### 发展

LLVM是构架编译器的框架系统，以C++写成，用以优化任意语言程序的编译、链接、运行时间的开源免费工具。

LLVM计划启动于2000年，由UIUC大学的Chris Lattner博士主持开展。2006年获得Apple Inc.的支持，并且LLVM被苹果公司广泛使用。

2012年，LLVM获得2012年ACM软件系统奖。

### 作用

提供了编译器，代码静态检查工具等实用开发工具。

### 优点

**现代化的设计**

高度模块化，使得代码更清晰更易于调试。

**语言无关的中间代码**

LLVM能够

**引用**[**https://blog.csdn.net/khlljm/article/details/51822973**](https://blog.csdn.net/khlljm/article/details/51822973)

**你或许应该知道的LLVM**

转载 2016年07月04日 18:57:30

* 标签：
* [LLVM](http://so.csdn.net/so/search/s.do?q=LLVM&t=blog)
* 5659

[原文链接](http://www.vienta.me/2014/06/11/%E4%BD%A0%E6%88%96%E8%AE%B8%E5%BA%94%E8%AF%A5%E7%9F%A5%E9%81%93%E7%9A%84LLVM/)

作为iOS或者Mac开发者，你也许非常眼熟LLVM这个字眼，但也许没有太去在意它。在很长的一段时间内，我就是处于这个状态，不知道它背后是在干嘛。随着苹果新语言swift的发布，我看到“Swift 是克里斯在 LLVM 和 Clang 之后第三个伟大的项目”，可见，这门swift的出现是建立在LLVM和Clang的基础之上的。激发我对于LLVM等的好奇，于是进行了一些探索研究学习。

**渊源**

这里不得不提到一个伟大的理想主义者——理查德·斯托曼。此人是名典型的黑客，在上个世纪80年代，黑客社群在软件工业商业化的强大压力下日渐土崩瓦解，这与他所追求的目标——可自由流通的软件的伟大理想格格不入，他对此感到气愤与无奈。于是在1985年的时候发表了著名的[GNU宣言](http://zh.wikipedia.org/wiki/GNU),它的目标是创建一套完全自由的操作系统GNU。这吸引了一帮神一般的程序员无偿（当然后面也有的有偿聘请）的参与到这个计划中来，从1985年开始，到了1990年的时候，GNU计划就开发出了一个功能强大的文字编辑器Emacs,C语言编译器GCC以及很多UNIX系统的程序库和工具。但是唯一，唯一没有完成的重要组件，称之为HURD的操作系统内核。

说到这，要提下**GPL——GNU通用公共许可证**（所有GNU软件都包含一份在禁止其他人添加任何限制的情况下，授权所有权利给任何人的协议条款），也就是“公共版权”概念。是为了保证GNU软件可以自由地“使用、复制、修改和发布”。上面说道一个唯一没有完成的组件——操作系统内核，但在1991年的时候，芬兰的牛大学生林纳斯·托瓦兹编写出了与UNIX兼容的Linux操作系统内核，并且是在GPL条款下发布的。想不到这个Linux大火，在1992年的时候，Linux与很多GNU软件结合，于是，完全自由的操作系统正式诞生。历史就是这么的奇特，GNU自己的内核Hurd还在开发中。这个有点扯远了，我们回到上文提到的GCC,因为这个和本文的关系很大。

**源头——GCC和GDB**

**GCC**——GNU编译器套装，是一个很强大的编译器。这个工具被移植到各种系统中，其中就包含了Mac OSX操作系统，并且成为其标准的编译器。这可以反映在Xcode中，Xcode早期的时候用的就是GCC，如果开发iOS比较早的同学肯定能够注意到这点。不仅仅如此，在早期（其实也很长时间）的Xcode中，调试代码用的一个工具是**GDB**。GDB是什么？它是GNU调试器（缩写：GDB）,GNU软件系统当中的标准调试器。它也是被移植到MacOSX系统当中，并且配合GCC作为其除错工具。Xcode早期的话我们在设置断点的时候会看到（gdb）的字样，例如你可以输入po打印对象。

**可是，为什么还是LLVM**

先来看段Xcode的版本历史：

Xcode3之前，用的是GCC

Xcode3,GCC仍然保留，但是也推出了LLVM，苹果推荐LLVM-GCC混合编译器，但还不是默认编译器

Xcode4,LLVM-GCC成为默认编译器，但GCC仍保留

Xcode4.2,LLVM3.0成为默认编译器,纯用GCC不复可能

Xcode4.6,LLVM升级到4.2版本

Xcode5,LLVM-GCC被遗弃，新的编译器是LLVM5.0，从GCC过渡到LLVM的时代正式完成

现在的Xcode，已经完全用的是LLVM了。那么，**LLVM是什么**？它的命名源自底层虚拟机（Low Level Virtual Machine）的缩写。它是一个编译器的基础建设，是为了任意一种编程语言写成的程序，利用虚拟技术，创造出编译时期，链结时期，运行时期以及“闲置时期”的优化。最早是以C/C++为实现对象，后来支持了Objective-C。LLVM项目由维克拉姆·艾夫和克里斯·拉特纳于2000年发起。LLVM最初被用来取代现有于GCC stack的代码产生器[13]，许多GCC的前端已经可以与其运行。例如在Xcode的LLVM GCC 4.2编译器时期，核心是LLVM，但是前端是GCC4.2。简单来说，LLVM可以作为多种语言编译器的后台来使用。

我们知道，克里斯·拉特纳后来被苹果收入麾下。在他毕业的时候，也就是2000年左右，苹果为了编译器正焦头烂额，按照苹果一贯的追求完美的风格，肯定会要求GCC对Objective-C进行优化。不过这帮开源界的人可能太忙，没有买苹果的帐。好的是，这个时候LLVM正悄然崛起，苹果看到了另外一个稻草，狠狠的抓牢了，将LLVM的项目发起者收下，并且将编译器后台彻底的从GCC转向了LLVM。

