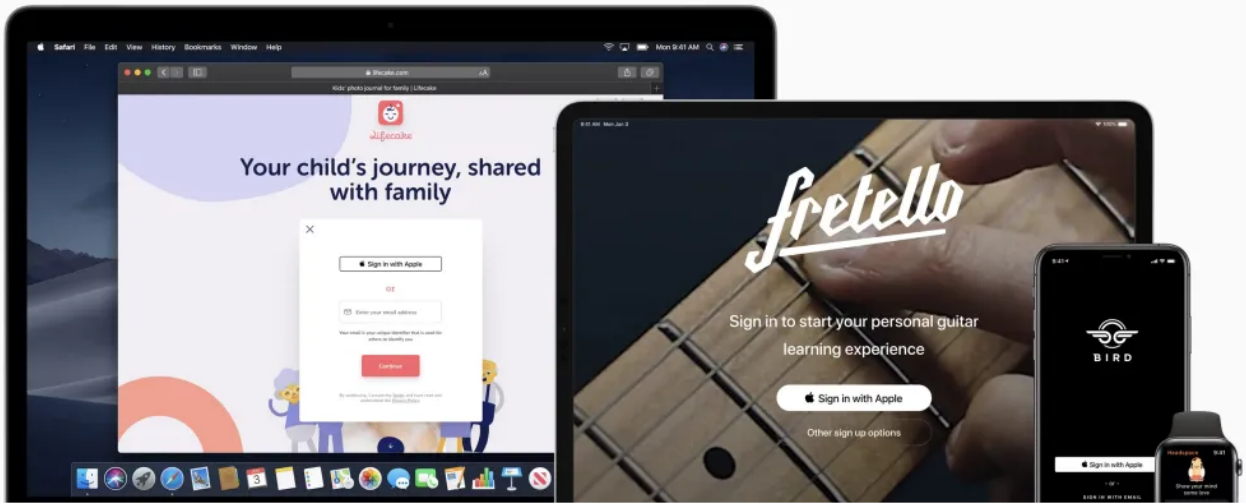
Sign in with Apple 新变化：强制与安全

原创 ZUBIN 老司机技术周报 今天

收录于话题

#iOS 74 #iOS Weekly 28

> 作者：ZUBIN，iOS 开发者，目前就职于阿里
>
> Session: https://developer.apple.com/videos/play/wwdc2020/10173/



Sign in with Apple [1] （“通过 Apple 登录”）在 WWDC 2019 随着 iOS 13 和 macOS 10.15 以及 Xcode 11 一起推出，去年，我也分享了一篇文章，详细介绍了app 如何接入 Sign in with Apple 能力：

Sign in with Apple 让用户能用自己的 Apple ID 轻松登录开发者的 app 和网站。用户不必填写表单、验证电子邮件地址和选择新密码，就可以使用“通过 Apple 登录”设置帐户并立即开始使用 app。所有帐户都通过双重认证受到保护，具有极高的安全性：

尊重隐私。

“通过 Apple 登录”的设计初衷就是为了让用户对自己的隐私感到放心。数据收集仅限于用户的姓名和电子邮件地址，而且即使用户更希望对自己的邮件地址保密，Apple 的私密电子邮件中继转发服务也能让他们收到相关电子邮件。Apple 不会跟踪用户与您 app 的交互情况。

内置安全性。

使用“通过 Apple 登录”的每个帐户均默认受到双重认证保护。在 Apple 设备上，用户会保持登录状态，并可以随时使用面容 ID 或触控 ID 重新进行身份验证。

