程序沙盒快速入门

在正式学习前，选择一款 OS 应用程序并在沙盒中运行它。然后开始学习如何解决典型的程序沙盒错误。这里你需要用到 Xcode、Keychain Access、Activity Monitor 以及 Console.

创建 Xcode 项目

在快速入门中创建的应用使用的是 WebKit 网页视图，因此需要一直连接网络。在 App Sandbox 下，网络连接需要你特别允许。

- **如何为 Quick Start 创建 Xcode 项目…**
  1. 在 Xcode4 下，为 OS X Cocoa 应用程序创建一个新的 Xcode 项目
     - 命名为 AppSandboxQuickStart
     - 如无，创建一个公司鉴别符，如 com.yourcompany
     - 确保除 Use Automatic Reference Counting 外的其他选项没有勾选
  2. 在项目导航中，选择 MainMenu nib 文件。
     弹出 Interface Builder 画布。
  3. 在 Xcode 码头，点击 Window 对象
     这样程序窗口在画布上可见。
  4. 在选项库中（在工具区），定位 WebView 对象。
  5. 拖曳网页视图到画布的窗口上。
  6. （可选）通过以下步骤改善运行程序的网页视图显示：
     - 拖动网页视图的尺寸控制使其完全填充窗口主视图
     - 使用网页视图的大小检测，确保所有的内部和外部自动调整大小已激活
  7. 在 AppDelegate 类别下为网页视图创建和连接一个外接口。在 Xcode，使用如下设置：

Outlet connection source	The `WebView` object of the `MainMenu` nib file.
Outlet variable location	The interface block of the `AppDelegate.h` header file.
Outlet name	webView
Storage	weak

  8. 添加 WebKit 框架到应用中。
     - 在 AppDelegate.h 头文件的界面块上添加以下语句来导入 WebKit 框架
       ```
       #import <WebKit/WebKit.h>
       ```
     - 将 WebKit 框架作为必需框架链接到 Quick Start
  9. 添加以下 awakeFromNib 方法到 AppDelegate.m 执行文件中：

```
- (void) awakeFromNib {
```

```
    [self.webView.mainFrame loadRequest:

        [NSURLRequest requestWithURL:

            [NSURL URLWithString: @"http://www.apple.com"]]];

}
```

应用程序启动时，这个方法要求从电脑网络连接指定的 URL，然后发送结果到网页视图显示。

现在，创建和运行应用——这个应用还没有被沙盒，所以能自由访问包括网络插口在内的系统资源。确定应用窗口显示的页面是你在 awakeFromNib 方法中设置的。完成后，退出应用。

开启 App Sandbox

在 Xcode 目标编辑器中勾选复选框来开启 App Sandbox。

在 Xcode 中，点击项目导航中的项目文件，并点击 AppSandboxQuickStart 目标。查看目标编辑器的 Summary 标签。

- **如何为项目开启 App Sandbox**
  1. 在目标编辑器的 Summary 标签中，点击 Enable Entitlements.

     Entitlement 是定义在属性清单文件中的键-值对，它授予目标一个特定的功能或安全许可。

     当你点击 Enable Entitlements 时，Xcode 会自动检查 Code Sign Application 和 Enable App Sandboxing 复选框。如此，这些都是在 App Sanbox 开启设置的关键项目。

     当你点击 Enable Entitlements 时，Xcode 会在项目导航中自动生成一个可见的.entitlements 属性清单文件。当你在目标编辑器中使用图解的 entitlements 界面时，Xcode 会更新属性清单文件。
  2. 清空 iCloud entitlements 中的内容

     Quick Start 无需用到 iCloud，因为当你 enable entitlements 的时候 Xcode 会自动添加 iCloud 权利值。按以下方法删除：
     - 在目标管理器的 Summary 标签中，选择并删除 iCloud Kay-Value Store 中的内容
     - 点击 iCloud Containers 栏的首行，点击减号按钮

到此，你已经开启了 App Store 但还没有为 Xcode 项目提供一个代码签名身份（code signing identity）。因此，如果你现在开始创建项目，那么注定是要失败滴~随后的两部分会帮助你搞定它！

创建一个测试用 Code Signing Certificate

为了在 Xcode 中创建一个有沙盒的应用，你必须有 code signing certificate 以及它相应的私人密钥，然后在项目中使用该证书的 code signing identity。当你创建项目时，你设置的 entitlements 包括开启 App Sandbox 的 entitlement 会成为程序代码签名的一部分。

你将在本部分创建一个代码签名证书。简化的过程让你能专注于开启沙盒的步骤。

重点：你在这里创建的代码签名证书在将发布的应用中并不适用。在你沙盒需发布应用之前，先读读“App Sandbox and Code Signing.”。

- **如何创建 App Sandbox 测试用代码签名证书**
  1. 在 Keychain 通道（在 Application/Utilities 中），选择 KeyChain Access > Certificate Assistant > Create a Certificate，打开证书助手。
     注意：在调用“Create a Certificate”菜单命令之前，确保在 Keychain Access 主窗口没有选择任何密钥。如果有一个密钥被选中了，那么菜单命令不可用。
  2. 在 Certificate Assistant 中，命名证书，如 My Test Certificate.
  3. 如下配置证书：

Identity type	Self Signed Root
Certificate type	Code Signing
Let me override defaults	unchecked

  4. 点击 Create
  5. 在弹出的警告框中点击 Continue
  6. 在结束窗口点击 Done

这样，新的代码签名证书和它相应的公共和私人密钥在 Keychain Access 中就可以用了。

指定 Code Signing Identity

现在，配置 Xcode 项目来使用你在前面创建的证书中的代码签名身份。

- 为项目指定代码签名身份
  1. 查看项目编辑器中的 Build Settings 标签。
     注意你使用的是项目编辑器（project editor）而非目标编辑器（target editor）
  2. 在 Code Signing 部分，选定 Code Signing Identity 行
  3. 点击 Code Signing Identity 行的赋值区
  4. 在弹出的菜单中选择 Other
  5. 在打开的文本输入窗口中，输入刚创建的代码签名证书的名称，然后按<return>
     如果你使用的是建议的名称，那么你需要输入的名称就是 My Test Certificate

现在你就可以创建应用了。Codesign 可能会弹出一个对话框请求允许使用新的证书。如果该警告出现，点击 Always Allow.

