

はじめに 3

サンプルコード	4
クラウドファンディングと PEAKS	4
免責事項	4

第1章 Siri Shortcuts 13

1.1 NSUserActivity APIによるショートカットの実装	13
1.1.1 ショートカットの提供	14
1.1.2 Siriに追加	15
1.1.3 Siriからの提案	17
1.1.4 ショートカットのハンドリング	18
1.2 Intentによるショートカットの実装	20
1.2.1 Intent Definition	20
1.2.2 Intentによるショートカットの提供とハンドリング	23
1.2.3 Intents Extensionによるショートカットのバックグラウンド実行	24
1.3 ショートカットを提案する	28
1.4 「Add to Siri」ボタンでショートカットの追加を促す	29
1.4.1 INUIAddVoiceShortcutViewControllerとINUIEditVoiceShortcutViewController	31
1.4.2 登録されたショートカットのフレーズの表示	32
1.5 Apple WatchでのSiriショートカット	32
1.5.1 Apple Watchでのショートカットの提供とハンドリング	32
1.5.2 関連するショートカット	33
1.6 メディアコンテンツのためのショートカット	34
1.7 ショートカットの国際化	35
1.7.1 SiriKit Intents Definition Fileの多言語化	35
1.7.2 deferredLocalizedIntentsString	36

第2章 Notifications 37

2.1 はじめに	37
2.2 Notification（通知）の基礎	37
2.3 iOS 12における変更点	38
2.4 グループ通知（Grouped Notification）	38
2.4.1 グループ通知とは	38

2.4.2	グループ通知の実装方法	43
2.5	通知コンテンツ Extension (Notification Content Extension)	47
2.5.1	基本的な通知コンテンツ Extension	47
2.5.2	iOS 12 で追加された通知コンテンツ Extension の機能	49
2.6	通知の管理 (Notification Management)	55
2.6.1	仮許可 (Provisional Authroiztion)	55
2.6.2	通知設定の分類	57
2.6.3	通知設定へのアクセス	58
2.6.4	アプリ内通知設定画面への遷移	59
2.6.5	重大な通知 (Critical Alert)	60

第3章 Password AutoFillと Authentication Services 63

3.1	はじめに	63
3.1.1	パスワード管理の現状	63
3.1.2	Password AutoFillによるユーザー体験の向上	64
3.2	Password AutoFillの有効化	65
3.2.1	iOS アプリとドメインの関連付け	65
3.2.2	入力フィールドのコンテンツの設定	67
3.2.3	有効化の確認	68
3.3	Password AutoFillの3種類の動作	68
3.3.1	新しいパスワードの生成	69
3.3.2	保存されたユーザー名とパスワードの入力	72
3.3.3	SMSで送信されたワンタイムコードの入力	73
3.3.4	Web ページにおける Password AutoFill	74
3.4	外部サービスのログイン情報の使用	75
3.4.1	ユーザーが体験するフロー	75
3.4.2	アプリと Safari と外部サービスの間のデータの受け渡し	76
3.4.3	実装方法	77
3.4.4	iOS 11 への対応	78
3.5	AutoFill Credential Provider	79
3.5.1	iOS アプリの Xcode プロジェクトの設定	79
3.5.2	App Extension でのパスワードの選択画面の実装	82
3.5.3	QuickType バーへの対応	84
3.6	まとめ	89
3.7	参考文献	90