LLVM既然可以作为很多语言编译器的后台来使用，自然就引发了一些人为专门的语言开发新的编译器，其中一个最引人注目的就是——**Clang**。Clang(发音为ˈklæŋ)是一个C、C++、Objective-C和Objective-C++编程语言的编译器前端，它的目标就是提供一个GNU编译器套装（GCC）的替代品。这个正是苹果需要的。当然，Clang也是克里斯做的，苹果收了这样的人才确实是很有眼光，不愧为伟大的公司。现在的Xcode已经是LLVM+Clang编译器。举个例子，在某个历史Xcode版本(4.x)的编译选项中出现过两种编译器：Apple LLVM compiler 4.2和LLVM GCC 4.2。Apple LLVM compiler 4.2的意思是LLVM编译器，前端用的是Clang。LLVM GCC 4.2的意思则是编译器核心是LLVM，但是前端使用的是GCC4.2编译器。

有了LLVM+Clang，从此，苹果的开发面貌焕然一新。从此摆脱了GCC的限制。客观的说GCC是有很多的优点，例如支持多平台，很流行，基于C无需C++编译器即可编译。这些优点到苹果那就可能是缺点了，苹果需要的是——快。这正是Clang的优点，除了快，它还有与GCC兼容，内存占用小，诊断信息可读性强，易扩展，易于IDE集成等等优点。有个测试数据:Clang编译Objective-C代码时速度为GCC的3倍。

我们在说到GCC的时候提到了GDB，说它能够纠错，我们也谈到Clang向比较GCC有更好的诊断纠错能力，相对应的，Clang下纠错工具就是**LLDB**。iOS或Mac开发，现在用到的纠错工具就是LLDB。熟悉的同学可以注意到，在设置断点程序跑到断点时，在控制台中就会出现(lldb)字样。没错，就是这么近距离的接触LLDB。为了说LLDB的一个优点，我们先说下GDB的一个限制——“用户界面”。GDB没有一个不错的GUI，默认只有命令行接口（CLI）可用，没那么亲合上手。虽然有几个前段程序为其补强，但还是差强人意。GDB的这个缺点在LLDB上没有，所以LLDB的一个优点就是有一个良好的GUI。下面，我们来谈谈LLDB。

**LLDB，有点意思**

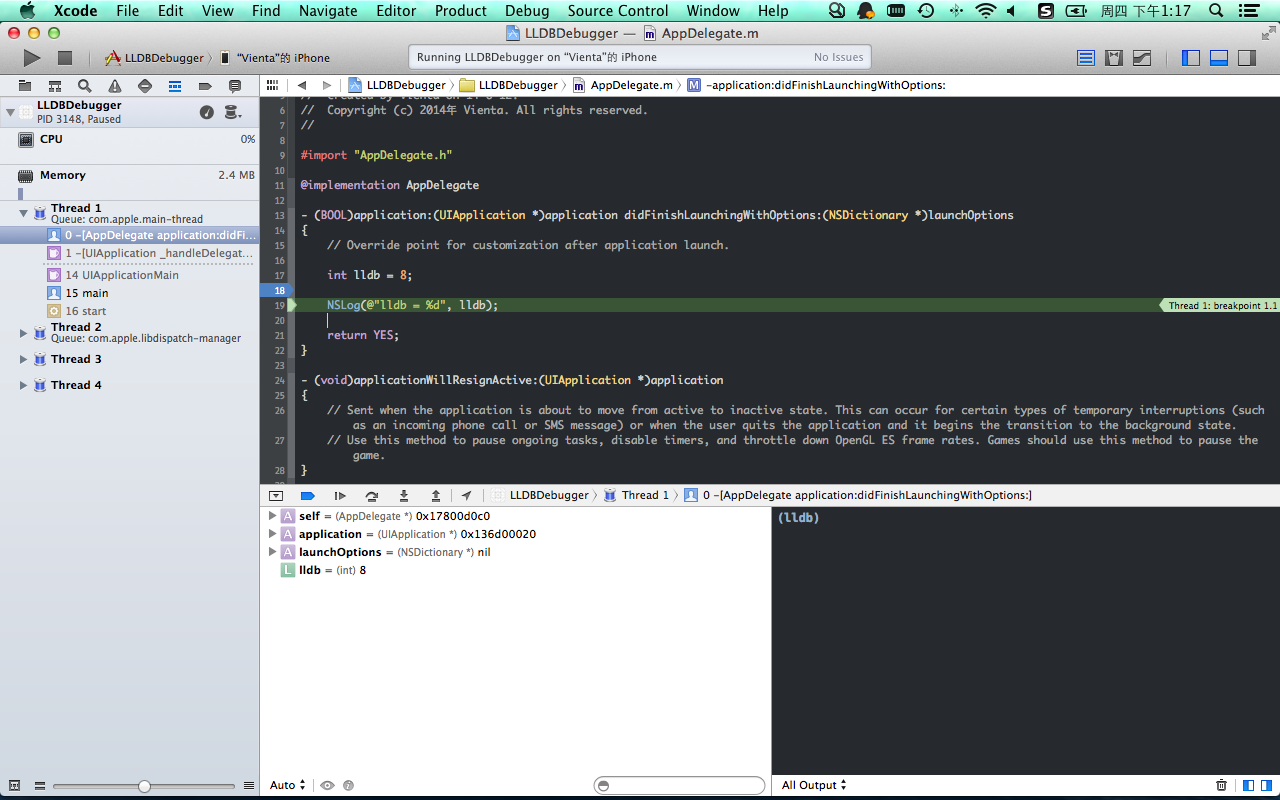
上面说到LLDB,有个良好的GUI。其实它继承了GDB的优点，弥补GDB的不足。iOS开发者从gbd过渡到lldb没有任何不适应感，最直白的原因就是lldb和gdb常用的命令很多都是一样的，例如常用的po等。[The LLDB Debugger](http://lldb.llvm.org/lldb-gdb.html)列举了GDB和LLDB的命令。

使用GDB的除错人员可以监督及修改程序内部的变量值，甚至监督与修改独立于主程序运作外，以独立个体型态调用（调用使用）的函数。同样的，LLDB肯定有这个功能。我们不妨做个小实验。  
打开Xcode新建一个demo工程。在AppDelegate里面。我们这么样写：