确认应用已被沙盒

创建并运行 Quick Start 应用。如果程序没有成功被沙盒，那么不会显示任何网页内容。这是因为你还没有允许连接网络。

除此之外，你可以根据另两种标志判断程序是否被成功沙盒。

- **如何确认 Quick Start 程序被成功沙盒**
  1. 在 Finder，查看 `~/Library/Containers/`文件夹下的内容
     如果快速入门的程序被沙盒，就会有个以你应用命名的文件夹。文件夹名称包括项

目的公司鉴别符，所以完整的名称可能是 com.yourcompany.AppSandboxQuickStart.
使用者第一次运行程序的时候，系统会为该用户创建一个程序文件夹。

2. 在 Activity Monitor, 检查系统识别程序为已沙盒。
   - 启动 Activity Monitor（/Application/Utilities 中）
   - 选择 View > Columns
     确认 Sandbox 菜单项已选
   - 在 Sandbox 栏，确认 Quick Start 程序的值是 Yes
     在 Filter 栏输入 Quick Start 中程序的名称，帮助你更快捷的在 Activity monitor 中找到应用。

技巧：如果在你尝试运行程序时它崩溃了，特别是收到 `EXC_BAD_INSTRUCTION` 的提示，最有可能的原因是你曾经运行过一个与之有着相同 bundle identifier 不同代码签名的沙盒应用。这个启动崩溃是 App Sandbox 安全功能在防止一个程序假冒另一个程序从而进入其他程序的 container（容器？？）

解决 App Sandbox 违例

当程序想做些 App Sandbox 不允许的事情时，就会出现 App Sandbox 违例。例如，Quick Start 中的沙盒程序能从网上检索内容。访问系统资源的微粒度限制是 App Sandbox 保护的核心。它能保护程序不被恶意代码干扰。

App Sandbox 违例最常见的原因是 Xcode 中的 entitlement 设置与程序实际需要不符。通过本部分内容，你能过研究从而改正 App Sandbox 违例。

- **如何诊断 App Sandbox 违例**

1. 创建并运行 Quick Start 应用
   程序正常启动，但是不能显示 awakeFromNid 方法中设置的网页（如你先前在 "Confirm That the App Is Sandboxed"观察到的）由于在应用被沙盒之前，网页的显示是正常的，这种情况下有理由怀疑是一个 App Sandbox 违例。
2. 打开 Console（/Application/Utilities 中），确认边栏的 All Messages 被选中。
   在 Console 窗口的过滤栏，输入 sandboxd，仅仅显示 App Sandbox 违例。
   Sandboxd 是 App Sandbox 后台程序的名称。这个后台程序会报告 App Sandbox 违例。
   相关的信息会呈现在 Console 上，如下：

   ```
   ▶ 3:56:16 pm sandboxd: ([4928]) AppSandboxQuickS(4928) deny network-outbound 111.30.222.15:80
   ▶ 3:56:16 pm sandboxd: ([4928]) AppSandboxQuickS(4928) deny system-socket
   ```

   产生这些信息的原因是 Quick Start 还没有获得访问外网的权利。
   技巧：查看任一违例完整的后台轨迹，点击相应 Console 信息右边的回形针。

前面的步骤阐明了识别 App Sandbox 违例的一般方式：

1. 确认违例仅发生在 App Sandbox 被开启后
2. 引起违例（如通过尝试连接网络）
3. 查看 Console 中的 sandboxd 信息。

这里同样有一种简单通用的方法来解决此类违例。

- **如何通过添加恰当的 entitlement 来解决 App Sandbox 违例**

1. 退出 Quick Start 程序
2. 在目标编辑器的 Summary 标签中，查找与 sandboxd 违例相对应的 entitlement。
   如，主要错误是 `deny network-outbound.`，相应的 entitlement 是 All Outgoing Network

Connections.

3. 在目标编辑器的 Summary 标签中，选中 Allow Outgoing Network Connections 复选框。这样做能为所需的 entitlement 应用一个 TRUE 值到 Xcode 项目中。
4. 创建并运行应用。

理想中的网页现在能显示在应用上了。另外，Console 上也不再有新的 App Sandbox 违例消息了。

App Sandbox 全析

App Sandbox 用来保护用户数据的访问控制机制很简洁且易懂。但采用 App Sandbox 的具体步骤却是唯一的。确定这些步骤之前，你必须对这项技术的关键概念有所了解。

需要最后防线

你按照 *Secure Coding Guide* 推荐的那样保护你的应用免受恶意软件的攻击。但是尽管你竭尽全力设置了无懈可击的安全壁垒——避免缓冲区溢出和其他存储侵蚀，防止用户数据泄露，减少其他缺陷——你的程序还是受到恶意代码的攻击。攻击者只需找到你防线或者任何一个你连接的框架和库中的一个漏洞就能控制程序与系统的互动。

App Sandbox 就是为应对这一情况量身打造的。它让你描述你的程序想要与系统进行的互动。这样系统只会授予程序它完成任务所需的访问权限。即使恶意代码控制了已沙盒的程序，它也只能访问应用沙盒中的文件和资源。

要想成功的采用 App Sandbox，使用与往常不一样的思维习惯，如表 2-1 建议：

开发时……	采用程序沙盒时……
增加功能	最小化使用系统资源
利用程序的所有使用权	分割功能，然后 distrust（不信任？）每个部分
使用最方便的 API	使用最安全的 API
视限制为局限	视限制为安全