4.1	はじめに	91
4.2	垂直平面の検出.....	91
4.2.1	検出する平面のタイプを指定する	92
4.2.2	検出した平面のアラインメントを判別する	92
4.3	平面ジオメトリの取得.....	93
4.3.1	ARPlaneGeometry	94
4.3.2	ARSCNPlaneGeometry	94
4.3.3	ARPlaneAnchorのgeometryプロパティ	94
4.3.4	実装.....	95
4.4	画像検出	96
4.4.1	ARWorldTrackingConfigurationのdetectionImagesプロパティ	96
4.4.2	ARReferenceImage.....	97
4.4.3	ARImageAnchor	99
4.4.4	実装.....	100
4.5	画像トラッキング	102
4.5.1	ARImageTrackingConfiguration	103
4.5.2	実装.....	104
4.5.3	画像トラッキングと画像検出の違い.....	105
4.5.4	ワールドトラッキングコンフィギュレーションで画像トラッキング	106
4.6	フェイストラッキング.....	107
4.6.1	ARFaceTrackingConfiguration	107
4.6.2	ARFaceAnchor	108
4.6.3	ARFaceGeometry	109
4.6.4	ARSCNFaceGeometry	110
4.6.5	実装.....	111
4.7	3D物体検出.....	113
4.7.1	ARReferenceObject.....	113
4.7.2	detectionObjects.....	114
4.7.3	ARObjectAnchor.....	115
4.7.4	実装.....	115
4.7.5	3D物体スキンの公式サンプルと独自実装について	117
4.8	AR体験の永続化と共有	119
4.8.1	ARWorldMap.....	119
4.8.2	ワールドマップの取得	120
4.8.3	ワールドマップからの復元 / initialWorldMap.....	121
4.8.4	ワールドマップのアーカイブ/アンアーカイブ	121
4.8.5	ワールドマップ取得タイミング	122
4.9	ビデオフォーマット	123
4.9.1	ARConfiguration.VideoFormat	123

4.9.2	videoFormat.....	123
4.9.3	supportedVideoFormats.....	124
4.10	USDZ の利用.....	126
4.10.1	USDZ ファイルから MDLAsset を初期化する.....	126
4.10.2	MDLAsset から SCNScene を初期化する.....	127

第5章 デプス 129

5.1	はじめに	129
5.2	デプスとは？	129
5.3	Disparity（視差）と Depth（深度）.....	130
5.4	AVDepthData.....	131
5.5	iOS におけるデプス取得方法.....	132
5.6	デプス取得方法 1：撮影済み写真から取得	133
5.6.1	CGImageSource オブジェクトを作成する.....	133
5.6.2	CGImageSource から Auxiliary データを取得する.....	134
5.6.3	Auxiliary データから AVDepthData を初期化する.....	135
5.7	デプス取得方法 2：カメラからリアルタイムに取得	136
5.7.1	デプスが取れるタイプの AVCaptureDevice を使用する.....	136
5.7.2	デプスが取れるフォーマットを指定する.....	136
5.7.3	セッションの出力に AVCaptureDepthDataOutput を追加する.....	138
5.7.4	AVCaptureDepthDataOutputDelegate を実装し、AVDepthData を取得する.....	138
5.7.5	AVCaptureDataOutputSynchronizer で出力を同期させる.....	138
5.8	デプス取得方法 3：ARKit から取得.....	141
5.9	デプス応用 1：背景合成.....	142
5.9.1	背景合成とは.....	142
5.9.2	CIBlendWithMask.....	142
5.9.3	デプスデータの加工.....	143
5.10	デプス応用 2：2D 写真を 3D 空間に描画する.....	149
5.11	Portrait Effects Matte	150
5.11.1	AVPortraitEffectsMatte	151
5.11.2	Portrait Effects Matte の取得方法.....	152
5.11.3	Portrait Effects Matte の取得条件／制約.....	153

第6章 Core ML 2, Create ML 155

6.1	はじめに	155
6.2	Core ML のカスタムレイヤー.....	156
6.2.1	カスタムレイヤーの動作原理.....	156
6.2.2	カスタムレイヤーのエクスポート.....	158

6.2.3	カスタムレイヤーの実装	163
6.3	Create ML.....	170
6.3.1	画像認識	170
6.3.2	回帰問題と分類問題	176
6.3.3	回帰問題のサンプル	180
6.3.4	分類問題のサンプル	182
6.4	ニューラルネットワークの圧縮	185
6.4.1	圧縮の仕組み	185
6.4.2	圧縮のサンプルコード	187
6.5	まとめ	189
6.6	参考文献	190