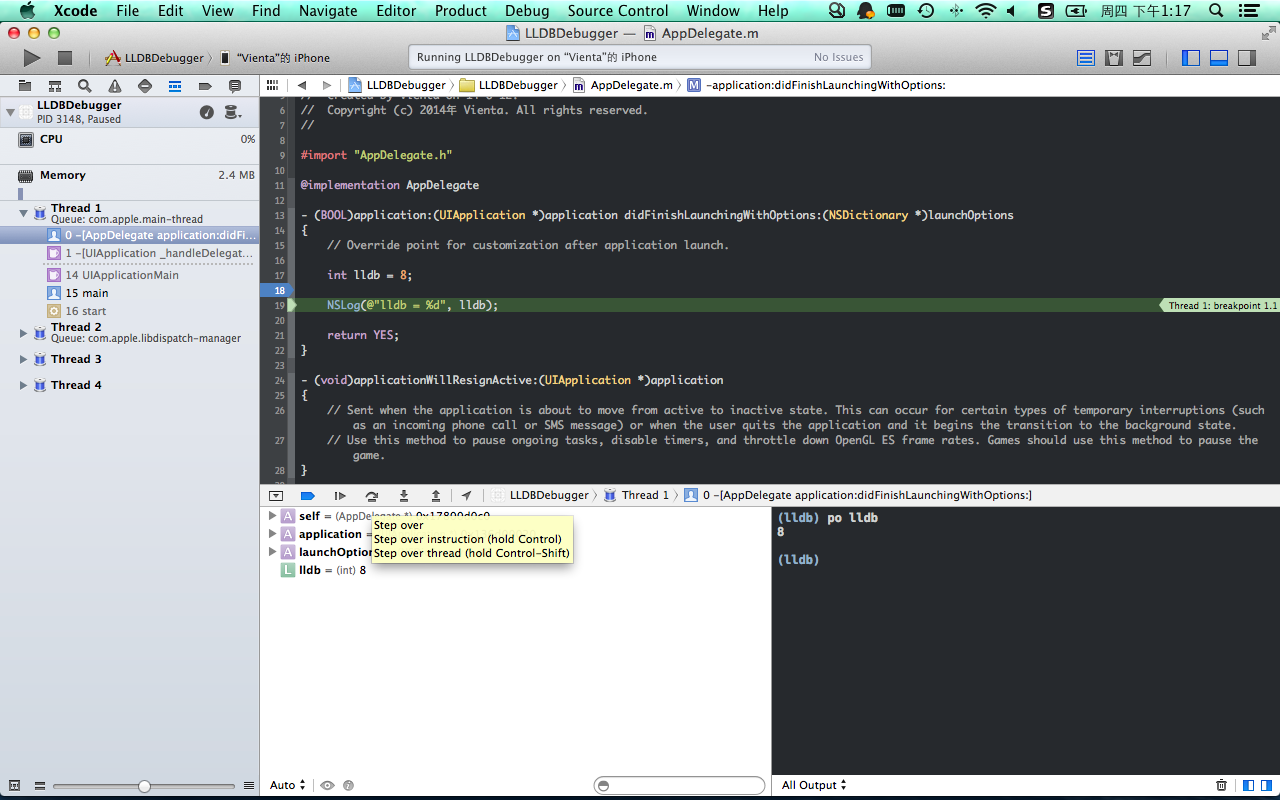
int lldb = 8;

NSLog(@"lldb = %d", lldb);

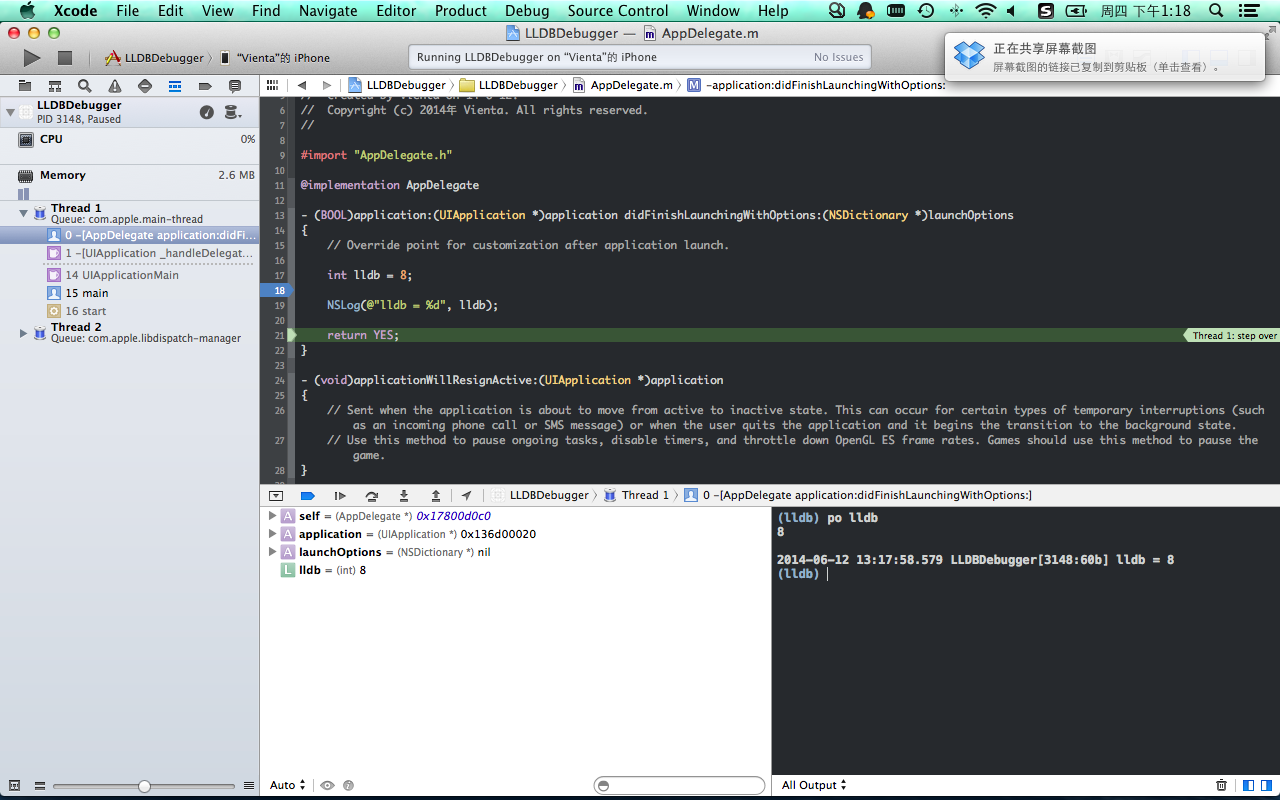
在18行设置了端点。好，run起程序，程序执行到了端点处，看到了常见的（lldb）：

