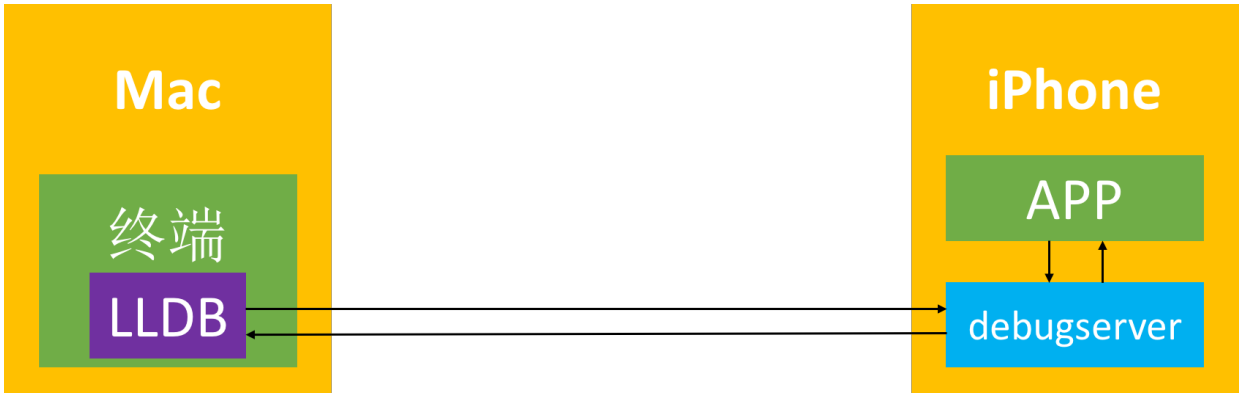
一、什么叫动态调试?

> 将程序运行起来，通过下断点、打印等方式，查看参数、返回值、函数调用流程等

二、Xcode的动态调试原理

- 关于GCC、LLVM、GDB、LLDB
  - Xcode的**编译器**发展历程：GCC → LLVM
  - Xcode的**调试器**发展历程：GDB → LLDB
- debugserver一开始存放在Mac的Xcode里面
  - **/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport/9.1/DeveloperDiskImage.dmg/usr/bin/debugserver**
- 当Xcode识别到手机设备时，Xcode会自动将debugserver安装到iPhone上
  - **/Developer/usr/bin/debugserver**
- Xcode调试的局限性
  - 一般情况下，只能调试通过Xcode安装的APP

三、动态调试任意APP



debugserver的权限问题

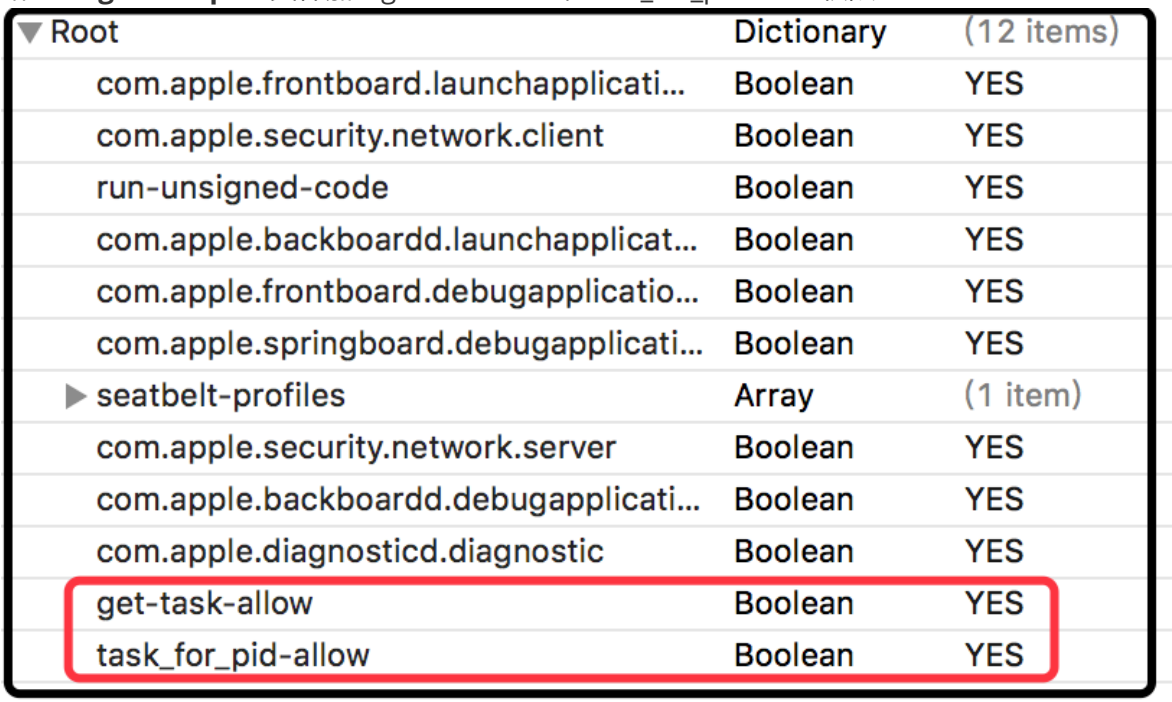
- 默认情况下，**/Developer/usr/bin/debugserver**缺少一定的权限，只能调试通过Xcode安装的APP，无法调试其他APP（比如来自App Store的APP）
- 如果希望调试其他APP，需要对debugserver重新签名，签上2个调试相关的权限
  - get-task-allow
  - task_for_pid-allow