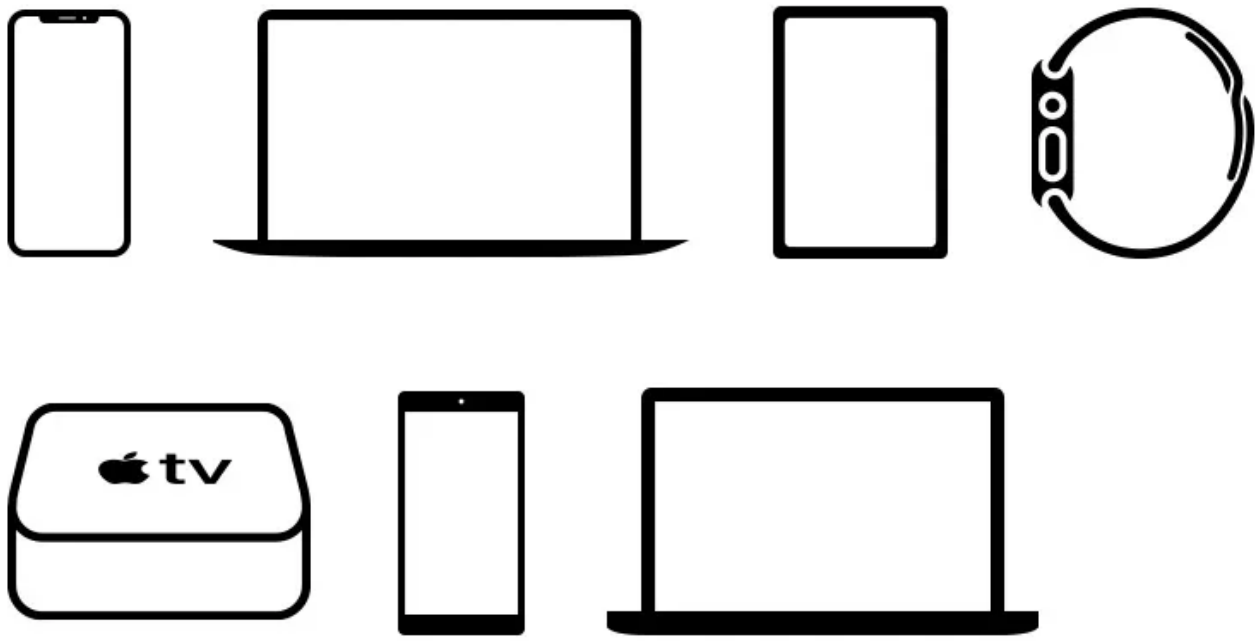
处处可用。

iOS、macOS、Apple tvOS 和 watchOS 都原生支持“通过 Apple 登录”。此外，该功能可在任何浏览器中使用，这意味着您可以将它部署在您的网站上以及在其他平台上运行的 app 版本中。

反欺诈。

“通过 Apple 登录”让您可以对新用户放心。它使用设备端机器学习功能和其他信息来提供一种能保障隐私的新信号，帮助您判断新用户是真人还是您可能需要再次检查的帐户。

其中，最重要的是 Apple 提供了 **Sign in with Apple JS SDK** [2]，使得它可以跨平台使用：



此外，苹果也在去年更新了《**App Store** 审核指南》[3]，加入了“4.8 通过 Apple 登录”一条，要求所有使用第三方或社交登录服务的 **app**，都必须同时接入 **Sign in with Apple**

作为同等选项。 目前，国内的大部分 app 也基本都集成了这个能力。

本文将带你先简单回顾一下 WWDC 2019 中介绍的如何快速集成“苹果登录”能力，然后解读 WWDC 2020 中 Sign in with Apple 的新增 API 和相关新特性：

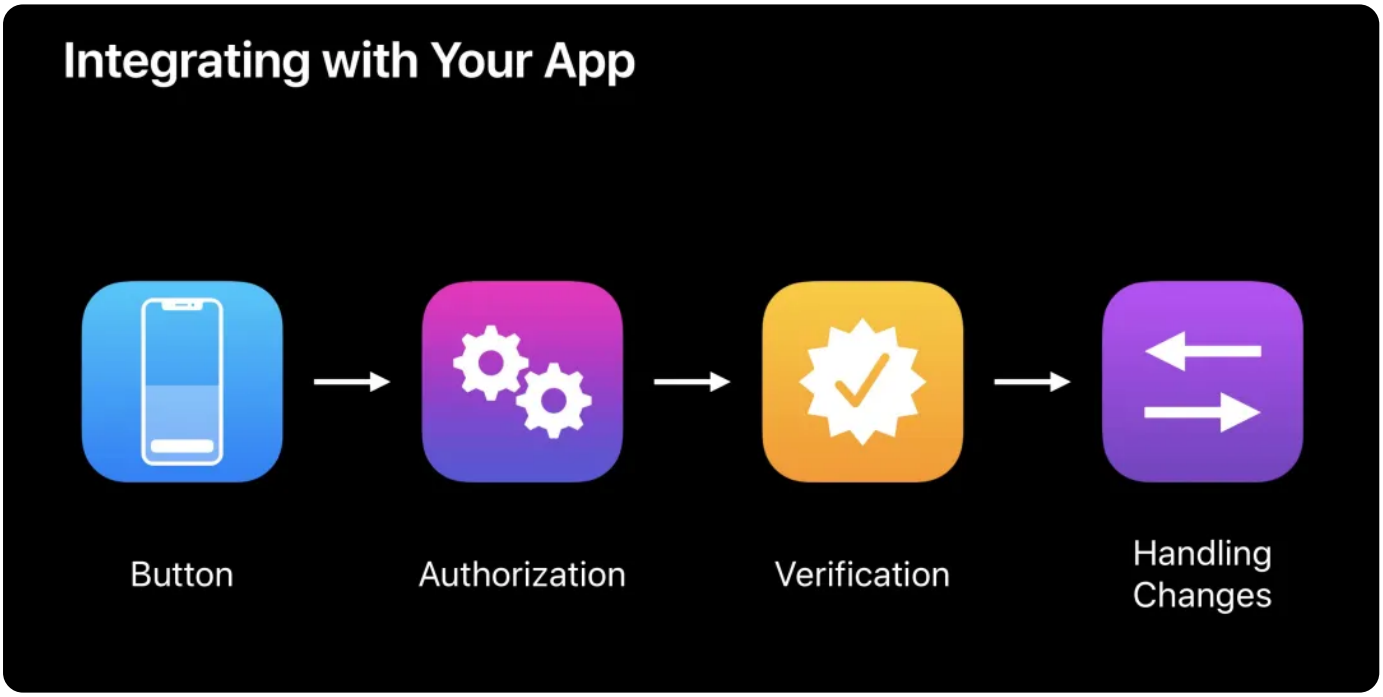
- Creating a secure request（更安全的授权请求）
- Credential state changes（处理授权凭证状态变化）
- Server to server notifications（服务端通知）
- Sign in with Apple Button（支持 **SwiftUI**）