第7章 Swift 4.0からSwift 4.2へのアップデート 191

7.1	はじめに	191
7.2	ABI安定化とSwift 5のリリース延期	191
7.3	Swift 4.1における変更点	192
7.3.1	SE-0075 : Adding a Build Configuration Import Test.....	192
7.3.2	SE-0143 : Conditional conformances (Swift 4.1)	193
7.3.3	SE-0166 : Swift Archival & Serialization (Swift 4.1)	194
7.3.4	SE-0157 : Support recursive constraints on associated types	194
7.3.5	SE-0161 : Smart KeyPaths: Better Key-Value Coding for Swift (Swift 4.1)	195
7.3.6	SE-0185 : Synthesizing Equatable and Hashable conformance (Swift 4.1)	196
7.3.7	SE-0186 : Remove ownership keyword support in protocols	197
7.3.8	SE-0189 : Restrict Cross-module Struct Initializers.....	198
7.3.9	SE-0190 : Target environment platform condition	199
7.3.10	SE-0187 : Introduce Sequence.compactMap(_:)	200
7.3.11	SE-0188 : Make Standard Library Index Types Hashable.....	202
7.4	Swift 4.2における変更点	202
7.4.1	SE-0054 : Abolish ImplicitlyUnwrappedOptional type	202
7.4.2	SE-0079 : Allow using optional binding to upgrade self from a weak to strong reference	204
7.4.3	SE-0143 : Conditional conformances (Swift 4.2)	205
7.4.4	SE-0185 : Synthesizing Equatable and Hashable conformance (Swift 4.2)	206
7.4.5	SE-0193 : Cross-module inlining and specialization	207
7.4.6	SE-0194 : Derived Collection of Enum Cases	208
7.4.7	SE-0195 : Introduce User-defined "Dynamic Member Lookup" Types.....	209
7.4.8	SE-0196 : Compiler Diagnostic Directives	213
7.4.9	SE-0212 : Compiler Version Directive	213
7.4.10	SE-0197 : Adding in-place removeAll(where:) to the Standard Library	215
7.4.11	SE-0199 : Adding toggle to Bool.....	215
7.4.12	SE-0202 : Random Unification	216

7.4.13	SE-0204 : Add last(where:) and lastIndex(where:) Methods	218
7.4.14	SE-0206 : Hashable Enhancements.....	220
7.4.15	SE-0207 : Add an allSatisfy algorithm to Sequence.....	221
7.4.16	SE-0201 : Package Manager Local Dependencies	221
7.4.17	SE-0208 : Package Manager System Library Targets.....	222
7.4.18	SE-0209 : Package Manager Swift Language Version API Update.....	224
7.5	まとめ.....	226
7.6	参考文献	226

第8章 Xcode 10の新機能 229

8.1	はじめに	229
8.2	UIの変更点.....	230
8.3	エディター機能のアップデート.....	233
8.3.1	複数カーソル	233
8.3.2	Code folding ribbon.....	235
8.3.3	Over-scroll.....	236
8.4	Playgroundのステップ毎実行.....	236
8.5	Run Script Phaseの改善.....	238
8.6	ソースコード管理機能.....	239
8.6.1	Bitbucket アカウントとGitLabのアカウントの設定.....	240
8.6.2	エディター上でGitステータス.....	241
8.7	SignpostsとInstrumentsによるパフォーマンス計測	243
8.7.1	同期的なメソッドの実行時間を計測する	244
8.7.2	非同期での実行時間計測	246
8.7.3	Signpost Events	248
8.7.4	OSLogのDisable設定	249
8.7.5	Auto Layoutのパフォーマンスを向上させる	250
8.8	エネルギーログの収集	253
8.9	おわりに	256
8.10	参考文献	256

第9章 Mojaveの新機能 257

9.1	はじめに	257
9.2	ダークモード	257
9.2.1	色のダークモード対応	258
9.2.2	画像のダークモード対応.....	261
9.2.3	ビューのダークモード対応.....	263
9.2.4	ダークモードのエフェクトに対応	266

9.2.5	Vibrancy.....	268
9.3	UIKit for Mac	269
9.3.1	iPadに対応する.....	270
9.3.2	キーボード操作に対応する.....	270
9.3.3	FirstResponderを制御する.....	271
9.3.4	マウス操作に対応する	273
9.3.5	ドラッグ&ドロップに対応する	274
9.3.6	ウインドウサイズの変更に対応する.....	274
9.3.7	アセットカタログで画像と色を管理する	274
9.3.8	メニューバーでプラスアルファの機能を提供する	275
9.3.9	マルチウインドウに備える.....	275
9.4	まとめ.....	275
9.5	参考文献	276