[](http://www.vienta.me/img/llvm1.png)

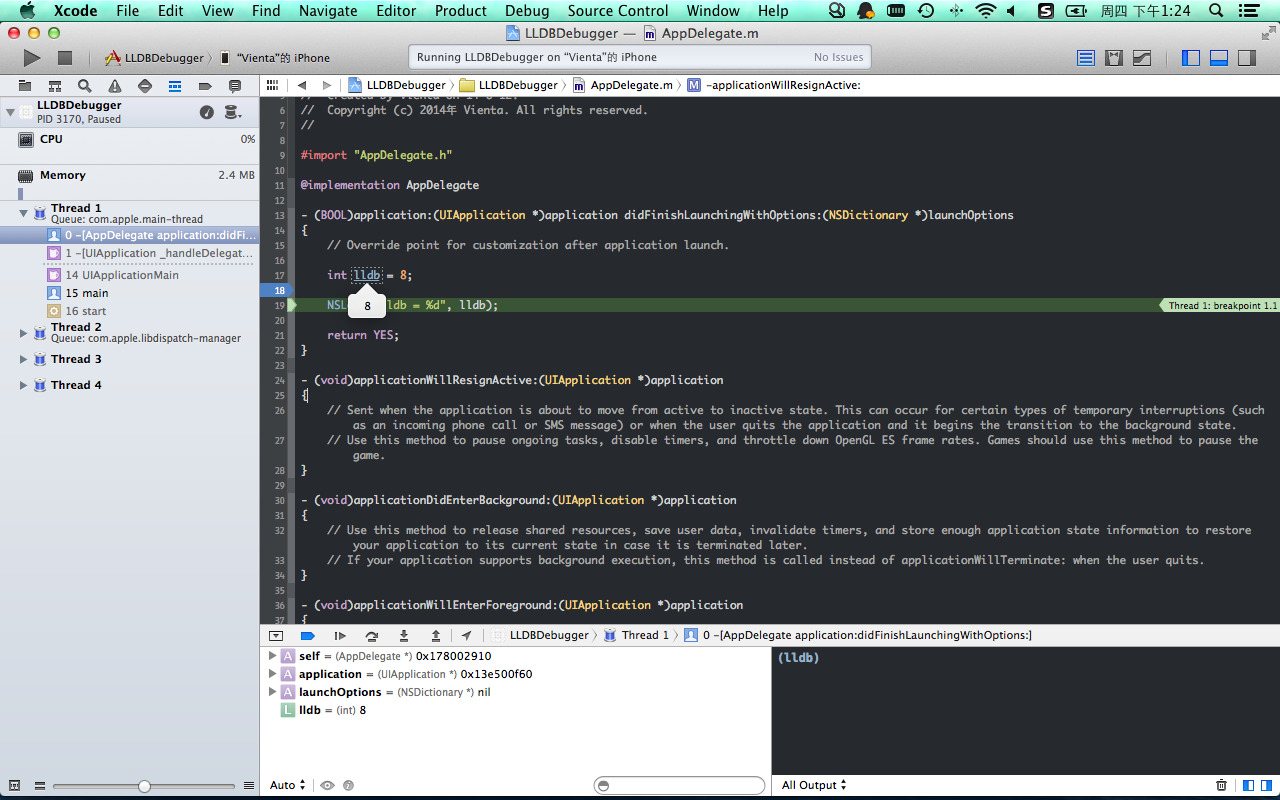
p下lldb，可以见到：

[](http://www.vienta.me/img/llvm3.png)

点下step next[http://www.vienta.me/img/llvm2.png](http://www.vienta.me/img/llvm2.png)按钮，NSLog打印出了lldb的值：

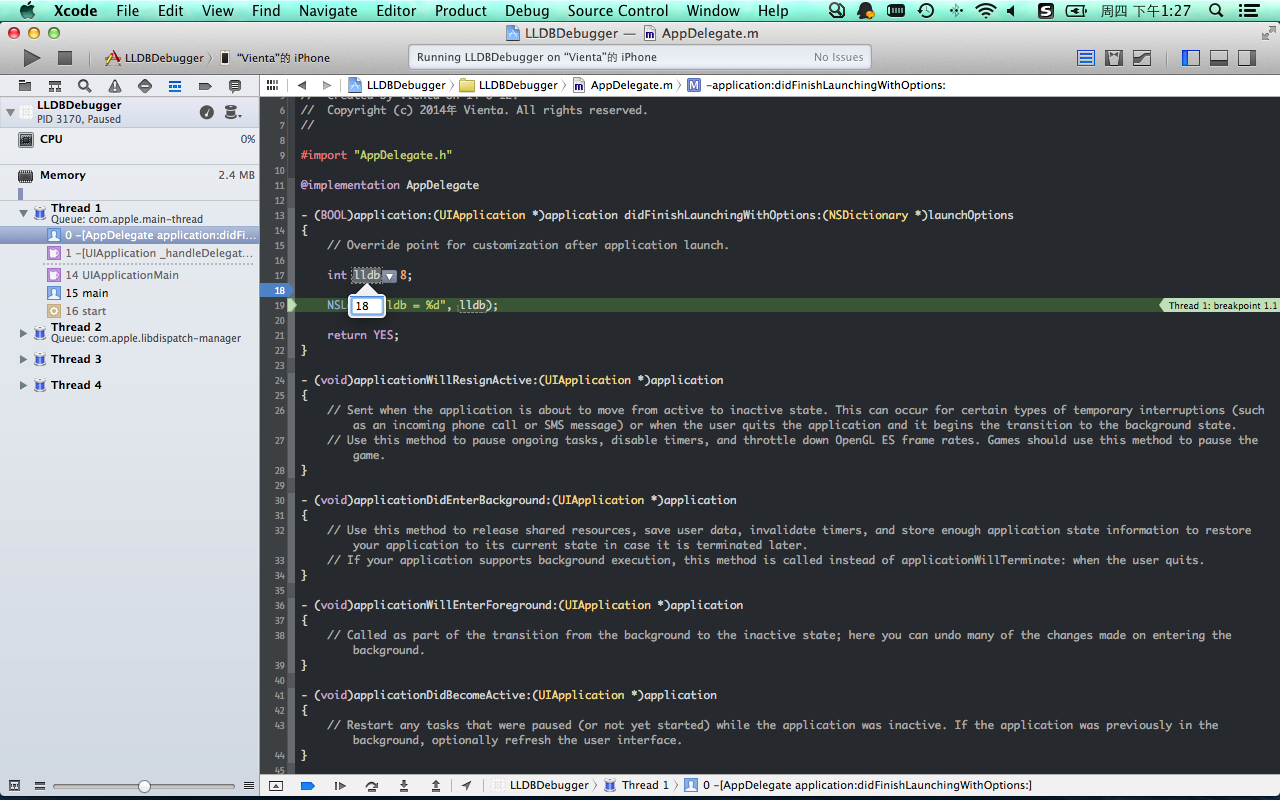
[](http://www.vienta.me/img/llvm4.png)