- Upgrading to Sign in with Apple（在现有的账号体系中快速集成苹果登录能力）

0. 集成 Sign in with Apple

在 app 中集成 “Sign in with Apple” 能力，大致需要以下 4 步骤：

（1）添加苹果登录按钮 （2）点击发起授权请求 （3）处理回调数据，并在服务端验证结果 （4）处理苹果账号会话发生变化



详细的集成说明和示例代码，请查看去年 WWDC19 内参的文章，这里不再赘述：

- WWDC 2019：Sign In with Apple - 使用苹果账号登录你的应用

补充：上述文章在集成苹果登录需要做哪些配置，以及在客户端拿到 authorizationCode 和 identityToken 后传给服务端，服务端如何调苹果提供的 **REST API** [4] 进行验证，没有比较详细的说明，可以参考如下两篇文章：

- 快速配置 Sign In with Apple

- **Sign In With Apple** 从登录到服务器验证 [5]

下面我们介绍一下 WWDC 2020 中 Sign in with Apple 的新内容。

1. 更安全的授权请求

发起授权

当我们点击 “Sign in with Apple” 按钮发起授权登录请求时，iOS 系统自带的 Apple ID 双重因子身份验证使得我们 app 的账户已经具备很好的安全能力，但我们仍然可以做一些事情，使得授权请求更加安全。

一般情况下，app 发起授权登录请求的代码（Swift）大致如下：

```
// Configure request, setup delegates and p
@objc func handleAuthorizationButtonPress()
    let request = ASAuthorizationAppleIDPro
    request.requestedScopes = [.fullName, .

    request.nonce = myNonceString()
```

```
    request.state = myStateString()

    let controller = ASAuthorizationControl

    controller.delegate = self
    controller.presentationContextProvider

    controller.performRequests()
}
```

这里有两个参数 `nonce` 和 `state`，可以用于验证执行请求后获得的授权凭证是否符合预期。

- **nonce**：发起授权请时，开发者设置给 request 的不透明数据块（opaque blob of data），它是一串字符串，用于唯一标识一次授权请求，每次创建新请求时都应设置不一样的值。该值会作为请求响应（ASAuthorization response）中 identityToken 的一个属性直接返回。因此

服务端在验证 identityToken 时，可以同时验证此值，检查该 nonce 之前是否已经使用过，有助于防止重放攻击（**replay attacks**）。

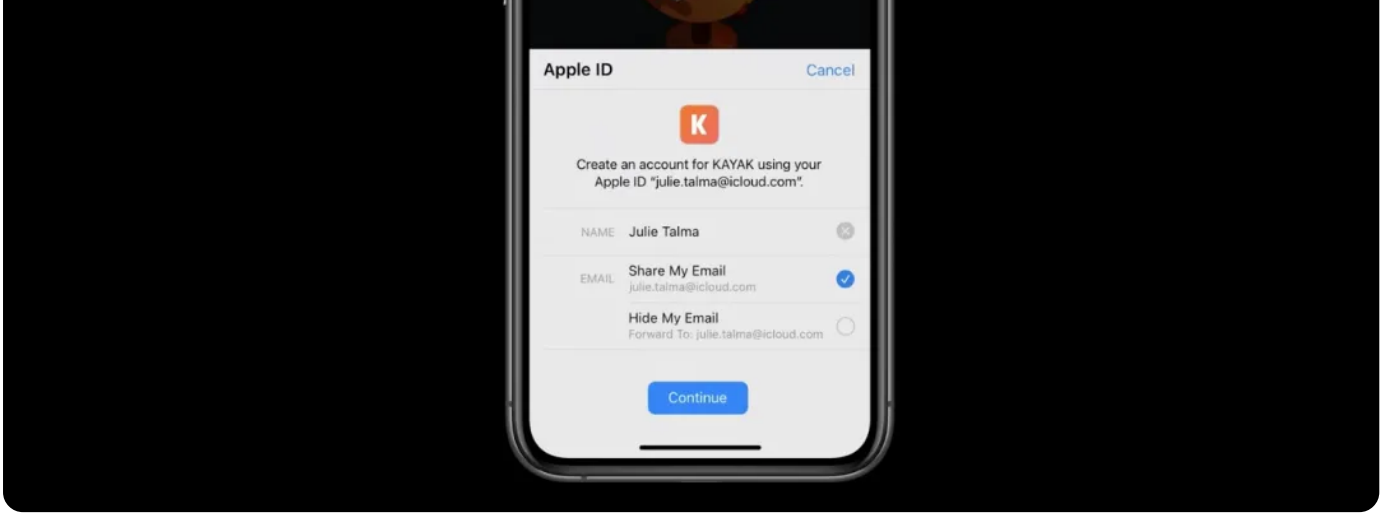
- **state**：与 nonce 类似，由开发者手动设置给 request，也是一串字符串；它会直接在响应 response 中返回，使得开发者可以在本地将授权凭证结果与请求匹配，判断当前响应是否由本设备中开发者自己的 app 发起的。

> 注1：Replay Attacks，重放攻击，又称重播攻击、回放攻击，是指攻击者发送一个目的主机已接收过的包，来达到欺骗系统的目的，主要用于身份认证过程，破坏认证的正确性。