第10章 tvOS 12の新機能

277

10.1	はじめに	277
10.2	TVUIKit.....	277
10.2.1	TVUIKitの存在意義.....	278
10.2.2	TVPosterView クラス	280
10.2.3	TVCaptionButtonView クラス	284
10.2.4	TVCardView クラス	288
10.2.5	TVMonogramView クラス.....	289
10.2.6	TVLockupView クラス	292
10.2.7	TVLockupHeaderFooterView クラス.....	298
10.2.8	TVLockupViewComponent プロトコル.....	299
10.2.9	TVDigitEntryViewController クラス.....	301
10.3	UIKitのtvOS 向けアップデート	305
10.3.1	UILabelのenablesMarqueeWhenAncestorFocused プロパティ	305
10.4	Password AutoFill.....	305
10.4.1	iOS アプリとドメインの紐付け	306
10.4.2	tvOS アプリとドメインの紐付け	306
10.4.3	ログイン画面の作成.....	307
10.4.4	Password AutoFillの実行結果.....	308
10.4.5	Password AutoFill後に自動ログイン	309
10.5	フォーカス周りのアップデート.....	311
10.5.1	UIFocusItemContainer プロトコル.....	311
10.5.2	UIFocusItemScrollableContainer プロトコル	318
10.5.3	UIFocusMovementHint クラスによるアニメーション.....	321
10.5.4	UIKit ベースの画面での活用	323
10.5.5	VoiceOver 対応	324

10.6 まとめ.....	325
10.7 参考文献	325
おわりに	327

謝辞	327
権利表記	327

著者紹介	341
-------------------	------------

加藤 尋樹 @cockscomb	341
佐藤 紘一 @justin999_	341
石川 洋資 @_ishkawa	341
堤 修一 @shu223	341
吉田 悠一 @sonson_twit	341
池田 翔 @ikesyo	342
佐藤 剛士 @hatakenokakashi	342
大槻 一司 @tamadeveloper	342
所 友太 @tokorom	342
編集	342
永野 哲久 @7gano	342
横田 孝次郎 @macneko_ayu	342

Siri Shortcuts

加藤 尋樹 / @cockscomb

Siri は 2011 年にリリースされた iOS 5^{注1)} で、インテリジェントなアシスタントとして導入されました。2016 年の iOS 10^{注2)} では **SiriKit** フレームワークが追加され、開発者に Siri の API が提供されました。そして 2018 年の iOS 12^{注3)} で、**Siri ショートカット** が新たに追加されます。

Siri は音声アシスタントという印象が強いかもしれませんが。しかし iOS 9^{注4)} で Siri からの提案が導入されて以来、OS に内蔵されたインテリジェンスという側面が強調されてきました。

Siri ショートカットは、アプリのユーザーが何度も繰り返して行う（であろう）操作を「ショートカット」として取り扱えるようにします。アプリがユーザーの行動をショートカットとして iOS に提供すると、iOS はユーザーの行動を元にして、そのショートカットを提案するようになります。加えて、ユーザーはショートカットを Siri の音声アシスタントに登録できます。

Siri ショートカットはこのように、アプリを開いていなくてもアプリの機能を利用可能にするものです。Apple Watch や HomePod とも連携でき、iPhone を手に持っていないときにも利用できます。アプリの提供する機能がユーザーに何度も繰り返し利用されることを想定しているなら、Siri ショートカットに対応するべきです。

1.1

NSUserActivity API によるショートカットの実装

アプリが Siri ショートカットを提供するにはふたつの方法があります。ひとつは **NSUserActivity** API を使う方法、もうひとつは **Intent** を使う方法です。はじめに、比較的簡単な **NSUserActivity** によるショートカットの実装を紹介します。

NSUserActivity は **Foundation** フレームワークに含まれるクラスです。ユーザーがアプリを使用する上での重要な行動を、システムに登録できます。

NSUserActivity は iOS 8 で追加され、当初は Handoff をサポートするために使われるものでした。iOS 9 で Spotlight の機能が開発者に解放されると、**Core Spotlight** API とともに、検索の候補を提

注 1) Apple、iPhone 4S、iOS 5 および iCloud の提供を開始： <https://www.apple.com/jp/newsroom/2011/10/04Apple-Launches-iPhone-4S-iOS-5-iCloud/>

注 2) Apple、iOS として史上最大のリリースとなる iOS 10 をプレビュー： <https://www.apple.com/jp/newsroom/2016/06/13Apple-Previews-iOS-10-The-Biggest-iOS-Release-Ever/>

注 3) Apple、iOS 12 をプレビュー： <https://www.apple.com/jp/newsroom/2018/06/apple-previews-ios-12/>

