XAMARIN 诞生记

Xamarin 吉祥物是一只憨态可掬的猴子。它背后的艰辛和汗水，你能否体会到呢?

2016年年初，微软公司收购了之前几年一直紧密合作的 Xamarin 公司，这家以小猴子为吉祥物的公司也进入了更多开发者的视野。那么，这个古怪的公司名字又是哪里来的？为什么它会选择猴子来做吉祥物呢?

Mono 项目的启动

时间倒回到2001年7月23日，O'Reilly 出版集团主办的第三届 Open Source Convention 大会在美国加州美丽的海滨城市 San Diego 开幕。作为开源社区年度盛会，媒体和业界精英齐聚。由于微软公司副总裁 Craig Mundie 的出席，现场最热的主题无疑是微软刚刚开始倡导的 Shared Source 运动和已经如火如荼之势的 Open Source 开源

运动正面交锋。所以当某个个子不高戴着眼镜的年轻人走上会场的讲台，开始介绍他领导下的一个新的开源项目时，不会有多少与会者意识到自己正在见证一段传奇的开启。这个其貌不扬的年轻人，便是 Miguel de Icaza。

Miguel de Icaza 知名开源社区领袖，领导了 GNOME 项目和 Mono 项目。Ximian 和 Xamarin 两家公司的联合创始人。现任职于微软公司。

Miguel 于1972年出生在墨西哥首都墨西哥城一个高级知识分子家庭，父母都是科学家。由于未知的原因，他没有完成本科学业就跑去工作，从二十岁起就投身到开源软件运动中。有着过人天赋和热情的他，先后参与 Midnight Commander、Wine等热点项目，逐渐成为开源社区的活跃分子。

有趣的是1997年夏天，他竟然跑去面试微软公司的职位，想加入 Internet Explorer Unix 项目。不知道是面试过程不够顺利，还是他没有学士学位对于申请工作签证的负面影响，他最终没有加入微软。关于这段面试经历，Miguel 也很长时间没有公开提及，直到被好哥们 Nat Friedman 在2009年公布出来。

失之东隅，收之桑榆，实际上就在他面试失意的这个夏天，另一件事情使他一跃成长为开源社区的大明星。

“.NET is a good platform to innovate from”

MIGUEL DE ICAZA

作为九十年代最重要的开源操作系统，Linux 在服务器场景中非常成功，但是它一直希望通过提供可以媲美 Windows 操作系统的桌面环境来扩大影响。当时最为流行和成熟的桌面系统自然非 KDE 莫属，可是它底层依赖 Qt 这个不开源的图形库，这使得很多开源运动的领袖意识到必须抛开 Qt 另起炉灶来设计新的桌面系统。可以想见这不是一个小工程，需要强有力的领导团队。

这个新的桌面系统，就是日后大家熟知的 GNOME 项目，技术方面的领导责任落到了年轻的 Miguel 和 Federico Mena 肩上。幕后支持开发的 GNOME 基金会主席一职，则由同样年轻的 Nat Friedman 担任。为了更好统筹 GNOME 的开发力量，1999年 Miguel 还和 Nat 联合创建了 Helix Code 公司（后来改名为 Ximian）。

两三年的工作下来，到2000年 GNOME 项目已经走上正轨，可以按部就班地发展。这使得 Miguel 有了时间来思考下面自己做点什么好玩的东西，也是这个阶段他接触到了微软公司刚刚公布的 .NET 和 C#。在年底返回家乡过圣诞节的航班上，他灵感突发，开始用微软的 C# 编译器来开发自己的 C# 语言编译器。对，这个全新的编译器就是使用当时仍在公开测试阶段的 Visual Studio .NET 和 .NET Framework 开发出来的。

Miguel 之所以尝试开始这个方面的研究，首先就是看中了微软在 CLR 和组件模型上面的优秀设计。CLR 并不针对 C# 一门语言设计，而是提供一个相对中性的运行环境和对象模型，为大量开发语言（包括动态语言）提供支持，这使得跨语言的开发者大合作由梦想变成现实。.NET 的组件模型相比于 Miguel 之前为了 GNOME 项目而设计的 Bonobo 来说要更加简单明了。这些都可能在未来给 GNOME 项目的开发带来帮助。