> 注2：identityToken 是一个 JWT 格式的加密数据，JWT 相关知识介绍详见：**JSON Web Token** [6]

简单地说，nonce 和 state 都是开发者在发起授权时设置给 request 对象，在授权回调中，Apple 都会原封不动地返回，nonce 会拼接在 identityToken 中，主要在服务端验证，用于防止重放攻击；而 state 则是直接在授权响应结果中返回，用于客户端本地验证请求是否有当前 app 发起的。

设置完这两个参数，我们就可以发起授权登录请求了，当 request 的 scopes 中我们设置了 `fullName` 和 `email`，用户会看的如下页面：



此处，用户可以选择“共享电子邮件”或者“隐藏邮件地址”。当用户选择“隐藏邮件地址”时，开发者会拿到一个中转的专用邮件地址（eg: `pfju4f59kj@privaterelay.appleid.com`），开发者发送到该 email 的邮件会被自动转发给用户真实的邮箱上，用户也可以直接回复邮件给开发者。

因此，开发者可以直接把这个中转的邮件地址当成真实的 email 来使用，且对于同一个开发者账号下的所有 app，同一个用户 Apple ID 得到的 private relay email 地址是一样的。

验证结果

如上所述，当授权成功后，我们可以在 delegate 方法中得到回调结果，代码如下：

```
// ASAuthorizationControllerDelegate
func authorizationController(controller: AS
    if let credential = authorization.crede
        let userIdentifier = credential.use
        let fullName = credential.fullName
        let email = credential.email
        let realUserStatus = credential.rea

        let state = credential.state
        let identityToken = credential.iden
        let authorizationCode = credential.

        // Securely store the userIdentifie
        self.saveUserIdentifier(userIdentif
```

```
            // Create a session with your serve
            self.createSession(identityToken: i
        }
    }
```

在回调结果中，我们可以取到用户的 email、fullName、realUserStatus、userIdentifier 等用户信息，同时也可以拿到 state、authorizationCode、identityToken（包含 nonce 字段），通过这些字段，我们可以安全地验证请求的合法性，并与服务端创建会话信息（session）。

此外，需要注意的一点是，由于 fullName、email 仅会在第一次授权 app 时才会包含在结果凭据中，后续的重新授权都将不会再包含这俩信息（即使请求 scopes 里设置了 `.fullName` 和 `.email`，除非用户在设置了删除了对该 app 的授权后，再次触发才会重新返回这俩字

段），因此，我们的 app 如果需要用到这两个字段，应该在第一次授权请求结果中把它们缓存起来，这样就不会丢失所需的重要信息。

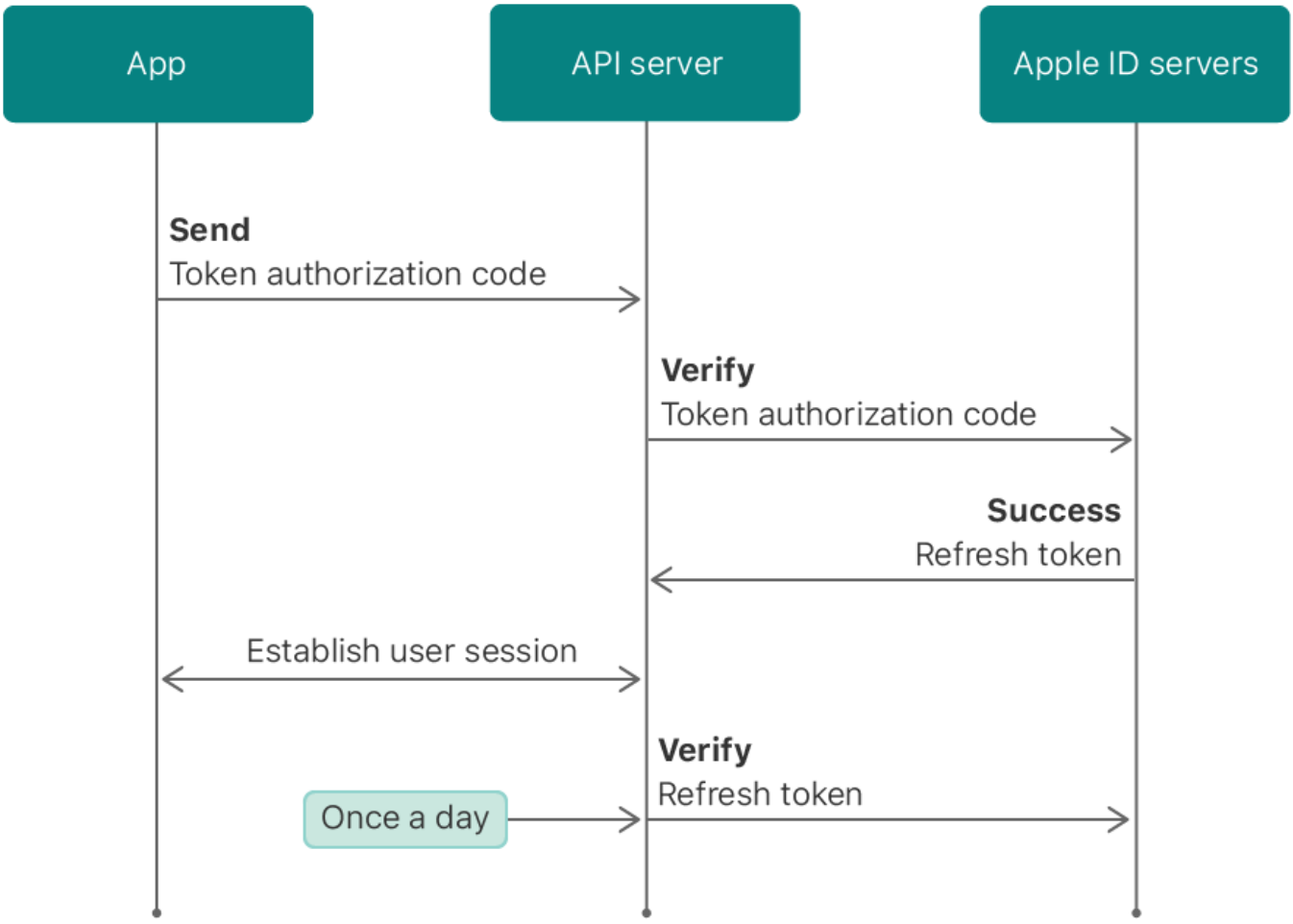
如前所述，响应结果中拿到 state 可用于客户端本地验证该结果是否属于当前的请求。而另外两个参数 authorizationCode、identityToken 需要传给服务端进行验证和解码（其实就是简单的 Base64 解码），解码后的结果如下：


```
{
  "iss": "https://appleid.apple.com",
  "aud": "<Bundle Identifier>",
  "exp": 1591157663,
  "iat": 1591157063,
  "sub": "001888.df37b9a8a63a423a8d80a5fc2069e308.1756",
  "nonce": "a3315daf11b",
  "c_hash": "glOEjO5pyJV9JdNzc2nhqQ",
  "email": "4r8ihiunz7@privaterelay.appleid.com",
  "email_verified": true,
  "is_private_email": true,
  "auth_time": 1591157063,
  "nonce_supported": true,
  "real_user_status": 1
}
```