注 4) iOS 9、9月16日より iPhone、iPad、iPod touch のユーザーに無料アップデートとして提供： <https://www.apple.com/jp/newsroom/2015/09/09iOS-9-Available-as-a-Free-Update-for-iPhone-iPad-iPod-touch-Users-September-16/>

供する機能をもつようになりました。

iOS 10ではSiriKit APIと関連して、Siriからアプリが起動されたときに、Siriからの情報を格納するためにも利用されるようになりました。

1.1.1 ショートカットの提供

iOS 12のSiriショートカットでも、`NSUserActivity`が拡張されています。`NSUserActivity`によるショートカットの提供はリスト1.1のように簡単です。

● リスト 1.1 `NSUserActivity` によるショートカットの提供

```
let activity = NSUserActivity(  
    activityType: "example.ios12programming.RecordMeal.AddRecord")  
activity.title = "Record meal time"  
activity.isEligibleForPrediction = true  
userActivity = activity
```

● リスト 1.2 Info.plist ファイルに `NSUserActivityTypes` を設定する

```
<key>NSUserActivityTypes</key>  
<array>  
    <string>example.ios12programming.RecordMeal.AddRecord</string>  
</array>
```

`NSUserActivity`を作成するには、リバースドメインで`activityType`を指定します。`activityType`は、リスト1.2のようにアプリのInfo.plistファイルの`NSUserActivityTypes`キーにも、文字列の配列として指定します。`NSUserActivity`の`title`プロパティには、行動のタイトルを設定します。そして`isEligibleForPrediction`を`true`にして、「予測されるのにふさわしい」行動であることを表します。iOSの場合は、`UIResponder`の`userActivity`プロパティに、作成した`NSUserActivity`を設定して完成です。

ショートカットにはサブタイトルと画像を設定できます。設定するにはリスト1.3のように、Core Spotlightの`CSSearchableItemAttributeSet`を使います。

● リスト 1.3 `CSSearchableItemAttributeSet` でサブタイトルと画像を設定する

```
let attributes = CSSearchableItemAttributeSet(  
    itemContentType: kUTTypeContent as String)  
attributes.contentDescription = "Record your meal time"  
attributes.thumbnailData = UIImage(named: "meal")!.pngData()  
activity.contentAttributeSet = attributes
```

1.1.2 Siri に追加

提供したショートカットは、設定アプリの「Siri と検索」に「Siri ショートカット候補」として表示されます（図 1.1）。



● 図 1.1 「Siri と検索」の設定に Siri ショートカットの候補が表示される

この候補を選択すると「Siri に追加」ダイアログが開きます（図 1.2）。このダイアログで、ショートカットを呼び出すフレーズを記録すると、Siri を使って音声でショートカットを呼び出せるようになります。

Notifications

佐藤 紘一 / @justin999_

2.1 はじめに

本章では、iOS 12から新しくなった**UserNotifications** フレームワークについて解説します。通知機能はiOS 3からあり、年々進化を重ねてきました。

今回のiOS 12のアップデートでは、通知をまとめる機能が追加されたり、通知UI上での操作の幅が広がるなど、さらに進化しました。

2.2 Notification (通知) の基礎

本節では、iOS 12の新機能の解説に先立って、Notification (以下、通知) の基礎について解説します。UserNotifications フレームワークを使って通知を送る手順は次のとおりです。

1. アプリ起動中の任意のタイミングで通知許可を得る。
2. (ローカル通知の場合) 通知をスケジュールする。
3. (プッシュ通知の場合) サーバーから通知の情報を送る。
4. 設定されている方法で通知が表示される。

ローカル通知では、アプリ内で通知を生成します。そのときに、通知する条件 (時間や場所など) を同時に設定できます。プッシュ通知では、通知したい内容をJSON形式のデータで自社のサーバーから、Appleの**Apple Push Notification service (APNs)** に送信します^{注1)}。

ローカル通知を送る場合のコードや、プッシュ通知を送る場合のJSONデータの書式については、次節以降に具体的な例を添えて解説しています。また、解説の中でプッシュ通知を送信するときは、デバイス・アプリの通知設定がオンになっていることを前提とします。

注 1) <https://developer.apple.com/documentation/usernotifications>

2.3 iOS 12における変更点

iOS 12では通知に関して、大きく次の変更がありました。

グループ通知 (Grouped Notification)

