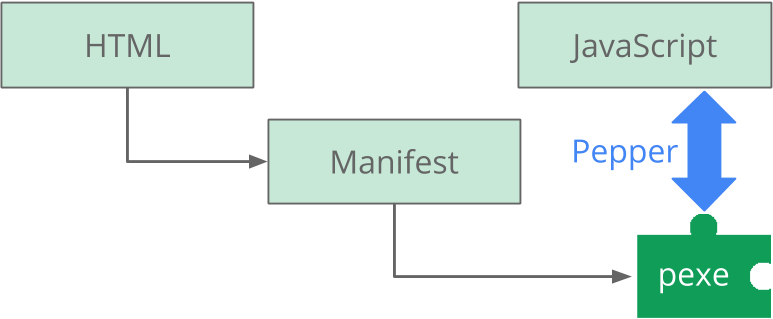


图1 Web应用的结构

关于[nmf](https://developer.chrome.com/native-client/reference/nacl-manifest-format)：每个Native Client应用都有一个JSON格式的NaCl Manifest File(nmf)。nmf告诉浏览器从哪里下载和加载你的Native Client应用的文件和库。这个文件也能包含设置选项。nmf中包含多个域，其中program域是必须有的。program域具体指明了在Native Client运行时环境中将要被加载进去的主程序。对于一个Portable Native Client应用来说，这就是一个已静态地连接好的二进制代码pexe文件的URL；对于与架构相关的Native Client 应用来说，这里是URL的一个列表，每个URL对应一个支持的架构。对于动态连接的可执行模块来说，program是用来加载动态可执行模块和它的动态库的动态加载器。nmf文件中的files域指明了Native Client应用所用到的文件资源的一个目录。Portable Native Client应用不需要这个域（取而代之的是采用PPAPI URL 加载器接口加载资源）。files域对于动态连接的可执行模块来说很重要，这种模块必须在PPAPI被初始化之前加载文件。files目录应当包含主要的动态程序和它的动态库。动态程序依赖并在程序开始运行时链接进的动态链接库必须被列在files目录里。

从高层次的抽象来讲，Native Client由工具链和运行时组件组成。工具链是一系列的开发工具（包括编译器、连接器等），它们负责将C/C++代码转换成Portable Native Client模块 或者Native Client模块。NaCl SDK提供两种工具链：

1.基于LLVM的工具链，它产生出一个单一的可移植的模块(pexe)。在运行时里，AOT翻译器（内置在浏览器中）将pexe翻译成与架构相关的本地代码。PNacl翻译器的任务是运行pexe模块。本质上讲，这个翻译器将一个pexe编译成一个nexe，并且在Native Client沙盒里执行这个nexe。翻译器采用了智能缓存技术以避免客户端浏览器之前已经编译过pexe的情况下重编译pexe。

2.基于GCC的工具链：它产生多类依赖于架构的模块(nexe)，这些nexe会被打包进应用。Native Client也支持nexe模块直接地在浏览器里执行 。在运行时，浏览器根据客户端机器的架构决定加载哪个nexe。运行时组件被嵌入在浏览器或者其它的允许Native Client模块安全高效地运行的宿主平台。各个组件间如何交互如图所示。