在解码后的 JSON 中，我们可以看到其与客户端响应结果中获取到基本一致。其中 sub 字段即为用户的唯一标识符（userIdentifier），nonce 字段可以用于校验与请求前设置的是否相同，防止重放攻击。

前面提到，identityToken 是一个 JWT 格式的数据，虽然服务端可以直接解码得到 JSON 数据，但我们仍需要通过 authorizationCode 调 **REST API** [7] 向苹果服务器换取 accessToken 和 refreshToken 以及一个新的 JWT 格式的身份令牌，该令牌的内容与 identityToken 是一致的，到此即完成服务端数据的验证。后续开发者服务端可根据 refreshToken 每天一次向苹果服务端验证当前 Apple 用户的最新状态。

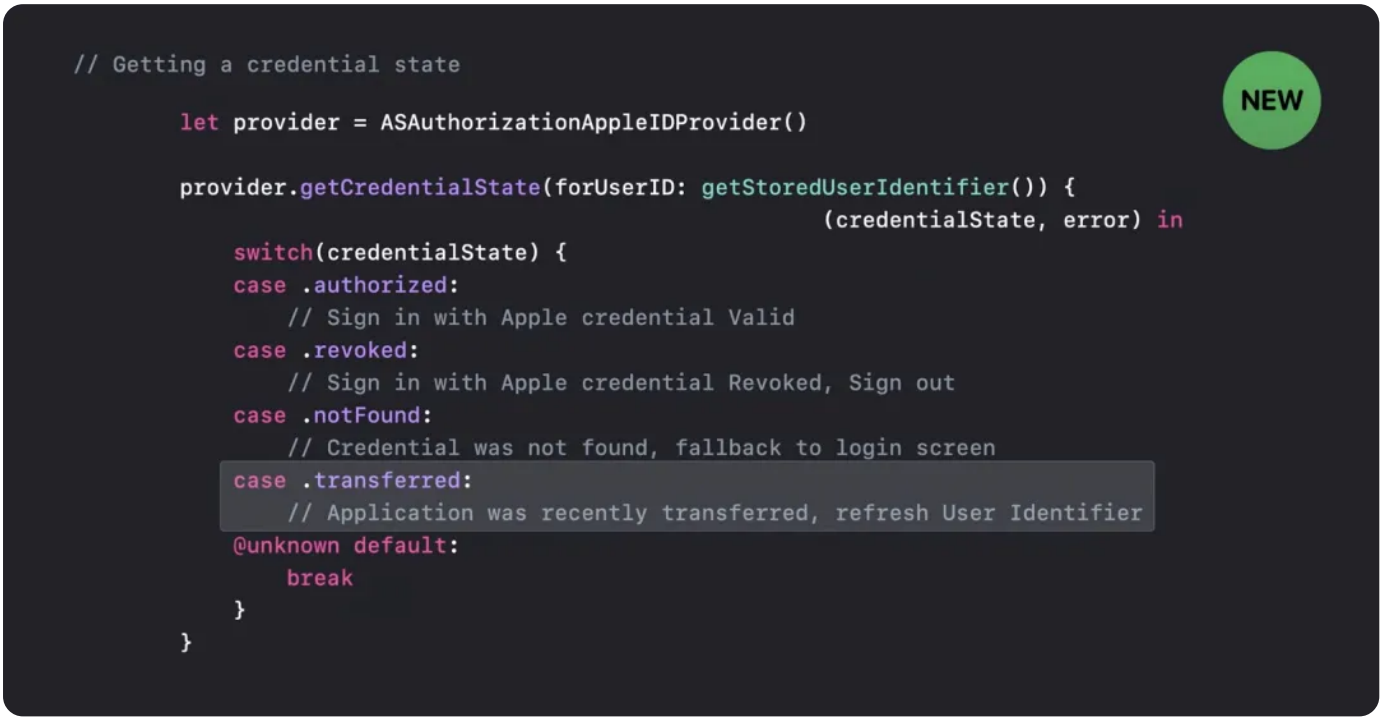
完整的流程如下图所示，具体的细节请参考文档：**Verifying a User** [8]