好了，上面是常规的流程，现在玩点稍有意思的。我们重新跑下程序，到了断点处。这个时候我们将鼠标光标停在lldb上稍微停留一段时间。这时你能看到如下：

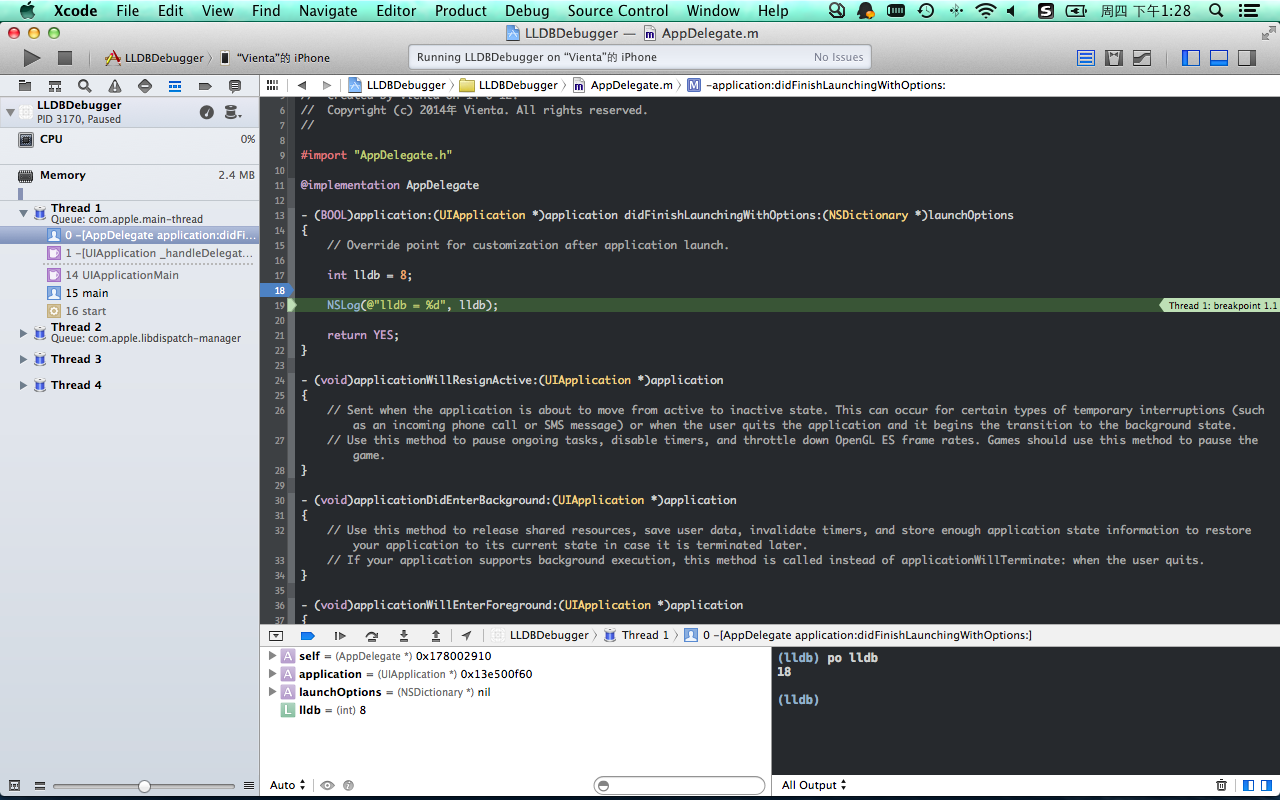
[](http://www.vienta.me/img/llvm5.png)