另外或多或少还要感谢微软公司在 .NET 平台技术方面表现出的难得开放性。2000年8月，微软在宣布 .NET 技术两个月后，就联合惠普和英特尔将 C# 和 MSIL 相关技术细节提交给 ECMA 委员会，将它变成了开放的标准。Miguel 自己的业余工程就是按照这些文件的描述一步步去实现基础功能。四个月后在丹麦哥本哈根召开的 GUADEC 2001 会议的间隙，他还向一些爱好者演示了这些成果。这时他做的 C# 编译器已经可以编译它本身的源代码。

到了这年的6月，Miguel 终于做好准备去向全世界宣布了这个全新的开源项目 Mono，将 C# 语言和 CLR 技术带到 Linux 平台 。这是他参加 Open Source Convention 此行的主要目的，通过展示他对于 Mono 项目各个部件的设计和规划，来吸引参会的开源开发者加入其中。十六年之后再看那一页页的开发计划，依然会让人觉得震撼——这样一个大规模的工程，就在过去这些年间一点一点实现，而且取得了大大超越当时计划的成果。

不过 Mono 这个项目的名字，倒是非常有趣。在 Miguel 的母语西班牙语中，它是猴子的意思。不知道套用《西游记》的说法是否合适，对于这个新项目来说，2001年真是石猴出世的一年。一经面世，它也开启一段灵动的故事，有起有伏，高潮迭起。

虽然 Miguel de Icaza 明智地选择了使用 C# 语言和 Visual Studio .NET 来开发 Mono C# 编译器，使这个部件的开发在时间上节省了不少，但是对于该项目的更多部件，尤其是像 CLR 这类部件，是不得不用 C/C++ 等底层语言来开发。

幸好 Ximian 本来的主业 GNOME 便是使用 C/C++ 开发，相关的经验可以快速迁移过来，而且从更大范围来看此类人才在 Linux 社区里也是很容易找到的。即使是垃圾回收这样稍微复杂些的技术，因为已经有大量相关论文等公开资料，实际做起来也不像登陆月球那般困难。在起步时 Mono 项目选择了公开的 Boehm-Demers-Wiser Conservative Garbage Collector 算法，而直到 Mono 3.0起才逐步打造更为高效的 SGen 新算法。

由于计划合理，开发顺利，到8月29日一个极为简单的 Hello World 程序就已经能在 Mono 自己的环境中顺利运行起来。等时间快进到2002年6月，Mono 项目主要组件都已经初步成型，并且可以独立在 Linux 平台上面完成部署和自我编译。Mono 到这时终于可以向扶

了自己一路的 Windows 和 .NET 告别，踏上独立发展的道路。这时的项目范围，即将跳出微软先前提交给 ECMA 的开放标准束缚，向着更大的领域进军。

首先 Mono 团队要去封装 Linux 平台各种原生 API，例如把 GNOME 图形库 GTK 通过平台绑定技术变为 C# 程序可以直接使用的 GTK# 框架。有了基础的图形库和其他基础类型库，Mono 就能立马变成一个 Linux 桌面程序开发的好工具。正是此时这些基础工作，日后成功吸引到 Banshee 和 Tomboy 等热门应用的开发者。

其次是一谈到开发桌面程序，我们不能回避一个问题，“什么时候 Mono 项目能有一个类似 Visual Studio 的 IDE 环境呢”？这个想法看似简单，实现起来却并不容易，毕竟 Linux 平台上已经存在的 IDE 环境，例如 Eclipse 和 KDevelop 是 Java 或者 C++ 开发的，都不能拿来给 Mono 项目使用。但是另一方面，开发这个 IDE 能够带来实际的好处：

- 这是一个不小规模的桌面程序，可以证明 Mono 和 GTK# 的可靠性。
- 这个程序能够立即为 Mono 项目自身开发带来一些实际便利，例如语法高亮、界面设计和程序调试等等。

2003年底，终于有几个人按耐不住开始做这方面的工作。他们选择了 Windows 平台上逐渐成熟的 SharpDevelop 项目，尝试将这个轻便灵活的 IDE 迁移到 Linux，用 GTK# 界面逐步替换 Windowns Forms 界面。最终 MonoDevelop 这个新 IDE 加入了 Mono 大家庭。SharpDevelop 项目的主程序员 Mike Krüger 也从此成为 MonoDevelop 项目的主力成员。

Mono 1.0 版本在2004年6月瓜熟蒂落，开始正式支持 Linux 平台上 GTK# 和 ASP.NET 两类程序的开发。不过这时 Mono 项目的主导方 Ximian 出了一点变化。它不再是一家独立公司，而是被 Novell 公司收归旗下。