2. 处理授权凭证状态变化

上面介绍了如何发起一个安全的授权请求并验证回调结果，接下来我们可以通过 `getCredentialState` API 验证当前用户

（userIdentifier）是否仍然登录到此设备和app。如下图所示，我们应该在每次应用冷启动或者从后台回前台时调该接口，并根据结果做相应处理。



该接口返回结果的枚举值如下：

- authorized：登录状态有效；
- revoked：上次使用苹果账号登录的凭据已被移除，需退出解除绑定并重新引导使用苹果登录；

- notFound：未登录，直接显示开发者 app 的登录页面；

此外，本次新增了一个枚举值 `transferred`，该值表示当前 app 的所有者发生了转移（ownership changed），从一个开发团队（账号）转移给另一个团队，例如一家公司被收购之后。

如前所述，用户标识符（userIdentifier）对于同一个开发者账号下的 app 是唯一的。因此，当转移 app 的所有权时，需要将现有用户迁移到新的用户标识符以匹配新的团队账号。

此迁移是静默处理的，不需要任何用户交互。当收到 `transferred` 枚举值时，与创建新帐户或登录现有帐户时发起授权请求的代码相同，需要额外将当前存储的用户标识符添加到请求参数中，我们就可以验证用户的状态，并

生成与新团队匹配的新用户标识符，代码如下，其响应结果也与之前完全一致，我们可以在回调更新用户的 userIdentifier，而用户完全无感。

```
// Migrating a user identifier

        let request = ASAuthorizationAppleIDProvider().createRequest()
        request.requestedScopes = [.fullName, .email]

        request.user = getStoredUserIdentifier()

        request.nonce = myNonceString()
        request.state = myStateString()

        let controller = ASAuthorizationController(authorizationRequests: [request])

        controller.delegate = self
        controller.presentationContextProvider = self

        controller.performRequests()
```

3. 订阅服务端通知

Server to server notifications 是今年新推出一个特性。通过监听这些通知，可以让我们直接从服务端监视授权凭证状态更改等事件，并接收其他类型的事件。

首先，我们需要在苹果开发者网站上注册一个服务器端点（server endpoint），完成此注册后，就可以开始接收用户 Apple ID 状态发生变化事件了。

> 注：至于如何操作注册服务端通知，Session 里并没有细讲，也没有相关文档，可能目前苹果开发者网站还没更新，后续应该会补充吧，Apple Developer Forums 上也有人在问这个问题：https://developer.apple.com/forums/tags/wwdc20-10173

事件将通过由苹果签名过的 JSON Web Token 格式来传送，内容如下：

```
{
    "iss": "https://appleid.apple.com/",
    "aud": "<Bundle Identifier>",
    "iat": 1508184845,
    "jti": "<unique events stream id>",
    "events": [
        {
            "type": "email-enabled",
            "sub": "<user_id>",
            "email": "<email@privaterelay.appleid.com>",
            "is_private_email": true
            "event_time": 1508184845
```

```
        }
    ]
}
```

JSON 的内容包含了一些重要的信息，包括 app 的颁发者（issuer）和 BundleId 等，以及 event（事件）的具体内容。事件的类型有以下几种：

- **email-disabled**：当用户决定停止从 private relay email 中接收邮件时，开发者会收到这个事件通知。
- **email-enabled**：表示用户选择重新接收电子邮件。需要注意的是，只有当用户在授权时，选择“隐藏邮件地址”，使用一个专用的中转邮箱作为账户时，才会发送这两个事件。
- **consent-revoked**：当用户决定停止在开发者的 app 中使用其 Apple ID 时，将向开发者发送“同意撤销”事件，此时 app 应将其视为用户已退出登录（sign out）。用户在 iOS 系统设置 -> Apple ID -> 密

码与安全性 -> 使用 Apple ID 的 App，可以选择对某个 App 停止授权使用 Apple ID。

- **account-delete**：当用户要求苹果删除其 Apple ID 时，将发送此事件。当收到此通知时，与用户关联的用户标识符将不再有效。

如你所见，通过监听这些通知，我们将能够以更好的方式直接从服务端对这四种不同的情况作出处理，以提升用户体验。

4. 苹果登录按钮支持 SwiftUI

今年，Sign in with Apple 的另一个新特性就是支持 SwiftUI。使用 SwiftUI，在 app 中展现一个 “通过 Apple 登录” 按钮，以及发起请求和处理响应结果都将变得非常简单，示例代码如下：

```swift
// SwiftUI example:
SignInWithAppleButton(.signIn) {
    onRequest: { (request) in
        // 发起请求
        request.requestedScopes = [.fullNam
        request.nonce = myNonceString()
        request.state = myStateString()
    }
    onCompletion: { (result) in
        // 处理响应结果
        switch result {
        case .success(let authorization):
            // Handle Authorization
        case .failure(let error)
            // Handle Failure
        }
    }
}.signInWithAppleButtonStyle(.black) // 设置
```