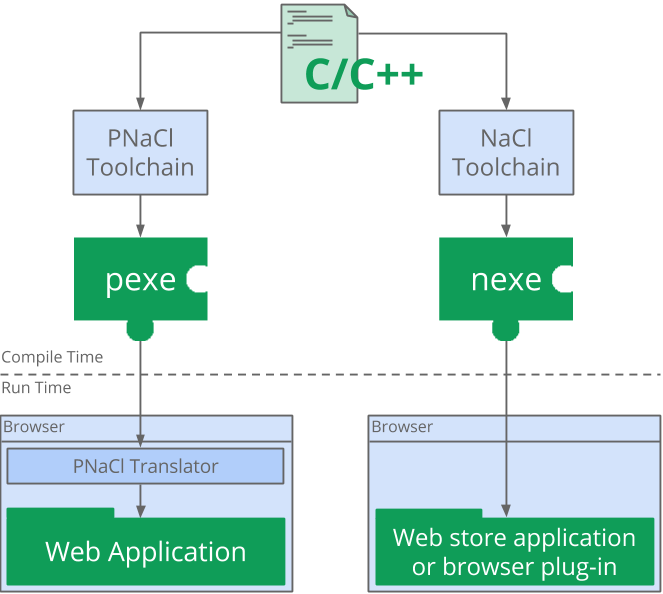


图2 NaCl工具链及它们的输出

 Native Client基于x86进程内“内层沙盒”构建而成。而为了进一步防御攻击，还构建了一个“外层沙盒”用于进程边界的隔离。内层沙盒使用代码静态分析技术在不可信的x86代码中检查安全隐患。以前这样的分析受到了很多技术的挑战，比如说可自我修改的代码，重叠指令。为了解决这样的问题，在Native Client中为代码制定了一系列限制和结构规则，从而确保本地代码模块能可靠地进行分解，然后通过代码验证器运用机器码验证器 (Code Validator)来确保可执行文件只包含合法指令集中的指令，即NaCl通过机器码验证器能保证只有安全的机器码才能在系统执行。由于x86-32/x86-64是[复杂指令集](http://zh.wikipedia.org/wiki/%E8%A4%87%E9%9B%9C%E6%8C%87%E4%BB%A4%E9%9B%86" \o "複雜指令集)，[指令](http://zh.wikipedia.org/wiki/%E6%8C%87%E4%BB%A4)长度不一，在[控制流程](http://zh.wikipedia.org/wiki/%E6%8E%A7%E5%88%B6%E6%B5%81%E7%A8%8B)中可能隐藏非安全机器码，从而使验证十分耗时。NaCl运用固定长度的16或32[位](http://zh.wikipedia.org/wiki/%E5%AD%97%E8%8A%82)的指令束(Instruction Bundle)，使机器码验证器设计简单(只是约600行[C语言](http://zh.wikipedia.org/wiki/C%E8%AA%9E%E8%A8%80" \o "C語言))和高效率； 而运用指令束只会失去5%的执行效率。

与语言无关的沙盒不是基于硬件保护域的，而是基于Software Fault Isolation(SFI)。SFI提供高度保险的安全保障并且很大程度上独立于操作系统和其它的系统级细节。SFI通过结合静态分析和软件防护实现高度保险的安全保障：运行时的安全性检查或沙盒化操作采用短的内联指令序列完成。SFI允许静态地验证的机器码的创建。这种代码通过静态分析能被独立地验证，进而只允许被限制的执行。SFI对直接和间接的控制转移进行限制且必须阻止控制流绕过内联的软件保护。

LLVM技术的要点在于能将编译型语言转化为一个统一的中间语言，NaCl通过对这个中间语言的执行，即可达成任何汇编型语言的运行。为支持SFI，对LLVM汇编程序加入了以下指令：bundle-aglin-mode <num>、bundle-lock <option>和bundle-unlock。其中num指定了指令束的大小是2的num次幂字节。被包含在bundle-lock <option>和bundle-unlock之间的指令不能跨越束边界。每个NaCl软件保护都和一个紧邻的潜在不安全的指令合并为一个伪指令，使得每个保护只限制单个指令。控制流指令操作的对象必须为有效的代码，而且所有的计算的控制转移必须在一个指令束的开始处。间接跳转只能通过与目标地址对齐的伪指令实现，NOP填充用以保证CALL指令只在指令束的结尾的地方出现，这样返回地址一直都是对齐的。一个由5条以32字节对准的NaCl指令束组成的代码区如图所示（灰色部分为NOP填充，并有两条伪指令：写寄存器R和调用寄存器R）：

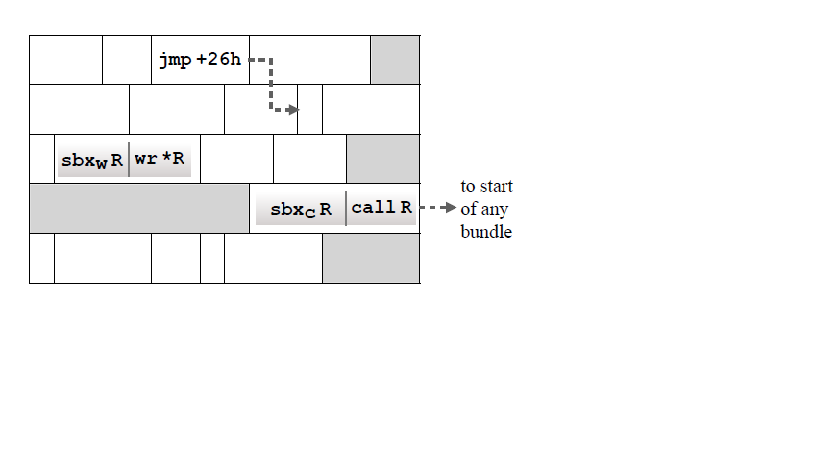


图3 抽象的NaCl机器码，32字节NaCl指令束对齐

内层沙盒的设计仅限于机器码明显的控制流（通常以调用和跳转形式出现）。其他类型的控制流，如异常，则由服务运行时管理。内层沙盒定义了一系列用于可靠分解的规则，一个用于监视这些规则的编译工具链，以及一个确保代码复合规则的静态代码分析器。这样的设计使得可信代码段（Trusted Code Base）因其中不包含汇编工具而体积很小，同时验证器也能在允许通过测试和审查的时候做到体积也足够小。