Novell 公司历来被看做是微软的死敌，因为双方的服务器软件产品 Netware 和 Windows NT Server 曾经在企业市场上有场你死我活的拼杀。由于微软再次施展自己擅长的产品组合拳，Novell 公司最后战败，Netware 黯然退出历史舞台。

这次挫折使得 Novell 像 IBM 公司一样重新审视当时蒸蒸日上的 Linux 平台，在2003年通过收购 SuSE Linux 等行动开始东山再起的计划。通过收购 Ximian 来获得 GNOME 和 Mono 两个项目的核心开发人员，对于它来说其实是顺理成章的事情。

换另一个视角，也是有了 Novell 的支持和 SuSE 项目的配合，Mono 团队得以在接下来的几年中稳健地发展，慢慢将微软私有的 ADO.NET 和 Windows Forms 等技术带到 Linux 平台。Mono 项目也跨出 Linux，涉足其他操作系统，例如 Mac 和 PlayStation。当然最后 Mono 还是通过及时登录 iOS 这个活力十足的移动平台，搭上了高速发展的快车。

Mono 登陆 iOS

在软件行业里面，电子游戏不是一个技术好就能挣钱的领域，因为好游戏必须要有好创意和好玩法，才能牢牢抓住玩家的心，让他们老老实实把真金白银投入进去。类似 Flappy

Bird 这样反潮流的现象级作品毕竟凤毛麟角。不过呢，在这个淘金者的乐土上，也有一些技术人员不需要自己亲自去辛苦淘金，他们通过为淘金者提供工具，形成了一个技术密集型的细分领域，这就是游戏引擎。

商业游戏引擎中知名的产品，比如老牌的 CryEngine，都是采用 C/C++ 这样的语言来实现，毕竟只有这样才能够完全释放硬件的计算能力。由于垃圾回收机制的限制，像 Java 和 C# 这类语言一向都很难应用在游戏引擎上。微软公司2004年3月24日发布的 XNA 框架算是一个有趣的尝试，让开发者可以使用 C# 以及 .NET 技术来创作游戏。即使 XNA 并不是非常强大，容易上手这个特点还是使它有了不少拥趸。

2008年开始和 Mono 团队进行技术合作的 Unity3D 公司，就是一家著名的游戏引擎供应商，早在2005年6月8日就推出了同名的游戏引擎。那么相比起 CryEngine 或者 XNA，Unity3D 引擎属于什么级别呢？简单来说，它很难在功能全面性上追上 CryEngine 之类高级产品，但是在上手难度上和 XNA 类似，绝对属于简单易用型。最为突出的是 Unity3D 非常看重多平台的支持，因此通过它来制作覆盖多个平台的游戏一直是特别亮点，对于游戏作者来说，能够快速将作品发布到所有主流游戏平台，比采用 XNA 那种绑定微软平台的引擎要灵便许多。

Unity3D 找到 Mono，肯定不是打算用 C# 来重写游戏引擎，而是它希望让 C# 成为开发游戏的脚本语言。这是个非常大胆的想法，一方面 C# 比起 Lua 这样简单的脚本语言来肯定特性丰富，可以给游戏开发工程师们送去天马行空恣意创作的一支神笔，而且搭载 Mono 不影响 Unity3D 的跨平台特性。但另一方面 Mono 作为一个开源项目，很多时候都是粗暴实现微软已有的 API，所以代码质量没有微软优秀，在性能上面还有着很多亟待提高的地方。

幸运的是，不论是 Unity3D 还是 Mono 团队对于合作都非常重视，Miguel 甚至还低调随 Unity3D 参加了2008年初的 Game Developer Conference 来实际体会游戏世界那种热闹场面。正式合作开始之后，Mono 团队在 Unity3D 的要求下立即着手开发了一些优化运行效率的新技术，这其中非常超前的就是 SIMD 指令集支持，于 Mono 2.2 版本开始正式加入，领先微软的同类技术好几年。

其实好消息还不止 Unity3D 方面。2008年8月，知名在线游戏 Second Life 宣布自己开始在服务器端软件中大规模使用 Mono 技术。这样这个游戏不仅支持原先的 LSL 脚本，而且开始支持 C#。在游戏领域，Mono 短时间内又取得一个重要胜利。

就在 Unity3D 和 Mono 接洽并启动合作前后，苹果公司在3月6日正式发布了 iPhone SDK，促使移动计算时代快速到来。出人意料的是，仅仅几周后的3月31日 Unity3D 就宣布自己搭载 Mono 的新版游戏引擎将在当年登陆 iPhone，难道技术上就没有丝毫障碍吗?