但你设计 App Sandbox 的时候，你要考虑到最糟的情况：尽管你竭尽全力，恶意代码还是突破一个无意识的安全漏洞——要么在你的代码中要么在你连接的一个框架中。你添加给程序的功能可能成为恶意代码的功能。在阅读以下文档时要牢记这一点。

容器目录和文件系统访问（Container Directories and File System Access）

但你采用 App Sandbox 时，系统你程序专用的特使目录，这个目录就称为 container。系统上的每个用户在他们的主目录下都会有一个你程序的个人容器。你的程序拥有自由的读取/写入当前用户的容器的访问权。

App Sandbox 容器目录

Container 具有以下特性：

- 它位于用户主目录下的系统定义路径，你可以通过调用 NSHomeDirectory 功能获取。
- 程序拥有自由读取/写入容器及其子目录的访问权。
- OS X 路径搜索 APIs（POSIX 层上）特指你程序的位置。
  这些路径搜索 APIs 的大多数都指的是你程序容器的相关位置。例如，容器里有一个你程序专用的个人 Library 目录（由 NSLibraryDirectory 搜索路径常量指定），包括个人 Application Support 和 Preferences 子目录。
  使用容器支持文件无需改写程序早期沙盒版本中的代码，但可能需要一次移植。具体内容参看"Migrating an App to a Sandbox."
  有些路径搜索 APIs（POSIX 层上）指的是用户主目录外的程序专用位置。如，在已沙盒的程序中，NETemporaryDirectory 功能提供了一个路径到目录。这个目录是程序专用的位于主目录外沙盒内的。当前用户拥有自由读取/写入它的访问权。这些路径搜索 API 的行为完全适用于 App Sandbox 而无需任何代码改变。
- OS X 通过程序代码签名构建和加强程序和它的容器之间的联系。
- 容器位于一个隐藏路径，所以用户不会与它直接互动。要特别说明的是，容器包含的并不是用户文档，而是包含程序所用的文件如数据库，缓冲和其他程序专用数据。
  对于鞋盒式的程序，你只为用户内容提供了唯一用户界面的，那些内容会在容器中而你的程序拥有它的完全访问权。
  iOS 注释：由于 OS X 容器不是为用户文档而存在的，所以它不同于 iOS 容器。在 iOS 中，容器是唯一的用户文档储存位置。另外，iOS 容器包含程序本身。而 OS X 则不然。
  iCloud 注释：在苹果的 iCloud 技术中也提到了"容器"，这个容器和 App Sandbox 容器没有功能上的关联。

多亏了代码签名，其他任何已沙盒的程序都不能访问你的容器，即使它尝试用你的 bundle identifier 乔装成你的程序。但假使你在程序的后期版本中使用的是相同的代码签名和 bundle identifier, 它会使用你程序的容器。

当程序初次运行时，一个开启了沙盒功能的程序的容器目录就会生成。由于容器是在用户主文件夹下的，所以系统上的每个用户都会拥有自己的程序容器。当用户初次运行你的程序时，相应的容器就会产生。

权力箱和容器外部的文件系统访问（Powerbox and File System Access Outside of Your Container）

通过以下三种方式，已沙盒的程序能访问它容器外的文件系统路径：

- 通过用户的具体方位
- 通过使用具体的文件系统路径的 entitlement（详见 "Entitlements and System Resource Access"）
- 当文件系统路径在某一全部可读的目录下

Powerbox 是指扩展沙盒功能的能与用户互动的 OS X 安全技术。权力箱没有 API.当你使用 NSOpenPanel 类和 NSSavePanel 类时，程序会用到 Powerbox。你可以通过用 Xcode 设置一个 entitlement 来开启 Powerbox 功能。

当你从已沙盒程序调用一个 Open 或者 Save 对话框时，弹出的窗口不是由 AppKit 呈现而是由 Powerbox. 当你采用 App Sandbox 时，Powerbox 的使用是自发的并无需修改代码。你打开或保存时用到的附带面板会如实地释放和使用。

注意：当你采用沙盒时，NSOpenPanel 类和 NSSavePanel 类的一些重要行为会改变。详见 "Open and Save Dialog Behavior with App Sandbox."

Powerbox 的安全益处在于它不能在程序上被操控。也就是说，没有机制供敌对代码使用来通过 Powerbox 访问文件系统。只有通过 Powerbox 与 Open 和 Save 对话框互动的用户才能通过该对话框访问沙盒外的部分文件系统。如，如果用户保存了一个新的文件，Powerbox 会扩展你的沙盒，使你能拥有读取/写入该文件的访问权。

当你程序的用户指明他们想使用某一文件或文件夹时，系统会添加相关的路径到你的程序沙盒中。如，用户将 `~/Documents` 文件夹拖到程序的 Dock 图标上（不是 Finder 图标或程序的某一打开窗口上），由此表明他们需要用该文件夹。系统会允许你的程序使用 `~/Documents` 文件夹，里面的内容和子文件夹。

如果用户只是打开一个特定的文件或者保存一个新文件，系统则仅允许你使用那个特定的文件。

此外，系统会自动允许已沙盒的程序：

- 连接系统输入法
- 调用用户在 Service 菜单中选用的服务（只有那些被服务提供商标记为安全的服务才可用）
- 打开用户在 Open Recent 菜单选择的文件
- 通过用户调用的复制和粘贴参与到其他应用中
- 读取目录下的通读文件，包括：
  - /bin
  - /sbin
  - /usr/bin
  - /usr/lin
  - /usr/sbin
  - /usr/share
  - /System

当用户指定了他们想要用的文件后，该文件就会在你程序的沙盒中。但由于程序沙盒没有对其采取任何保护措施，如果你的程序受到恶意代码的侵蚀，该文件就很容易收到攻击。要想为沙盒中的文件提供保护，参看 *Secure Coding Guide*.

遵照用户意图的一个关键方面就是 OS X 不允许模拟或改变用户的输入。这其中包含有辅助程序的内容，详见"Determine Whether Your App Is Suitable for Sandboxing."