通知をアプリ単位や任意の単位でまとめることができるようになります。ユーザが通知を確認しやすくするために重要な機能です。

通知コンテンツ Extension (Notification Content Extension)

通知のUIをカスタマイズする機能です。今まではできることが限られていましたが、iOS 12でインタラクティブに操作できるようになるなど、大きく拡張されました。

通知の管理 (Notification Management)

通知許可をユーザに求めなくても通知を送れるようになる仮許可など、通知の設定が新しくなりました。

2.4 グループ通知 (Grouped Notification)

本節では、iOS 12で新しく導入された、**グループ通知 (Grouped Notification)** について解説します。

2.4.1 グループ通知とは

グループ通知は名前のとおり、ロック画面や**通知センター**に表示される通知をグループ化して表示する機能です。



iOS11 **iOS12**

● 図 2.1 iOS 11 と iOS 12 の通知センターの比較

グループされた通知をタップすると、まとめられていたものが縦に並べて表示されます。



● 図 2.2 グループ化された通知が展開された状態

開発者が何も設定しない場合、アプリごとに通知がまとめられます。開発者側で独自にグループを決めることもできます。

Password AutoFillとAuthentication Services

石川 洋資 / @_ishkawa

3.1 はじめに

Password AutoFillは、iOS 11で導入されたパスワードの自動入力仕組みです。iOS 12では自動入力対象が広がり、より幅広い場面で手入力を省けるようになりました。本章では、Password AutoFillの全体を把握できるようにするため、iOS 11で導入された仕組みも含めて解説します。

Authentication ServicesはiOS 12で導入された、認証をサポートする仕組みです。Authentication Servicesの役割は大きく分けると2つあり、1つはSafariのログインセッションを利用して外部サービスを利用した認証をすること、もう1つはサードパーティのアプリからのAutoFillへのパスワードを提供／受け入れることです。本章では、どちらの役割についても解説します。

3.1.1 パスワード管理の現状

具体的な仕組みの説明に入る前に、パスワード管理が置かれている現状をおさらいしておきましょう。現在、インターネットサービスは無数にあり、いたるところで本人確認のためのIDとパスワードが必要とされています。本来、パスワードは第三者に知られることなく守られるべきものですが、強度不足や情報漏洩により第三者に渡ってしまうことがあるのが現状です。

パスワードを適切に管理するため、サービスの利用者は次の2点を守るべきでしょう。

- 高い強度のパスワードを使用する
- サービスごとに異なるパスワードを使用する

サービスごとに個別に設定された高い強度のパスワードを人間が記憶するのは現実的ではないため、パスワード管理ツールなどの記憶を補助する仕組みが必要です。

記憶を補助する仕組みを利用しない場合、人間が頭でパスワードを記憶しなければなりません。したがって、パスワードには記憶しやすさが必要となり、結果として低い強度のパスワードの使い回しが起きやすくなります。さまざまな場所で警鐘が鳴らされているにもかかわらず、低い強度のパスワードの使い回しがなくなる理由、それが手軽だからでしょう。

より多くの人が適切にパスワードを管理できるようにするには、パスワード管理ツールがもっと手軽に利用できなければなりません。Password AutoFillは堅牢なパスワード管理を手軽に行う仕組みであり、現状を開閉できる見込みがあります。

3.1.2 Password AutoFill によるユーザー体験の向上

ユーザーにとって、新規登録やログインは煩わしいものです。その煩わしさの大半を占めるのは、IDやパスワードの入力でしょう。新規登録時には、新たに高い強度のパスワードを考えて、それをどこかに控えておかなければなりません。ログイン時には、そのアプリに使用したパスワードを探して入力しなければなりません。

Password AutoFillは新規登録やログインの煩わしい部分をiOSが担い、わずか数タップで処理を完了させるというものです。新規登録時には、iOSがパスワードを生成して自動入力します。このとき、自動生成されたパスワードはキーチェーンに保存されます。そして、再度ログインする時には、iOSがキーチェーンからパスワードを取り出して自動入力します。したがって、ユーザーはどのようなパスワードを入力したのか意識する必要はありません。