熟悉 iOS 平台的朋友们会知道 Java 或者 .NET 这类基于虚拟机的技术很难登陆这个平台。这其中一个难题是，基于虚拟机技术的程序需要在运行时由即时编译器（Just-in-time compiler/JIT）再次编译，而 JIT 的基本原理要求和 iOS 的安全设计（ASLR + “W^X” non-

executable memory policy）是相悖的。那么 Mono 团队在短短几周之内便为 Unity3D 解决了这个难题吗？那是个多大的工程啊！

其实得以顺利越过这个障碍，靠的是Mono 团队数年之前无心插柳而研究出来的一个独门绝技。嗯这个同时推动 Unity3D 和后来 MonoTouch 两个重要产品的新技术不是别的，就是 AOT（全名叫做 Ahead of Time Compilation）。

下面我们来简单看看 JIT 和 AOT 两个技术的区别。因为 C# 程序通过初次编译已经成为 PE32 格式文件中的 MSIL 指令，所以到了运行时 JIT 会一边读取 MSIL，一边把它翻译成机器码执行。这种执行方式有好有坏。一大好处是 JIT 可以针对运行时实际环境去实施优化，以生成高质量的机器码，达到全面提升程序性能的效果。可是同时一大坏处是在程序刚刚启动的那个阶段，JIT 必须花点额外的时间做初始编译，这会带来一个明显的延迟。简而言之，JIT 算法的好坏，对于程序运行的效率有着直接影响。

得益于他麾下一代代资深工程师了得的工程经验，微软的 .NET 就有一个不错的 JIT 算法实现。相比之下 Mono 开发者在这方面就很难企及微软的技术高度，从诞生之日起 Mono 的 JIT 实现就有些不太理想。于是2003年4月5日起，Mono 团队开始研发 AOT，换一个新的思路来提高 C# 程序的执行效率。

AOT 并不打算等到程序开始运行时才读取 MSIL 开始编译，而是直接在编译器生成 MSIL 之后就对 MSIL 进行再次编译，马上生成和目标平台对应的机器码。如此一来，真的到了运行时操作系统可以省去 JIT 过程，而直接加载这个程序，没有额外开销。更妙的是，如果把 Mono CLR/BCL 与开发者的应用捆绑到一起，那么这个经过 AOT 处理的程序就和使用 C/C++ 开发的原生程序没什么区别了。

当然 AOT 技术的研发并不像上面说起来这么简单，实际中还有很多技术挑战。而且在这个技术上面，Mono 团队没有了微软设计做参考，甚至放眼望去像 Java 之类的语言也没有类似的实现。所以直到2006年11月2号发布的 Mono 1.2 版本，这个新技术的独立研发才告一段落，成为一个正式的特性，从而被部分 Mono 程序拿去使用。其实这么好一个技术，如果仅仅拿来提高程序启动速度不免大材小用，只是那时谁也不会预见到它居然在数年之后的 iPhone 时代大显身手吧。

当 Unity3D 希望支持 iPhone 平台的时候，团队马上发现 AOT 是突破 iPhone 平台限制关键中的关键，开始更有针对性地仔细测试和完善起它来。到了10月1号，Mono 2.0 隆重发布，不仅包含与 Unity3D 合作之后专门研发的几项新技术，而且修正了很多以前版本的老毛病，特别是 AOT 方面的问题。两天之后，Unity3D 也顺势正式推出 Unity iPhone 这个新产品，帮助追随它的游戏开发者赶上了 iPhone/iOS 这趟快车。

这番研发结束，对于 Mono 团队来说，仿佛这个忙碌年份就要过去了，但是 Miguel de Icaza 这样的帅才会轻易满足吗？当然不是。既然 Unity3D 游戏借着 AOT 登陆 iPhone，那么更加通用的 C# 程序也可以登陆 iPhone 才对啊！

要知道那时 iPhone 上的 Objective C 程序开发起来并没有今天这么方便。一个尤为烦恼的问题出在内存管理方面。由于 ARC 技术当时还不支持 iPhone，开发者稍不留神，就容易忘记释放内存，出现程序崩溃等严重问题。而这类内存管理问题，一旦有了 C# 和 Mono 便可迎刃而解。