默认情况下，打开或保存的文件会一直在沙盒中直至程序终止。但如果程序终止时，该文件是打开的，它就不会被删除。下次程序启动的时候，此类文件会自动打开并再次自动添加沙盒中。这是 OS X 的恢复功能。

如果想要一直拥有对容器外资源的访问权，则需要用到 "Security-Scoped Bookmarks and Persistent Resource Access."中提到的安全范围书签（Security-scoped bookmark）。

App Sandbox 中的打开和保存对话框动作

当程序开启了 App Sandbox 功能，NSOpenPanel 和 NSSavePanel 方法会变的不同：

- 不能用 `ok:` 方法调用 OK 按钮
- 不能从 NSOpenSavePanelDelegate 协议用 `panel:userEnteredFilename:confirmed:` 方法重写用户选择

此外，有效的 NSOpenPanel 和 NSSavePanel 类常规继承路径与 App Sandbox 不同，如表 2-2

Without App Sandbox	NSOpenPanel : NSSavePanel : NSPanel : NSWindow : NSResponder : NSObject
With App Sandbox	NSOpenPanel : NSSavePanel : NSObject

由于这一常规的区别，App Sandbox 中的 NSOpenPanel 或者 NSSavePanel 对象基本没有继承什么方法。如果你想向 NSOpenPanel 或者 NSSavePanel 对象发送信息，并且该方法被定义在 `NSPanel`, `NSWindow`, 或者 `NSResponder` 类，系统会出现异常。Xcode 编码器**不会**向你就这一常规行为提示错误或警告。

权利和系统资源访问（Entitlements and System Resource Access）

没有被沙盒的程序能访问所有用户能接触到的系统资源——包括内置摄像头和麦克风，网络套接字，绘画和绝大多部分的文件系统。如果恶意代码成功攻击了程序，那这个程序就会成为变身博士大范围的造成损害！

当然，如果程序开启了 App Sandbox 功能，一切就不同了。你只需开放了最少的权限集，并一个一个谨慎地还原它们。Entitlement 是定义某项具体功能的键-值对，如打开外围网络套接字的能力。

其中一个特殊的权利——Enable App Sandboxing——开启 App Sandbox。当你开启沙盒时，Xcde 会在项目导航中生成一个.entitlements property list 文件。

基于目标请求 entitlement。如果程序只有一个目标——主应用程序——只需为该目标请求 entitlement。如果程序需要用到主应用程序和助手（以 XPC 服务的形式），需要单独为每个目标请求 entitlement。详见"XPC and Privilege Separation."

你可以在 Xcode 目标编辑器中对程序的 entitlement 进行精细控制。例如，由于 App Sandbox 不提供程序需要的某项功能，你可能需要请求一个临时的例外权限，如，向 Apple 发送事件的能力。可以通过使用 Xcode 属性清单编辑器来直接编辑目标的.entitlement 属性清单来实现这一目的。

注意：如果你请求一个临时的例外 entitlement，确保你按照 iTunes Connect 网页上关于 entitlement 的指引来操作。

安全范围书签和持久资源访问（Security-Scoped Bookmarks and Persistent Resource Access）

程序对容器外位置上文件系统的访问权——通过用户意图如 Powerbox 授权——在程序或系统重启后不能保留。当程序重启后，你必须从头再来。（例外情况是程序终止时开着的文件，由于 OS X 的恢复功能这个文件还保留在你的沙盒中）。

在 OS Xv10.7.3 中，你可以通过使用 *security-scoped bookmarks* 安全机制来保留对文件系统资源访问权，这样就保留了用户意图。以下是几个能从中受益的程序功能：

- 用户选择的下载、处理或输出文件夹
- 指向用户在任意位置指定的图片的图片浏览库文件
- 储存在其他位置的包括内嵌多媒体的复杂文档格式

两种独特的 security-scoped bookmark

- App-scoped bookmark 能为已沙盒程序提供对用户指定文件和文件夹的持久访问权。
  例如，如果程序采用了一个程序容器外的下载或处理目标文件夹，通过 NSOpenpanel 对话框获得用户想使用该文件夹的初始访问权。
  然后，为其创建一个 app-scoped bookmark，将它作为程序的配置储存（可能用属性列表文件或 NSUserDefaults 类）。有了 app-scoped bookmark，程序就能获得对该文件夹的未来访问权了。
- Document-scoped bookmark 提供了对特定文档的持久访问权。
  例如，代码编辑器能支持指向其他文件的项目文档的概念。这需要对那些文件的持久访问权。其他例子还有图片浏览器或含有图片库的编辑器，其中库文件需要它图片的持久访问权；或者支持持内嵌图片、多媒体或字体的文字处理器。这些情况下，你要配置文档格式（项目文件的、库文件的、文字处理器的，等等）使其能储存 security-scoped bookmark 到文档指向的文件。

通过询问用户意向来获得援引项的初始访问权。然后，为项目创建一个 document-scoped bookmark，并将它作为文档数据的一部分储存它。

Document-scoped bookmark 能被任何拥有书签数据或含有书签文档访问权的程序采用。这种可移植性能使用户，如，向另一个用户发送文件，而接收者仍能使用文档的安全书签。这个文档可以使一个平面文件或者以束形式发送的文档。

Document-scoped bookmark 只能指向文件而不是文件夹。并且这个文件必须不在系统使用的位置上（如`/private` 或`/Library`）。

使用 security-scoped bookmarks

通过以下 5 步使用任一种 security-scoped bookmark:

1. 一旦需要使用 security-scope bookmarks 的每个目标成为配置 Xcode 项目的一部分，在目标里设置恰当的 entitlement。
2. 当用户表明了想要使用程序容器外文件系统资源的意图（如通过 Powerbox），并且你希望持久保留程序对该资源的访问权，创建 security-scoped bookmark。
3. 当程序过后（如，程序重启后）需要访问第 2 步中标记过书签的资源，重构 security-scoped bookmark。此步后会产生一个 security-scoped URL.
4. 在获取了 security-scoped URL 后（或当你想要重获资源访问权时），立刻明确指明你想要使用步骤 3 获得的文件系统资源的 URL。
5. 一旦你知道你不在需要该资源的访问权时，明确指明你要停止它的使用。
   当你撤销了文件系统资源的访问权后，如果想要重新使用资源，你必须返回步骤 4（重申你想要用该资源）。
   如果程序重启了，你必须返回步骤 3（重构 security-scoped bookmark）

上面步骤的第一步——请求 entitlement——是使用任一 security-scoped bookmark 的先决条件。按以下步骤：

- 在目标中使用 app-scoped bookmark，设置 com.apple.security.files.bookmarks.app-scope 权限值为 true
- 在目标中使用 document-scoped bookmarks，设置 com.apple.security.files.bookmarks.document-scope 权限为 true

有了恰当的 entitlement，你可以通过调用 NSURL 类中的 bookmarkDataWithOptions:includingResourceValuesForKeys:relativeToURL:error: 方法（或者使用与它有相同作用的 Core Function——CFURLCreateBookmarkData）来创建 security-scoped bookmark.

当你过后需要访问已标记书签的资源时，可以通过调用 NSURL 类下的 URLByResolvingBookmarkData:options:relativeToURL:bookmarkDataIsStale:error: 方法（或者使用与它有相同作用的 Core Foundation——CFURLCreateByResolvingBookmarkData）来重构它的 security-scoped bookmark。

在已沙盒的应用中，只有当你在 URL 上调用了 startAccessingSecurityScopedResource 方法（或者使用与它有相同作用的 Core Foundation—— CFURLStartAccessingSecurityScopedResource）才能访问 security-scoped URL 指向的文件系统资源。

当你不再需要某一资源的访问权时，你必须调用资源 URL 上的 stopAccessingSecurityScopedResource 方法（或者或者使用与它有相同作用的 Core Foundation—— CFURLStopAccessingSecurityScopedResource）。

开始和终止访问权能根据每个步骤套用。这就是说，如果程序要在 URL 上调用 start 方法两次，那么必须两次调用相应的 stop 方法来完全终止对该资源的访问权。如果在一个无权访问的资源的 URL 上调用 stop 方法，什么都不会发生。

警告 每个 startAccessingSecurityScopedResource 方法的调用都必须有对应的 stopAccessingSecurityScopedResource 方法调用。如果你不在需要文件系统资源的时候没有废除访问权，程序就会泄露内核资源。如果泄露了足够的内核资源，程序会丧失通过 Powerbox 或者 security-scoped bookmarks 添加文件系统位置到沙盒的功能，直至重启。

程序沙盒和代码签名

当你在 Xcode 项目中开启 App Sandbox 并为目标指定其他 entitlement 后，你必须为项目代码签名。注意设置 entitlement 和设置 code signing identity 的区别：

- 以目标为基本单位，使用 Xcode 目标编辑器设置 entitlement
- 将项目作为整体，使用 Xcode 项目创建设置来为其配置 code signing identity

代码签名是必须的，因为 entitlement（包括开启 App Sandbox 的特殊 entitlement）是创建到程序代码签名中的。从另一方面来讲，未签名的程序必然是没有被沙盒的，不管你在 Xcode 目标编辑器中如何设置，它只有默认的 entitlement。

OS X 将程序容器和代码签名捆绑在一起。这一关键的安全特性保证了其他已沙盒的程序不能访问你的容器。这个机制是这样运作的：在系统为程序创建容器后，每次拥有相同束身份（bundle ID）的启动时，系统就会检查它的代码签名和容器内的是否相符。如果系统查出不符，它就会组织程序启动。

OS X 对容器完整性的强制性会影响开发和发布周期。这是因为，在创建和发布程序的过程中，程序会用到各种不同的签名。以下是这个过程的运作方式：

1. 在你创建项目之前，你需要从 apple 获取两项代码签名:开发证书和发布证书。
   开发和测试时，用开发代码签名签署你的应用。
2. 当你的程序在 Mac App Store 上发布时，它会使用 Apple 代码签名。

当测试和调试程序时，你可能想要运行程序的两个版本：你签名的版本和苹果签名的版本。但是 OS X 会将苹果签名的版本视为入侵者并且不允许它启动：因为它的代码签名与程序容器的不符。

如果硬要运行苹果签名的版本的程序，你会收到类似与以下陈述的崩溃报告：

```
Exception Type:  EXC_BAD_INSTRUCTION (SIGILL)
```

解决办法是通过调整程序容器上的访问控制列表（ACL）使其能识别苹果签名版本的程序。具体来说，就是添加苹果签名版本的程序的特定代码请求到容器的 ACL。

- **如何调整 ACL**

1. 打开 Terminal（在/Applications/Utilities）
2. 打开包含苹果签名版程序的 Finder 窗口
3. 在 Terminal 输入下列命令：

   ```
   asctl container acl add -file <path/to/app>
   ```

   用苹果签名版的程序的路径替换 `<path/to/app>`中的 path 占位符。无需一个一个敲进去，只用把程序的 Finder 图标拖到 Terminal 窗口。

现在，容器的 ACL 就包含了程序的两种版本的特定代码请求了。这样，OS X 就允许你运行程序的任一版本了。

你可以运用这项技术在 1）用自己生成的代码签名原始签署版本的程序，如在 “App Sandbox Quick Start,”中创建的；2）用苹果开发代码签名的后期版本；间共享容器。

你可以在容器 ACL 中查看代码请求清单。如，在添加苹果签名版本程序的特定代码请求后，容器 ACL 清单中会列出 2 个获得准许的代码请求。

- **如何显示容器 ACL 中的代码请求清单**

1. 打开 Terminal（在/Application/Utilities）
2. 在 Terminal 中输入下列命令：

   ```
   asctl container acl list -bundle <container name>
   ```

   用程序容器目录的名称替换`<container name>` 中的占字符。（程序容器目录的名称通常和程序束身份的名称相同。）

XPC 和权限分离

有些程序操作使其更可能成为恶意开发的目标，如剖析网络数据、解码视频框架。通过使用 XPC，你能加强程序沙盒的围堵破坏的有效性。App Sandbox 通过将潜在危险活动隔离在他们自己的地址来保护程序。

XPC，是 OS X 的一项进程间通讯技术，它给程序沙盒添加了权限分离功能。**Privilege separation** 是一种开发策略，通过它你能按照所需系统资源访问权的不同将程序分成几份。而这些部分被称为 XPC services.