在上述代码中，SignInWithAppleButton(.signIn) 可以设置不同的提示文案，枚举值有 .s

ignIn 、 .signup 、 continue ；在 onReque st 闭包中，我们可以给 request 设置 scopes、nonce、state 等参数；在 onCompl etion 闭包中，可以处理授权回调结果，包括成功和失败。最后，可以通过 .signInWithAp pleButtonStyle(.black) 设置按钮的样式，目前有 .black 、 white 、 whiteOutline 三种。

此外，我们可以在苹果提供的在线编辑地址自定义按钮的样式：

- https://appleid.apple.com/signinwitha pple/button

5. 从现有的“账密登录”升级到“苹果登录”最佳实践

上面主要介绍的是如何通过 “Sign in with Apple” 能力登录 app 或者创建一个新的账号，但对于绝大多数 app 来说都有自己的账号体系，且用户在升级新版本前也基本已经登录当前 app 了。

对于这种情况，国内大部分 app 的解决方案是：

- 当用户未登录时，点击 “Sign in with Apple” 进行授权登录，根据返回的 userIdentifier 用户标识符判断当前 Apple ID 是否有绑定过 app 的账号，如果绑定过，则服务端安全验证通过后创建对应的登录态；如果未绑定过，则引导用户将 Apple ID 绑定到现有账号上或者创建一个新的账号；

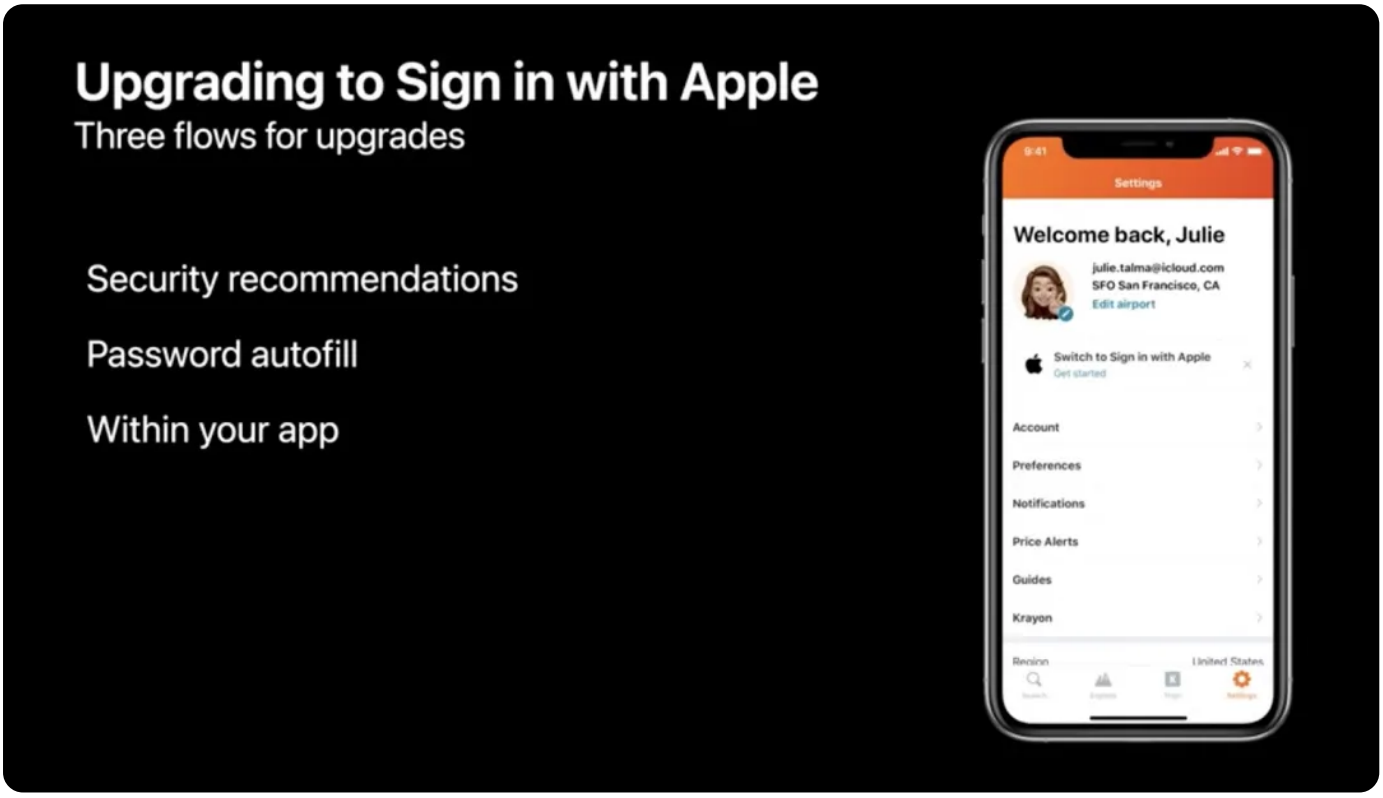
- 当用户已登录时，则一般在 app “设置” 页面的 “账号绑定管理” 提供一个 “绑定 Apple 账号” 入口，让用户可选将当前登录 app 的账号与 Apple ID 进行绑定或者解绑。