那么这个时候将 Mono 全面带到 iPhone 还有什么技术难点吗？Unity3D 登陆 iPhone，已经证明 Mono 项目的几个关键部件 AOT、CLR 和 BCL 都是可靠的。所缺少的一是 iPhone 原生 API 比如 CocoaTouch 的 C# 绑定。幸亏当时 iPhone SDK 中 API 还不是特别多，Mono 团队尚且可以完全通过手工编写绑定代码来应对这个挑战。另一个就是可视化开发支持。虽然不那么容易，但是通过更新 MonoDevelop，让它和 Xcode 中的 Interface Builder 工具巧妙集成，Mono 团队还是很快做出一个便利的开发体验。

就这样 Mono 团队一面继续配合 Unity3D 改进 Mono 运行时和 MonoDevelop，一面研发着自己的 iPhone 应用开发工具，时间也就不知不觉便到了2009年。再经过一段时间的封闭测试，Novell 在9月14号水到渠成地正式发布 MonoTouch 1.0。

相比 Objective C 开发 iPhone 应用，MonoTouch 具有下面几个巨大的优势：

1. 相比起设计差劲的 Objective C，C# 是一个更加简洁高效的语言。这也是后来苹果公司设计 Swift 语言的初衷之一。
2. CLR 能够自动收集垃圾，减少了内存管理导致的程序崩溃。
3. 拥有完善的原生 API 绑定，开发者可以使用全部底层 API。
4. 同样使用 Interface Builder，能够设计标准 iPhone 应用界面。
5. 能够复用已存在的很多 C# 代码，例如 .NET Framework 上面的三方程序库。

这中间还出过一个插曲。因为团队刚刚完成 Linux 和 Mac 平台 Windows Forms API 的跨平台实现，所以出现一种潜在可能将 Windows Forms API 迁移到 iPhone 上面。如此一来，可能一部分 Windows 程序（尤其 Windows CE 和 Windows Mobile 程序）就可以快速登录 iPhone 平台。

Miguel 斟酌再三还是放弃了这个方向，毕竟微软自己就是前车之鉴。微软为 Windows CE 平台做的 Windows Forms 实现，从用户界面角度说是严重缺乏美感的。要知道苹果从一开始就为 iPhone 应用制定了一套非常高的界面交互设计标准。假如 Mono 团队强行把 Windows 桌面的界面和交互方式带到手机平台，那么最后结果是几乎不可能达不到苹果要求的那个高度。

作为一款收费产品，MonoTouch 不容易像免费的 Xcode 那样吸引个人开发者，但对于在 C# 和 .NET 上面早已投入巨大的相关公司来说，它绝对是个福音，因为只需要花上很少的成本购买 MonoTouch，就能迅速开发出标准 iPhone 应用来打入新市场。果不其然，他们中间很快出现了第一波吃螃蟹的勇士，比如大名鼎鼎的美国国家仪器公司。

国家仪器的主打产品是一款叫做 LabView 的设计工具。用户只需要在这个工具界面上拖拽和简单配置，就能让一堆基础元器件连接成有特定功能的电子系统。它本来是一个运行在 Windows 上的 .NET 桌面程序。一般来说使用 LabView 的工程师，需要常常带个笔记本（那个时候大多数笔记本还是挺厚重的）去项目现场，帮助配置和调试电子系统。

苹果公司2010年推出了堪称神奇的 iPad 平板，在智能手机外又开启了平板这个新市场。LabView 团队慧眼识珠，立即有了让产品登陆到平板的强烈愿望。通过使用最新版的 MonoTouch，他们在极短的时间内就设计出了支持 iPad 平台的产品并推向市场。从此使用 LabView 的工程师们，只需拎着个轻便的小平板就可以上现场了，这是多么贴心的一件事情啊！

浴火重生的 Xamarin

Novell 是一家神奇的技术公司，它自行开发和收购了很多不错的软件和标准，涵盖 NetWare、IPX、WordPerfect、Quattro Pro，Unix，SUSE和Mono。它曾有心挑战微软的霸权，最后不免以失败收场，但在开源运动的发展历程中，它所扮演的角色无可取代：

- 1993年它收购 Unix System Laboratories 并由此获得 UNIX 版权，并在 SCO 诉 Linux 社区这个世纪大案中保卫了 Linux。
- 收购 Ximian 和 SUSE，使得 GNOME 和 Mono 等开源项目得到了一个相对宽松的发展环境并进入企业市场。
- 与微软达成了专利合作协议，并且增进产品互操作性，也促成了 Moonlight 项目开发过程中 Mono 团队和微软公司 .NET 团队之间的初次合作。