你创建的 XPC service 在 Xcode 项目中被当做是独立的目标。每个 service 拥有它自己的沙盒——具体来说，它拥有自己的容器和自己的一套 entitlement。另外，只有你的程序能访问 XPC service。这个好处加起来是 XPC 成为执行权限分离的最好的技术。

相比之下，通过 posix_spawn 功能或 `NSTask` 创建的子进程只能简单地继承生成它的进程的沙盒。你不能为其配置 entitlement。所以，子进程不能提供有效的权限分离。

在 App Sandbox 中只用 XPC：

- 按照需要授予每个 XPC service 最小权限
- 安全设计主程序和每个 XPC service 之间的数据传输
- 恰当的构建程序束

XPC service 的生命周期以及其与多线程优化技术（GCD）的整合是由系统完全控制的。获得该技术支持，你只需正确构建程序束。

设计程序沙盒

尽管对于设计或转化 App Sandbox 的程序有一个常见的基础的工作流，但具体的步骤会根据特定的程序而有所不同。为程序沙盒创建工作计划，需要用到这里的纲要，以及对本文档前面章节概念的理解。

采用 App Sandbox 六步走

转化 OS X 程序使其能在沙盒中运行的工作流一般包括以下六步：

1. 明确你的程序是否适合沙盒
2. 设计开发和发布策略
3. 解决 API 不兼容性
4. 申请你需要的 App Sandbox 权限
5. 用 XPC 添加权限分离
6. 实施移民策略

明确程序是否适合沙盒

绝大多数 OS X 程序都能与 App Box 兼容。如果程序中你需要的动作不被 App Sandbox 允许，可以换另一种方式。例如，如果程序依赖硬编码路径到用户主目录的位置，可以试着利用 Cocoa 和 Core Foundation 路径搜索 API。它是使用的沙盒容器。

如果你选择现在不沙盒你的程序，或者你决定使用一个临时例外权限，使用苹果的 bug reporting system 让苹果了解你问题。苹果会在改进 OS X 平台时参考功能请求。

以下程序行为与 App Sandbox 不兼容：

- 使用 Authorization Service
  在 App Sandbox 下，你不能使用 *Authorization Services C Reference* 中提及的功能。
- 在辅助软件中使用可访问性 API
  在 App Sandbox 下，你能够并且应该开启程序的可访问性，详见 *Accessibility Overview*。但是你不能沙盒像屏幕阅读器这样的辅助程序，也不能沙盒一个控制其他应用的程序。
- 发送 Apple 事件给任意程序
  在 App Sandbox 下，你能够接收和回复 Apple 事件，但是你不能将它发送给任意程序。通过使用临时例外权限，你能够发送 Apple 事件给经你指定的程序。
- 发送 broadcast notifications 中的 user-info dictionaries（用户信息辞典？）到其他任务
  在 App Sandbox 下，通过发布 `NSDistributedNotificationCenter` 对象来发送信息给其他任务时，你不能包含用户信息辞典。（你能像往常一样在通过 `NSNotificationCenter` 对象给程序的其他部分发送信息是包含一个用户信息辞典。）
- 加载 kernel extension（内核扩展）

在 App Sandbox 下，不允许加载内核扩展。

- 在 Open 和 Save 对话框模拟用户输入
  如果程序依赖于程序性操控的打开或保存对话框来模拟或更改用户输入，那么该程序不适合沙盒。
- 在其他程序上设置参数选择
  在 App Sandbox 下，每个程序的参数选择都保留在容器内。你的程序不能访问其他程序的参数选择。
- 终止其他程序
  在 App Sandbox 下，你不能使用 `NSRunningApplication` 类来终止其他程序。

设计开发和发布策略

在开发过程中，你可能会遇到要运行程序不同代码签名版本的情况。在你使用一个签名运行程序后，系统将不会允许你启动另一个签名的程序版本，除非你修改程序容器。在设计时请确定你知道如何解决这个问题，参见"App Sandbox and Code Signing,"。
当用户第一次启动程序的被沙盒版本时，系统会为程序创建一个容器。容器的访问控制列表（ACL）也会同时产生，并且与该程序版本的代码签名绑定。这意味着你将来所发布的所有版本必须使用相同的代码签名。

解决 API 不兼容

如果你使用 OS X API 不当，或是暴露用户数据使其易受攻击，你可能会面临 App Sandbox 的不兼容。本部分内容列举了一些不兼容的例子并且提供了解决方案。

打开、保存和跟踪文件

你需要使用 `NSDocument` 类来管理文档，这样你就能利用它内置的 App Sandbox 支持，她也能自动和 Powerbox 配合工作。同时，如果用户使用 Finder 移动了文件，她也能在沙盒中保留文件。
记住当程序被沙盒后，NSOpenPanel 和 NSSavePanel 类的传承路径是不同的。
如果你不使用 NSDocument 来管理程序文件，你也可以通过 `NSFileCoordinator` 类和 `NSFilePresenter` 协议定制你自己的文件系统支持。