如何给debugserver签上权限

- iPhone上的**/Developer**目录是只读的，无法直接对**/Developer/usr/bin/debugserver**文件签名，需要先把debugserver复制到Mac
- 通过ldid命令导出文件以前的签名权限

```
$ ldid -e debugserver > debugserver.entitlements
```

- 给**debugserver.plist**文件加上get-task-allow和task_for_pid-allow权限



- 通过ldid命令重新签名

```
$ ldid -Sdebugserver.entitlements debugserver
```

- 将已经签好权限的debugserver放到**/usr/bin**目录，便于找到debugserver指令
- 关于权限的签名，也可以使用codesign

```
# 查看权限信息
$ codesign -d --entitlements - debugserver

# 签名权限
$ codesign -f -s - --entitlements debugserver.entitlements debugserver
# 或者简写为
$ codesign -fs- --entitlements debugserver.entitlements debugserver
```

让debugserver附加到某个APP进程

```
$ debugserver *:端口号 -a 进程
```

- ***:端口号**
  - 使用iPhone的某个端口启动debugserver服务（只要不是保留端口号就行）
- **-a 进程**
  - 输入APP的进程信息（进程ID或者进程名称）

在Mac上启动LLDB，远程连接iPhone上的debugserver服务

- 启动LLDB

```
$ lldb
(lldb)
```

- 连接debugserver服务

```
(lldb) process connect connect://手机IP地址:debugserver服务端口号
```

- 使用LLDB的c命令让程序先继续运行

```
(lldb) c
```

- 接下来就可以使用LLDB命令调试APP

通过debugserver启动APP

```
$ debugserver -x auto *:端口号 APP的可执行文件路径
```

四、常用LLDB指令

- 指令的格式是

```
<command> [<subcommand> [<subcommand>...]] <action> [-options [option-value]] [argument [argument...]]
```

  - ：命令
  - ：子命令
  - ：命令操作
  - ：命令选项
  - ：命令参数
  - 比如给test函数设置断点

```
breakpoint set -n test
```

    - **breakpoint**是
    - **set**是
    - **-n**是
    - **test**是

- **help**
  - 查看指令的用法
  - 比如**help breakpoint**、**help breakpoint set**

- **expression --**
  - 执行一个表达式
    - ：命令选项
    - **--**：命令选项结束符，表示所有的命令选项已经设置完毕，如果没有命令选项，--可以省略
    - ：需要执行的表达式

```
expression self.view.backgroundColor = [UIColor redColor]
```

  - **expression**、**expression --**和指令**print**、**p**、**call**的效果一样
  - **expression -O --**和指令**po**的效果一样

- **thread backtrace**

    - 打印线程的堆栈信息
    - 和指令**bt**的效果一样

- **thread return []**
    - 让函数直接返回某个值，不会执行断点后面的代码

- **frame variable []**
    - 打印当前栈帧的变量

- **thread continue**、**continue**、**c**： 程序继续运行
- **thread step-over**、**next**、**n**： 单步运行，把子函数当做整体一步执行
- **thread step-in**、**step**、**s**： 单步运行，遇到子函数会进入子函数
- **thread step-out**、**finish**： 直接执行完当前函数的所有代码，返回到上一个函数
- **thread step-inst-over**、**nexti**、**ni**
- **thread step-inst**、**stepi**、**si**
- **si**、**ni**和**s**、**n**类似
    - **s**、**n**是源码级别
    - **si**、**ni**是汇编指令级别

- **breakpoint set**
    - 设置断点
    - **breakpoint set -a 函数地址**
    - **breakpoint set -n 函数名**
        - `breakpoint set -n test`
        - `breakpoint set -n touchesBegan:withEvent:`
        - `breakpoint set -n "-[ViewController touchesBegan:withEvent:]"`
    - **breakpoint set -r 正则表达式**
    - **breakpoint set -s 动态库 -n 函数名**

- **breakpoint list**
    - 列出所有的断点（每个断点都有自己的编号）

- **breakpoint disable 断点编号**： 禁用断点
- **breakpoint enable 断点编号**： 启用断点
- **breakpoint delete 断点编号**： 删除断点

- **breakpoint command add 断点编号**
  - 给断点预先设置需要执行的命令，到触发断点时，就会按顺序执行
- **breakpoint command list 断点编号**
  - 查看某个断点设置的命令
- **breakpoint command delete 断点编号**
  - 删除某个断点设置的命令

- 内存断点
  - 在内存数据发生改变的时候触发
  - **watchpoint set variable 变量**
    - `watchpoint set variable self->age`
  - **watchpoint set expression 地址**
    - `watchpoint set expression &(self->_age)`
  - **watchpoint list**
  - **watchpoint disable 断点编号**
  - **watchpoint enable 断点编号**
  - **watchpoint delete 断点编号**
  - **watchpoint command add 断点编号**
  - **watchpoint command list 断点编号**
  - **watchpoint command delete 断点编号**

- **image lookup**
  - **image lookup -t 类型**： 查找某个类型的信息
  - **image lookup -a 地址**： 根据内存地址查找在模块中的位置
  - **image lookup -n 符号或者函数名**： 查找某个符号或者函数的位置
- **image list**
  - 列出所加载的模块信息
  - `image list -o -f`
    - 打印出模块的偏移地址、全路径

- 小技巧
  - 敲Enter，会自动执行上次的命令
  - 绝大部分指令都可以使用缩写