内层沙盒用于在一个本地进程中创建一个安全的子域。在这个子域里我们可以将一个可信的服务运行时 (Service Runtime)子系统（后面详细说明）和不可信模块放置在同一个进程中。通过Trampoline调用门和Springboard返回门实现可信代码和不可信代码之间的控制转移。Native Client 将本地程序加载到SelLdr 进程之中，所有SelLdr 进程的地址空间即表示了整个运行时的内存分布。在SelLdr 的地址空间之中，低地址最开始64K 为Native Client 自身的服务运行时（前4K 用于识别NULL 指针，剩余60k 用于Trampoline 调用和SpringBoard 返回门），用来实现Native Client双层沙箱中本地程序的系统调用。

NaCl沙盒保证代码只能通过安全的被列在白名单中的API来访问系统资源，并且，代码在其限制内进行操作， 而不会试图 干扰别的运行在浏览器内外的其他代码。NaCl验证器在代码运行前静态地分析它们。NaCl模块始终在一个权限受限制的进程内执行。这个进程同外面进程的唯一交互是通过定义好的浏览器接口完成的。

为扩展沙盒，在沙盒中迁入了两个JIT-compiled语言系统：Mono CLR运行时和V8JavaScript引擎。迁移它们的工作主要是改动它们使得它们能输出可验证安全的代码，并在代码有改动时保持这种安全性。扩展的沙盒化允许 引进的语言运行时和它们的本地库的安全执行，即使在它们为了效率而使用高级的代码修改技术时。NaCl沙盒采用SFI限制哪些指令可以执行、以什么顺序执行以及限制指令对内存地址的使用。

可执行内存地址只能通过可信的接口进行修改，而不能够直接通过NaCl代码修改。Just-In-Time（JIT）编译程序，在一个临时的数据存储器缓存区里生成机器码，它作为NaCl不可信代码运行。为了安装这些生成的代码，JIT调用nacl\_dyncode-create将控制转移到NaCl可信的运行时。运行时验证代码并将之安装在可执行内存里。安装后，JIT可能想修改代码，如更新存储在直接指针里的指针地址，nacl\_dyncode\_modify(可信的)接口支持这种修改。当某个时候这些代码不再需要了，它可以通过nacl\_dyncode\_delete删除以重用存储空间。

Native Client为进程间通信提供了一个可靠的数据报抽象机制——IMC服务或直接称为IMC。IMC允许可信和不可信模块之间越过进程边界发送/接收包含无类型字节数组的数据包，还可以包含用于文件共享、共享内存对象及通信通道的“NaCl资源描述符”。IMC同时也是两个更高级的抽象机制——SRPC和NPAPI的基础。第一种SRPC，它用于定义和使用子过程来跨模块边界访问，包括从浏览器中的JavaScript调用NaCl代码。第二种NPAPI，它提供了与浏览器状态交互的接口，包括打开URL和对DOM的访问。该接口遵循已有的关于内容安全方面的规则。这两种机制的任一机制都可以用来与浏览器进行交互，包含页面内容修改、处理鼠标和键盘事件和获取其他站点内容，因为这些都可以通过JavaScript访问。

   服务运行时用以模拟系统调用，为NaCl模块之间，以及和浏览器之间的交互提供容器。它一般会提供应用程序编程环境相适应的一系列系统服务。为了支持malloc/free接口或其他内存分配抽象机制，它提供了sbrk()和mmap()系统调用。它提供了POSIX线程接口的一个子集，用于线程的创建和销毁、条件变量、互斥量、信号量及线程本地存储。它还提供了常用的POSIX文件I/O接口，用于通信通道以及基于web的只读内容的操作。为了防止恶意的网络访问，connect()和accept()这样的系统调用不可使用。