保留文件系统资源的访问权

如果你的程序依赖于对程序容器外文件系统资源的持久访问权，你需要采用 security-scoped 书签。

为程序创建登陆项

使用 SMLoginItemSetEnabled 功能（在 ServiceManagement/SMLoginItem.h）为已沙盒的程序创建登陆项。
（在 App Sandbox 下，你不能用 LSSharedFileList.h 头文件中的功能创建登陆项。如，你不能使用 LSSharedFileListInsertItemURL.也不能通过 LSRegisterURL 功能控制登陆服务的状态。）

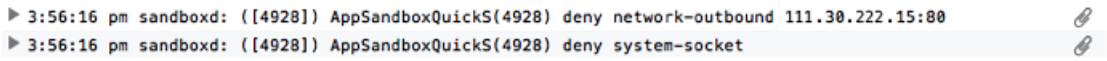
访问用户数据

POSIX 层上的 OS X 路径搜索 API 返回路径到相关的容器而非相关的用户主目录。如果程序在被沙盒之前访问的是用户主目录下的位置且你使用的是 Cocoa 或者 Core Foundation API，那么，在你开启沙盒后，路径搜索代码会自动使用程序容器。
首次启动已沙盒的程序时，OS X 会自动移植程序的主要的参数选择文件。如果程序要使用额外的支持文件，对这些文件进行一次性移植到容器中。
如果你使用的类似于 `getpwuid` 的 POSIX 命令来获取到用户主目录的路径，那么请使用 Cocoa 或者 Core Foundation 符号如 `NSHomeDirectory` 功能。这样，你就能支持 App Sandbox 对直接访问用户主目录的限制。
如果你的程序需要访问用户主目录来运转，通过 buy reporting system 让苹果知道你的需求。
访问其他程序的参数选择
因为 App Sandbox 指引路径搜索 API 到程序的容器，所以读取或写入用户参数只能发生在容器内，不能访问其他已沙盒程序的参数选择。但是，你的已沙盒的程序能访问没有被沙盒的程序的参数选择，它们位于`~/Library/Preferences` 目录。
如果你程序需要访问其他程序的参数选择才能运转——如需要访问用户为 iTunes 定义的播放列表——通过 bug reporting system 让苹果知道你的需求。
记住这些限制条件，你可以通过基于路径临时例外权限来获得用户 `~/Library/Preferences` 文件夹的程序访问权。记住使用只读权限来防止用户参数选择受到恶意侵犯。一个 POSIX 功能如 `getpwuid` 可以提供你需要的文件系统的路径。

申请你需要的 App Sandbox 权限

在 Xcode 项目中为目标采用 App Sandbox，通过在目标编辑器中选择 Enable App Sandbox 复选框来定义该目标的 `com.apple.security.app-sandbox entitlement` 键值为 true.
重要 当你请求最小的权限时，App Sandbox 能最有效的保护用户数据。注意不要请求程序不需要的权限。多想想如何改变程序来减少对权限的需求。
以下是检验该权限是否是你需要的基本工作流：

1. 运行程序，使用其功能
2. 在 Console（`/Applications/Utilities/`中），在 All Messages 系统日志查询中查找 sandboxd 违例。
   每个这样的违例说明了程序有沙盒不允许的操作。
   以下是 Console 中 sandboxd 违例：



点击违例信息右侧的回形针图标来查看导致该违例的原因。

3. 根据查找到的 sandboxd 违例决定如何解决问题。一些情况下，程序的一点简单的改动，如使用容器位置而非其他文件系统的位置，能解决问题。另一些情况，使用 Xcode 目标管理器申请 App Sandbox 权限是最佳选择。
4. 使用 Xcode 目标编辑器，开启你认为能解决问题的权限。
5. 运行程序，再次使用其功能。
   要么确保你已经解决了问题，要么进一步调查。

只用 XPC 添加权限分离

在 App Sandbox 下开发时，从权限和访问权的角度查看程序的行为。考虑将高风险操作隔离在他们自身 XPC 服务的安全性和稳健性的潜在好处。
如果你决定这一功能应该被安置在 XPC 服务中，参考 *Daemons and Services Programming Guide* 中的 "Creating XPC Services" 来操作。

执行移植策略

确保刚使用了沙盒前程序版本的用户能顺利的更新至沙盒后的版本。具体操作详见 "Migrating an App to a Sandbox."。

移植程序到沙盒

程序已沙盒版本不能访问相同程序未沙盒版本的支持文件的保存地址。例如，支持文件一般存储在这里：

Path	Description
`~/Library/Application Support/<app_name>/`	Legacy location
`~/Library/Containers/<bundle_id>/Data/Library/Application Support/<app_name>/`	Sandbox location

如图所示，Application Support 目录的沙盒储存位置是在容器内的，这样已沙盒的程序就能拥有这些文件的无限制的读取/写入访问权。如果你前期发布了一个未沙盒的程序，现在想发布一个已沙盒版本，你必须将支持文件移到新的沙盒能访问的位置。
注意：系统会在首次启动已沙盒程序的时候自动移植程序的参数选择文件（`~/Library/Preferences/com.yourCompany.YourApp.plist`）。
当某一用户首次启动程序已沙盒版本的时候，OS X 会以每个用户为基础提供支持文件移植。这项支持功能依赖于你创建的一个叫做 *container migration manifest* 的特殊属性列表文件。**Container migration manifest** 包含了一系列定义了你想要移植的支持文件和目录的字符串。文件的名称必须是 container-migration.plist。对于每个你想要移植的文件或目录，你可以选择让系统安置项目到容器中，也可以明确指定目标位置。
OS X 会移动——而不是复制——你在容器移植清单中的文件和目录。这就是说移到程序容

器中的文件和目录不在存在于原来的位置。另外，容器移植是单行过程，如果你在开发或测试阶段需要撤销操作，参看 “Undoing a Migration for Testing” 。