Great！这其实就是LLDB的可视化之处。

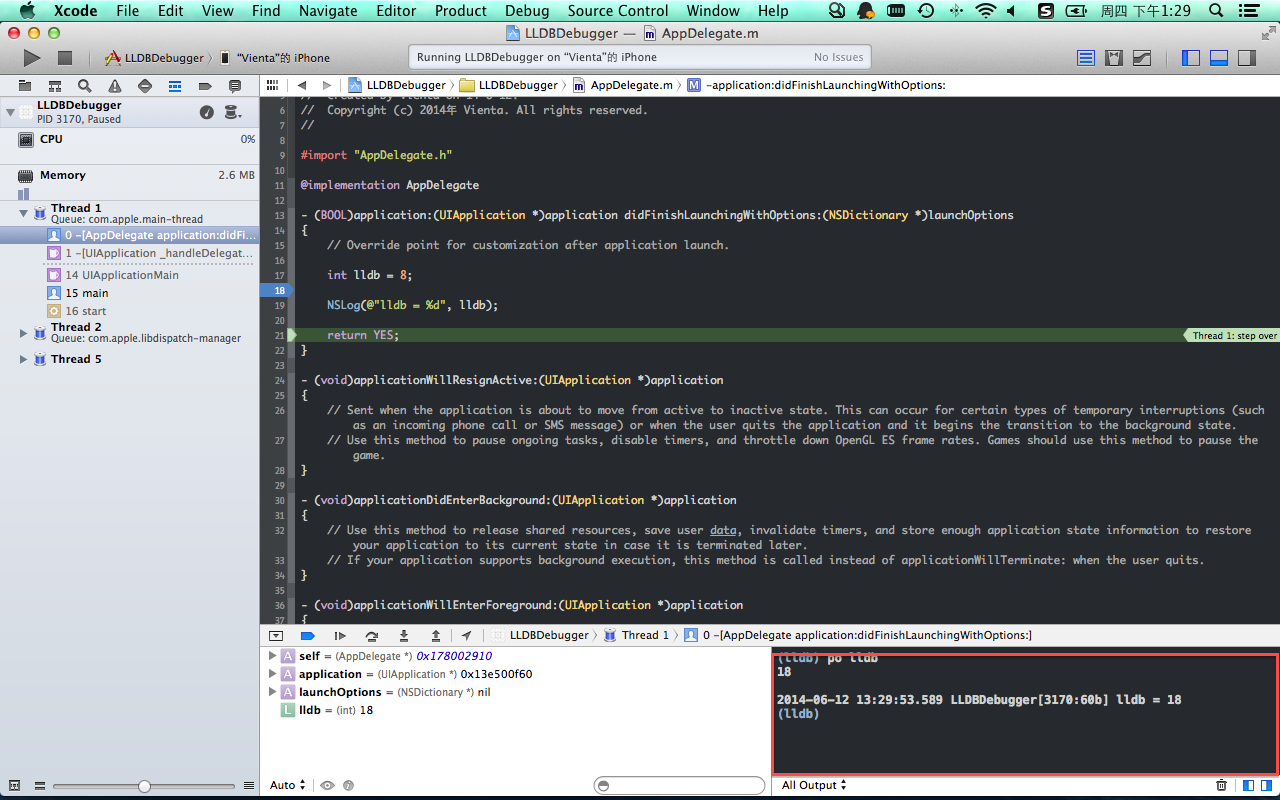
这样还不够，还能接着玩，在弹出白色小框上点击，即可进入编辑，将8改为18。如下图：

[](http://www.vienta.me/img/llvm7.png)

现在我们再po一下：

[](http://www.vienta.me/img/llvm8.png)

Amazing!lldb变成了18。我们点下step next[http://www.vienta.me/img/llvm2.png](http://www.vienta.me/img/llvm2.png)按钮，看到NSLog出来的也是18。

[](http://www.vienta.me/img/llvm16.png)

真的改变了lldb的值了。

我们还可以拿对象做个示范。在上次的代码中接着加上：

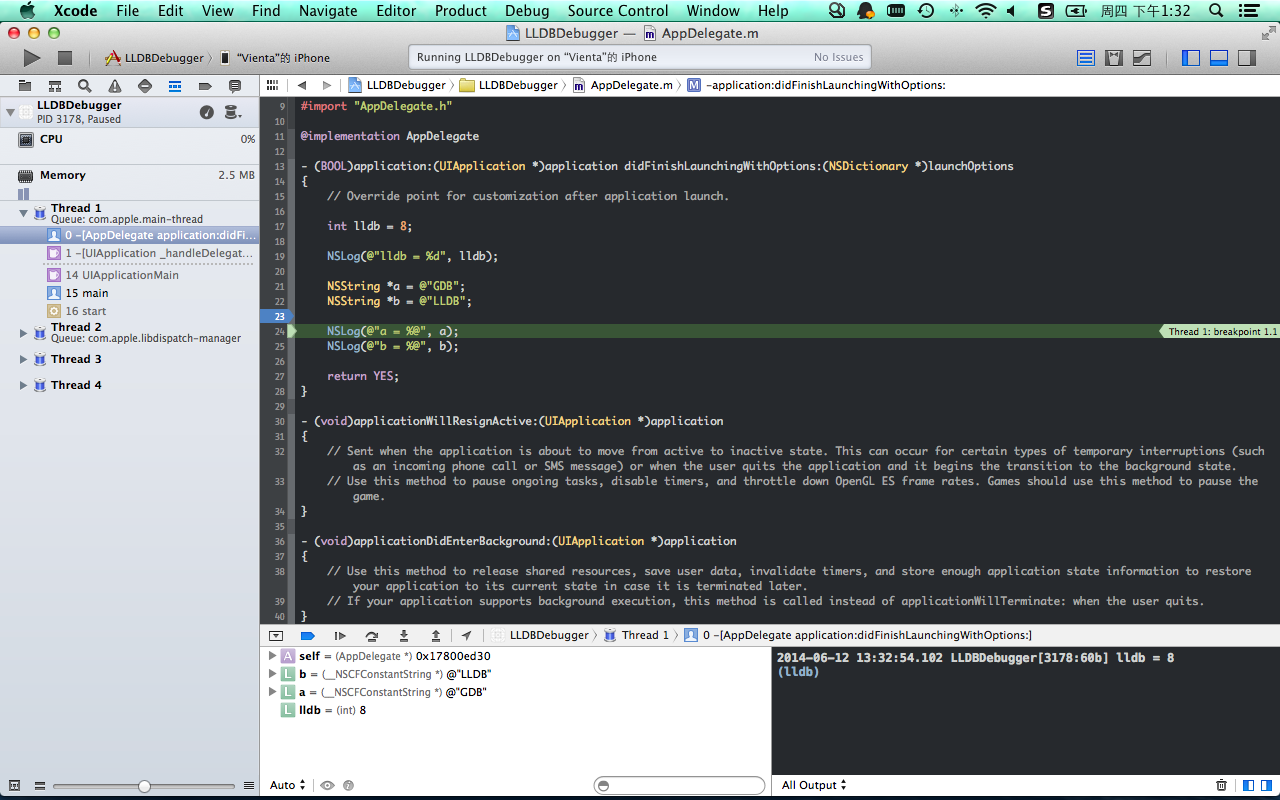
NSString \*a = @"GDB";

NSString \*b = @"LLDB";