对于那些通过传统的账号和密码登录一个 app 的用户来说，他们并不想放弃原有的账号，也不想通过 Sign in with Apple 重新创建一个新的账号。今年，苹果也提供了新的 API 用于解决这个问题，这个新的 API 是一个 extension，它集成了 “通过 Apple 登录” 和 “升级密码强度” 能力。

- **Account Authentication Modification Extension**

该 extension 主要有以下 3 个调用场景：

（1）安全推荐识别出用户的授权凭证弱（weak credential），需要升级；（2）当用户与 app 交互时使用密码自动填充，所选择的是一个弱凭证（weak credential）；（3）开发者在 app 中引导用户升级交互调用新的身份验证服务 API；



其主要包含以下两个类，view controller 用于展示一个引导升级到 Sign in with Apple 的页

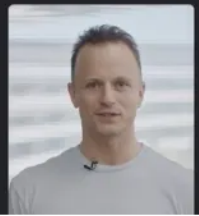
面，开发者也可以自定义其 UI，而 extention context 则用于上下文信息交互和控制流程。

- ASAccountAuthenticationModification ViewController

```
// ASAccountAuthenticationModificationViewController

import AuthenticationServices

class AccountModificationViewController: ASAccountAuthenticationModificationViewController {

   // convert without UI
    override func convertAccountToSignInWithAppleWithoutUserInteraction(for
serviceIdentifier: ASCredentialServiceIdentifier, existingCredential: ASPasswordCredential)
{...}

    override func viewDidLoad() {...}

    // convert with UI
    override func prepareInterfaceToConvertAccountToSignInWithApple(for serviceIdentifier:
ASCredentialServiceIdentifier, existingCredential: ASPasswordCredential) {...}

}
```

- ASAccountAuthenticationModification ExtentionContext

```
// ASAccountAuthenticationModificationExtensionContext
open class ASAccountAuthenticationModificationExtensionContext : NSExtensionContext {

    open func getSignInWithAppleAuthorizationWithState(state: NSString?, nonce: NSString?,
       completionHandler: @escaping (ASAuthorizationAppleIDCredential?, Error?) -> Void)

    open func completeUpgradeToSignInWithApple(userInfo: [AnyHashable : Any]? = nil)

    open func cancelRequest(withError error: Error)
}

//  ASExtensionErrors.h
public struct ASExtensionError {
    …
    public static var userInteractionRequired: ASExtensionError.Code { get }

    public static var userCanceled: ASExtensionError.Code { get }

    public static var failed: ASExtensionError.Code { get }
```



具体的流程细节可以 review 该 **session** [9] 23:20 后的讲解，此外，对于该 extension API 更完整的使用，请查看如下 2020 session：

- **One-tap account security upgrades** [10]

推荐阅读

使用苹果账号登录你的应用

关注我们

我们是「老司机技术周报」，每周会发布一份关于 iOS 的周报，也会定期分享一些和 iOS 相关的技术。欢迎关注。

老司机技术周报

微信扫描二维码，关注我的公众号

关注有礼，关注【老司机技术周报】，回复「**2020**」，领取学习大礼包。

支持作者

这篇文章的内容来自于《**WWDC20** 内参》。在这里给大家推荐一下这个专栏，专栏目前已经创作了 108 篇文章，只需要 29.9 元。点击【阅读原文】，就可以购买继续阅读 ~

WWDC 内参 系列是由老司机周报、知识小集合以及 SwiftGG 几个技术组织发起的。已经做了几年了，口碑一直不错。 主要是针对每年的 WWDC 的内容，做一次精选，并号召一群一线互联网的 iOS 开发者，结合自己的实际开发经验、苹果文档和视频内容做二次创作。

参考资料

[1] Sign in with Apple: *https://developer.apple.com/sign-in-with-apple/*

[2] Sign in with Apple JS SDK: *https://developer.apple.com/documentation/signinwithapplejs*

[3] 《App Store 审核指南》: *https://developer.apple.com/cn/app-store/review/guidelines/*

[4] REST API: *https://developer.apple.com/documen*

tation/sign_in_with_apple/sign_in_with_apple_rest_api

[5] Sign In With Apple 从登录到服务器验证: *https://www.yuque.com/zhanglong/bb0s5d/cxbh7n*

[6] JSON Web Token: *http://www.ruanyifeng.com/blog/2018/07/json_web_token-tutorial.html*

[7] REST API: *https://developer.apple.com/documentation/sign_in_with_apple/sign_in_with_apple_rest_api*

[8] Verifying a User: *https://developer.apple.com/documentation/sign_in_with_apple/sign_in_with_apple_rest_api/verifying_a_user*

[9] session: *https://developer.apple.com/videos/play/wwdc2020/10173/*

[10] One-tap account security upgrades: *https://developer.apple.com/videos/play/wwdc2020/10666/*

收录于话题 #iOS 74个

下一篇

阅读原文