可是和 Sun 公司类似，因为运营状况不佳，2010年11月 Novell 被 Attachmate 财团以22亿美金价格收购。这家专业收购公司的财团很快开始重新组合各种资产，在大约半年后开始对 Novell 公司进行大规模裁员。各个产品组中 Mono 团队受到的影响最大，全部成员失去了饭碗。

裁员目标的选择当然体现了 Attachmate 管理层的市场判断。他们知道 SUSE 是个优质资产，必须加以保护。但他们没看好当时正值发布初期的 MonoTouch，对于尚在全力研发中的 Mono for Android 这个极有市场潜力的产品也是视若无睹。这种短视不仅给两个极有价值的项目立即带去负面影响，也直接导致相关技术支持的停止，已经购买相关产品的用户们发现竟然不能激活！

Miguel de Icaza 在这个关键时间做出了一个重要的选择，体现出了他强大的个人魅力和严谨做事的腔调。首先通过安抚人心，他成功地团结到 Mono 团队中各位主力成员，使他们再次聚首，然后立即给正在外国休假的 Nat Friedman 打了个紧急电话。这两个从1999年就精诚合作的老朋友一拍即合，自己投资来创建新公司来继续 Mono 系列产品的开发。和给 Mono 项目起名时一样，他们再次选择了猴子这个机灵的家族，把公司命名为 Xamarin（来自 Tamarin 柽柳猴属）。

Mono 因为是一个开源项目，Xamarin 员工继续参与其中并没有法律方面的障碍，但是 Attachmate 当时仍然掌握着 MonoTouch 等收费产品的代码版权，怎样通过合适的渠道拿到这个版权对于初生的 Xamarin 来说极为重要。

这里 Miguel 采用了一个有趣的方法，一面保留和 Attachmate 接洽的可能，一面利用 Mono 开源项目的源代码来从头开发 MonoTouch 和 Mono for Android 的模仿产品。为了避嫌，还专门由原来开发 MonoTouch 的几个工程师转去做 Mono for Android 的仿品，而参与预研 Mono for Android 的几个工程师开始从头重写 MonoTouch 的仿品。

大概是因为技术方面已经没有特别挑战，Xamarin 很快开始了自己新产品的封闭测试，准备在最短时间内将它们推向市场。很多老用户也开始给 Attachmate 施加压力，而潜在新用户则持币待购，焦急地等待新产品的发布。

2011年7月事情峰回路转，Attachmate 资源整合后刚刚独立的 SUSE 公司主动找到 Xamarin 来谈合作，并在一揽子协议签字后将 MonoTouch 和 Mono for Android的版权拱手相让。当然作为交换条件，Xamarin 公司将在一段时间内继续为 SUSE 客户提供 Mono 相关产品的技术支持服务。这个合作启动之后，Mono开源项目的主导权也正式由 SUSE 转交到 Xamarin 手中。

至此 Xamarin 终于卸下专利包袱，全力投入到下一代新产品的开发中。有趣的是，之前交叉开发的过程中据说团队里诞生了很多全新的想法。在拿回 MonoTouch 等产品的代码和版权后，这些新想法也逐步在后代产品之中一一实现。

值得注意的是和 Novell 时代的 Mono 团队相比，Xamarin 是一家更为纯粹的移动平台开发工具公司。很多带有 Novell 时代印记的老技术明显不再大幅改进，失去了活力，包括：

- libgdiplus 和 Mono 的 Windows Forms 实现
- Mono for Visual Studio，一个在 Visual Stdio 中开发和远程部署调试 Mono 程序的插件
- Moonlight，开源的 Silverlight 实现

之后的数年间，Xamarin 迅速地补全了自己的产品线，在 iOS/Android 平台之外又加入 macOS 桌面程序开发支持，加入使用大量物理手机设备搭建的移动应用单元测试云平台 Xamarin Test Cloud，加入 Xamarin.Forms 跨平台界面框架，从而获得了与微软 Visual Studio 团队紧密合作的机会。有这些铺垫在前，随后发生在2016年2月的收购便实在不出意料了。

想了解更多 .NET/Mono/Xamarin 的历史故事吗？请查阅我的新书《.NET 传奇》。

官方网站 http://dotnet.sxl.cn

你也可以同时关注我的播客 dotnet.fm

官方网站 http://podcast.sxl.cn