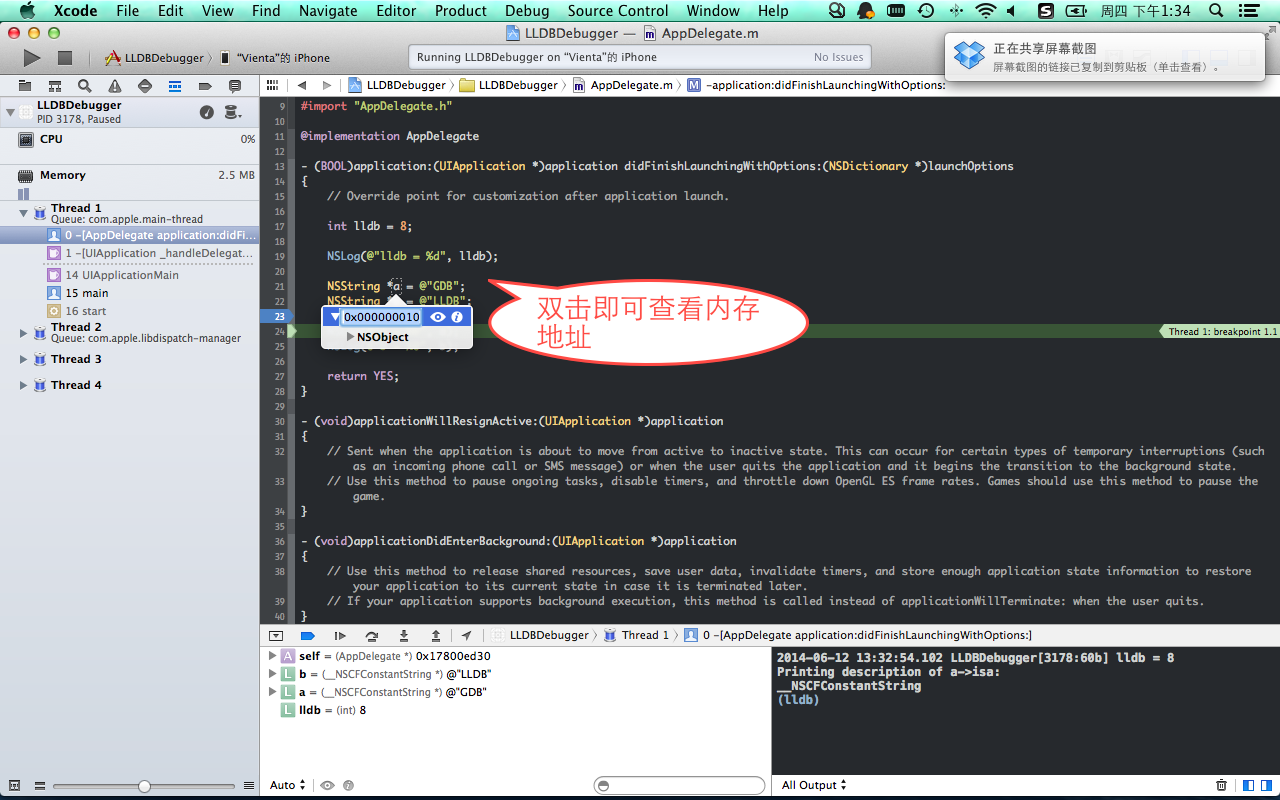
NSLog(@"a = %@", a);

NSLog(@"b = %@", b);

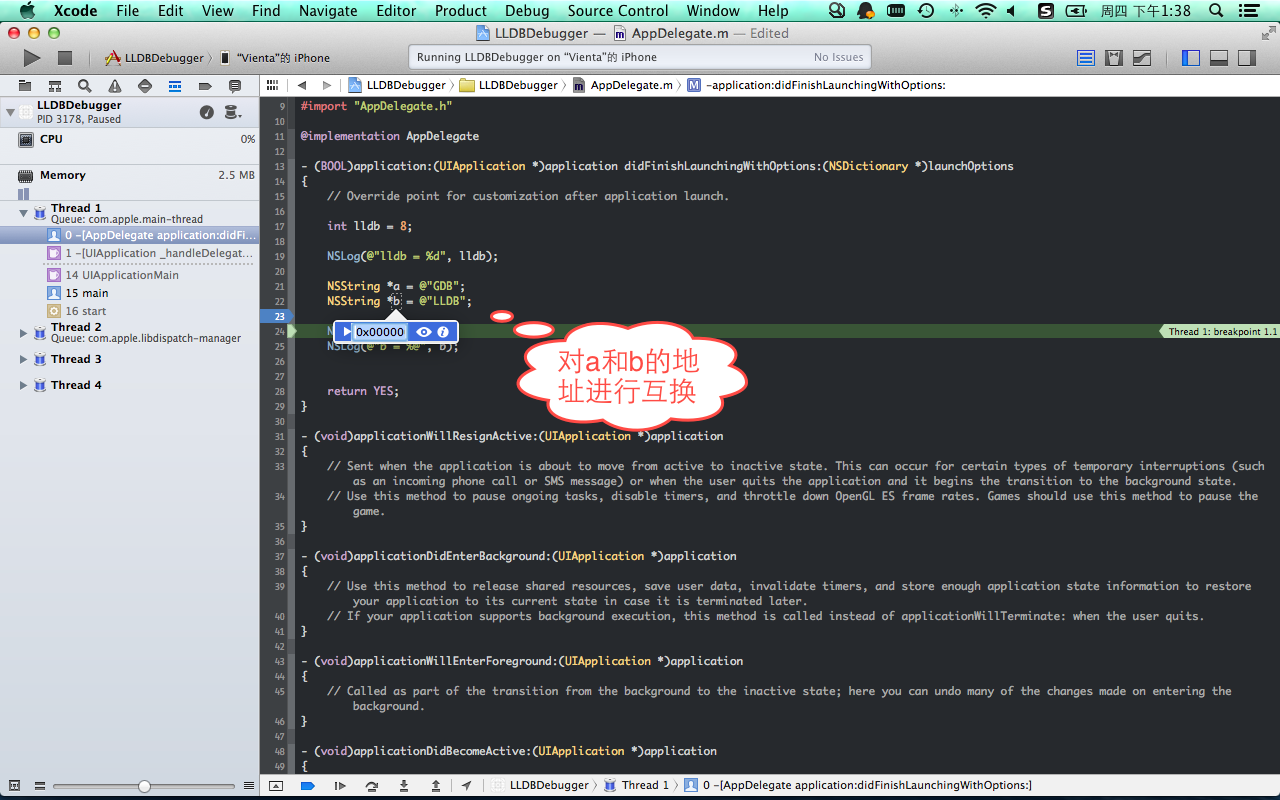
把断点调到23行。run起来，断在断点处，如下图：

[](http://www.vienta.me/img/llvm9.png)