服务运行时是一个本地可执行代码，它被一个NPAPI插件调用，这个插件也用来支持服务运行时和浏览器之间的交互。它实现了dynamic enforcement机制，这个机制维护了内层沙盒的完整性，并提供了一个资源抽象，这个资源抽象用来把NaCl应用程序从主机资源和操作系统接口中分离出来。它包含了可信的代码和数据，但与包含的NaCl模块共享同一个进程，且后者可以通过一个受控的接口去访问服务运行时。服务运行时通过x86的内存分段和分页机制相结合的方式来阻止不可信代码的非法内存访问行为。

关于插件：

Chrome 共有两种插件类型In-Process 和Out-Process，Native Client 的运行时就是建立于in-Process 插件之上的。Out-Process 插件即通用架构的插件，它有自己单独的进程，其并不在Chrome的沙箱之中， 可以不受任何限制的访问本地资源（直接进行系统调用）。

Out-Process 插件通过主流的NPAPI 接口标准与浏览器进行通信。由于其能够直接访问本地资源，Out-Process 插件可以到达本地应用的运行效率，也容易引起安全问题。相对于Out-Process 插件，In-Process 插件更安全。它被放到浏览器渲染引擎的进程之中，也就位于沙箱之中。通过Chrome 自定义的PPAPI 接口标准与浏览器进行通信，而不能直接进行本地系统调用。Chromium的安全架构将浏览器分成了两个保护域：浏览器内核和渲染引擎。在沙盒中的渲染引擎负责进行复杂的、易错的任务，如解析HTML和执行JavaScript。这样就保护了用户的文件系统将渲染引擎当作一个黑盒减少了浏览器内核安全监视的复杂性。

通过隔离web程序和支持它们的浏览器组件的多个实例，一个多进程架构能有效地解决当今浏览器的可靠性问题。

Pepper plug-in API :

缩写为PPAPI,是web浏览器插件的跨平台C/C++ API.PPAPI允许一个C/C++模块与主机浏览器通信并以一个安全的可移植的方式访问系统级功能。Native Client模块的一个安全限制就是不能使用OS级的功能调用，PPAPI为那些模块提供了类似的API。

PPAPI是为了简化out-of-process插件执行的是实现而设计的扩展，进一步讲，它的目标是为使插件完全的跨平台提供一个框架。NaCl提供了一个跨平台的类似POSIX的环境。Naclports是样例程序和库补丁的仓库。[可用的NaCl ports列表](https://code.google.com/p/naclports/wiki/PortList)由generate\_port\_list.py脚本自动生成。

Native Client插件及其他各组件间运行及通信情况如图所示。

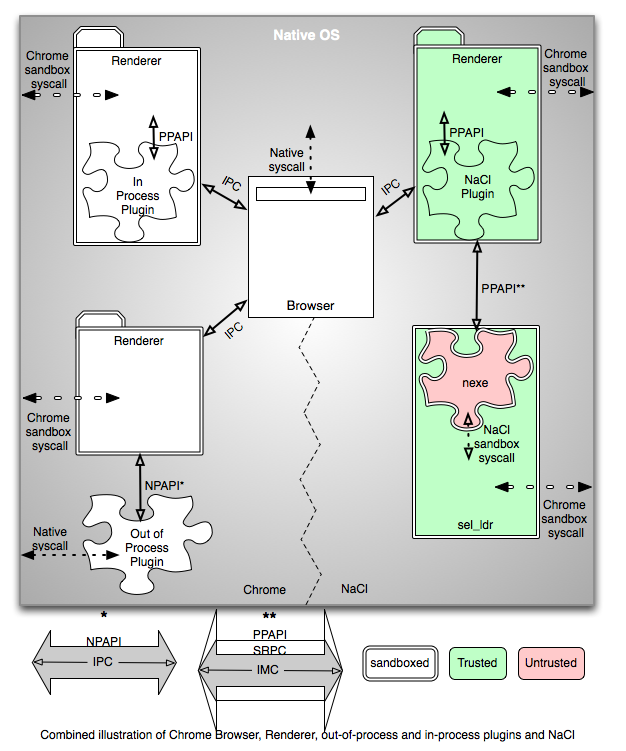


图4 Native Client插件及其他各部分的运行及通信情况总览

Native Client相关论文：

1. Native Client：A Sandbox for Portable,Untrusted x86 Native Code.
2. Adapting Software Fault Isolation to Contemporary CPU Architectures.
3. The Security Architecture of the Chromium Browser.
4. Isolating Web Programs in Modern Browser Architectures
5. PNaCl: Portable Native Client Executables.
6. Language-Independent Sandboxing of Just-In-Time Compilation and Self-Modifying Code.