创建容器移植清单

- **如何 code 项目创建和添加容器移植清单**
  1. 添加属性列表文件到 Xcode 项目
     Property List 模板在 OS X “Resource” 组下的文件模板对话框中。
     重要 确保 `container-migration.plist` 拼写正确并且全部小写。
  2. 添加 Move 属性到容器移植清单
     Move 属性是在容器移植清单中的独立的顶层键（top-level key）。按以下方法将其添加到空白文件：
     - 右键点击空白的编辑器创建新的.plist 文件，然后选择 Add Row.
     - 在 Key 栏，输入 Move 作为键的名称
       必须拼写及大小写正确。
     - 在 Type 栏，选择 Array
  3. 为你想移植的第一个文件或文件夹添加字符串到 Move 列。
     例如，假设你想移植 Application Support 目录（包括里面的文件和子目录）到你的容器中。如果目录的名称是 App Sandbox Quick Start 而它是包含在 `~/Library/Application Support` 目录中的，使用以下字符串作为新属性列表项的值：

```
${ApplicationSupport}/App Sandbox Quick Start
```

此处无须尾斜杠，可以空格。路径搜索常量相当于`~/Library/Application Support`。相似地，添加其他字符串来指明其他你想要移植的文件或文件夹的原始（沙盒前）路径。
当你指定移动某一目录时，记住该移动是递归的——它包括了该目录下所有的子目录和文件。
首次测试移植清单前， “Undoing a Migration for Testing.”介绍了撤销移植的方法。

- **如何测试容器移植清单**
  1. 在 Finder 中，打开以下两个窗口：
     - 在一个窗口查看 `~/Library/Containers/`目录中的内容
     - 在另一个窗口查看你想要移植的文件
  2. 创建并且运行 Xcode 项目

一旦成功移植，支持文件会从原始（未沙盒）目录消失而出现在程序的容器中。
如果你想在移植过程中改变一下支持文件的保存位置，可以使用稍复杂点的.plist 结构。具体来说，为你想要控制移植位置的文件或目录提供一个开始和结束路径。结束路径和容器中的 Data 目录相关，你可以使用任何 “Use Variables to Specify Support-File Directories.”中介绍的搜索路径常量。
如果的你储存路径是自定义的（不是标准容器的一部分），系统会在移植过程中创建目录。

- **如何控制移植文件或目录的储存位置**
  1. 在容器移植清单中，添加新项目到 More 栏
  2. 在 Type 栏，选择 Array
  3. 添加两条字符串作为新 array 项目的子项目
  4. 在第一条字符串中指明移植文件或目录的原始路径
  5. 在第二条字符创中指明移植文件或目录的自定义的目标（沙盒）路径。

文件移植会从容器移植清单第一个项目到最后一个项目一次进行。注意它的优先级。例如，

如果你想移动除了一个文件外的整个 Application Support 目录，并且你想将该文件移植到容器中与 Application Support 平行的新目录中。
这种情况下，你必须在指定移动 Application Support 目录之前指定单个文件的移动。也就是说，指定单个文件的移动在容器移植清单中优先级更高。（如果 Application Support 被先指定移动了，那么当移植程序试图移植单个文件到容器内的定制位置时，它已经不在其原始位置了。

撤销测试移植

当测试支持文件的移植时，你可能需要不止一次进行移植。为了能支持这个，你需要一种方式来恢复你开始目录结果——也就是说，移植前它存在的结构。
一种方式是，在首次移植前，复制移植目录。将该复件保存在不会受移植影响的位置。以下内容是建立在你已经做了这种备份的基础上的。

- 如何手动撤销测试用容器移植
  1. 手动从备份复件处复制文件和目录——在清单中指明的——原始（移植前）位置
  2. 删除程序容器

  你下次启动程序时，系统会重建容器并且按照容器移植清单的当前版本移植支持文件。

容器移植清单示例

列表 4-1 容器移植清单的一个示例

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>Move</key>
   <array>
      <string>${Library}/MyApp/MyConfiguration.plist</string>
      <array>
         <string>${Library}/MyApp/MyDataStore.xml</string>
         <string>${ApplicationSupport}/MyApp/MyDataStore.xml</string>
     </array>
  </array>
</dict>
</plist>
```

该清单指定了用户 Library 目录中的两个项目到程序容器的移动。第一个项目，MyConfiguration.plist，仅指定了原始路径，这样移植程序会自行妥当安置它。
第二个项目，MyDataStore.xml，指定了原始路径和自定义的储存位置。
你可以根据自己的需要改变路径中的`${Library}`和`${ApplicationSupport}`部分。

使用变量来指定支持文件目录

在容器移植清单上指定路径的时候，你可以用某些与常用支持文件目录相对应的的变量。这些变量在原始和目标路径中有效，但是变量解析到的路径根据环境而改变。参考表 4-1

环境	变量解析到
原始路径	相关主路径（与 ~ directory 相关的）
目标路径	相关容器路径（与容器中的 Data 目录相关的）

可以用来指定支持文件目录的变量参见表 4-2。如何使用这些变量，见表 4-1。

你也可以用 `${BundleId}` 来解析到程序 bundle identifier，这使你能方便地将其编入一个原始或目标路径中。

表 4-2 支持文件目录的变量

变量	目录
`${ApplicationSupport}`	包含程序支持文件的目录。相当于 `NSApplicationSupportDirectory` 搜索路径常量。
`${AutosavedInformation}`	包含用户自动保存文件的目录。相当于 `NSAutosavedInformationDirectory` 搜索路径常量。
`${Caches}`	包含垃圾缓存文件的目录。相当于 NSCachesDirectory 搜索路径常量。
`${Document}` `${Documents}`	每个变量相当于包含用户文件的目录。相当于 `NSDocumentDirectory` 搜错路径常量。
`${Home}`	当前用户的主目录。相当于通过 `NSHomeDirectory` 功能返回的目录。当在清单中的目标路径时，解析到 Container 目录。
`${Library}`	包含程序相关的支持和配置文件的目录。相当于 `NSLibraryDirectory` 搜索路径常量