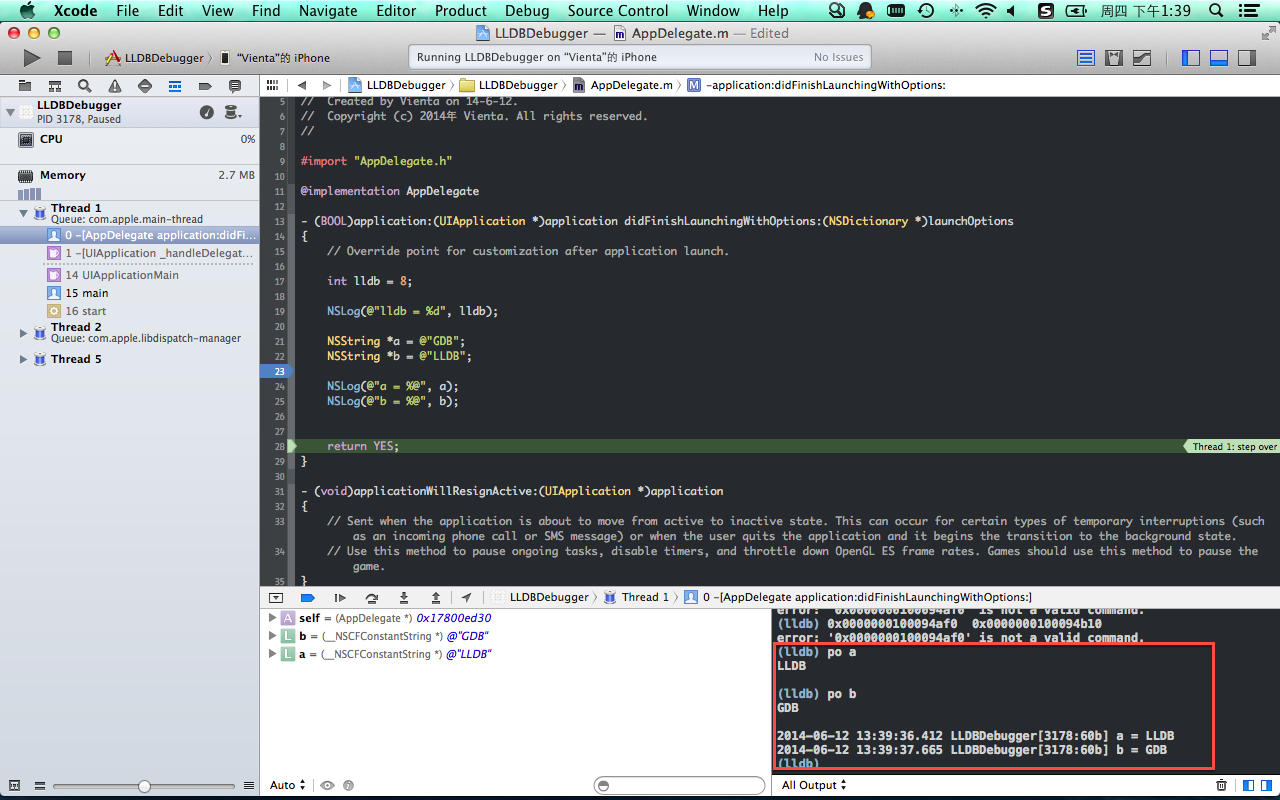
我们按照对待lldb那样的也对待a和b，我们看到弹出的小白框，双击后我们得到了a和b对象在内存中的地址：0x0000000100094af0和0x0000000100094b10

[](http://www.vienta.me/img/llvm11.png)

我们编辑a对象，将b的地址0x0000000100094b10 拷贝到a上。编辑b对象，将a的地址0x0000000100094af0 剪切到b上。如图

[](http://www.vienta.me/img/llvm13.png)

好了。我们再po一下a和b。看到了把，a和b的值发生了变化。点击next step[http://www.vienta.me/img/llvm2.png](http://www.vienta.me/img/llvm2.png),同样的NSLog出来的值也发生了变化。

[](http://www.vienta.me/img/llvm15.png)

上面演示说明了LLDB同样可以修改内部程序的值。并且有良好的GUI。在Xcode5之后UIImage对象的图片都可以用上面的方法展示出来，也许作为开发者的你早就发现了。实际上，还有更强大的功能等着我们去发掘。

苹果的这次发布会发布了新的语言——**swift**。也是克里斯的杰作。我们看到了playground，它能够在编码的同时实时预览输出。有段对playgound的描写：“Playground 不仅实现了很多脚本语言支持的交互式编程，而且提供控制台输出、实时图形图像、时间线（timeline）变量跟踪等功能，开发者除了可以看到代码的实时运行结果，还能根据时间线阅读某个变量在代码片段中值的变化。这真是太棒了！”。这些功能只能在Xcode提供的playground文件中实现。支撑这个的背后应该就是LLVM+Clang。本次发布会发布的swift、HomeKit、HealthKit和之前的M7协同处理器，让我对Apple的未来充满了好奇。从GCC到LLVM，纵观苹果的历史，稳扎稳打，致力于打造出自己的生态圈，有战略的推进着自己，改变着世界。也为自己是一名iOS开发者感到高兴，庆幸自己生在一个不那么好但却迅速变革的时代。

引用Apple的主题结束这篇文章——**“Write the code, Change the world!”**

注：本文参阅不少文章，在自我理解的基础上写作而成，如有任何错误欢迎留言反馈，不甚感激。另，转载需注明出处。