● 図 3.1 新規登録時(左)とログイン時(右)の Password AutoFill

iOSが新規登録時に自動生成するパスワードは、高い強度を持っています。つまり、Password AutoFillはわずか数タップという手軽さと、パスワードの堅牢さを両立しているということです。視点を変えれば、新規登録やログインの負荷が下がるという短期的なユーザー体験の向上と、パスワードを不正アクセスから守れるという長期的なユーザー体験の向上を両立しているともいえます。

3.2 Password AutoFillの有効化

iOSは、Password AutoFillに対応したアプリでなくても、Password AutoFillが利用可能か自動的に判定し、可能であればユーザーにPassword AutoFillの利用を促します。しかし、Password AutoFillの完全な体験が確実に得られるようにするため、次の2点の対応をしておくことが推奨されています。

- アプリとドメインを関連付ける。
- 入力フィールドのコンテンツの種類を適切に設定する。

この節では、この2つの手順を説明します。

3.2.1 iOS アプリとドメインの関連付け

iOS アプリとドメインの関連付けは、iOS アプリと Web サイトの間で連携を行うための仕組みです。従来のバージョンのiOSでも、iOS アプリと Web サイトでパスワードなどの機密情報を共有したり、Web サイトからiOS アプリへのシームレスな移動の実現に使われています。

ドメインの関連付けには、次の2つが必要となります。

- iOS アプリの設定で関連付けるドメインを宣言する。
- ドメインが割り当てられたサーバーに関連付けるアプリを宣言するファイルをホストする。

iOS アプリは関連付けるドメインを、ドメインが割り当てられたサーバーでは関連付けるアプリを宣言することで、両者が同じ所有者であることが確認できる仕組みになっています。したがって、他人のドメインを勝手に関連付けることはできません。

■ iOS アプリの設定

iOS の設定で関連付けるドメインを宣言するには、Xcode プロジェクトのアプリのビルドターゲットの「Capabilities」の「Associated Domains」を有効にします。有効にすると「Domains」というリストが表示されるので、`webcredentials:ios12demo.firebaseio.com`のように `webcredentials:` に続いて関連付けるドメインを宣言します。

第4章

ARKit

堤 修一 / @shu223

4.1 はじめに

ARKitはその精度と安定性、そしてiPhone/iPadという広く普及しているデバイスで利用可能という点から注目を浴びました。しかし、はじめて発表された当時のバージョンでは水平面しか検出できず、AR体験の共有もできなかったため、実現できることがかなり限られていたのも事実です。

その後Appleは矢継ぎ早にARKitをアップデートし、**ARKit 1.5**（2018年3月リリース）では垂直平面検出や任意形状の平面検出が可能となり、**ARKit 2.0**（2018年9月リリース）ではAR体験の永続化や共有、画像トラッキングや3D物体検出も可能になりました。これによりARKitで実現できるアプリケーションの幅が大きく広がります。たとえば、画像検出や3D物体検出により美術館の絵画や彫像に仮想コンテンツを設置しておくことが可能になります。また、AR体験の共有機能により、あるユーザーが設置した仮想コンテンツを別のユーザーが閲覧するということが可能になります。

本章では、ARKit初発表時以降に登場した新機能^{注1)}について解説します。なお、ARKitの基礎から順序立てて学びたい方は『iOS 11 Programming』の第2章を参照してください^{注2)}。

4.2 垂直平面の検出

iOS 11.3（ARKit 1.5）より、従来の水平面に加えて、垂直平面を検出する機能が追加されました。これにより、ARKit 1.0で検出した現実世界の水平面を利用して行っていたことすべてが垂直平面に対してもできるようになりました。壁に仮想コンテンツを表示したり、ドアの高さを測ったりといったことが可能になったわけです。

この新機能の利用にあたって、実装面でやることは非常にシンプルです。本節では、本機能に関連する追加APIと実装を解説していきます。

■ サンプルコード: 01_PlaneDetection

注1) フェイストラッキングなど、ARKit 1.0に含まれるものの、WWDC 2017の時点ではまだ発表されていなかった（2017年秋のiPhone X発表と共に発表された）機能も本章では新機能として取り上げます。

注2) 3行で実現できるARKitのはじめの一步から、水平面の検出・仮想オブジェクトの設置・インタラクションといった重要な基礎機能の解説、そして現実世界の長さを測るメジャーアプリや空中に絵や文字を描くお絵かきアプリといった応用アプリケーションの実装方法を解説しています。iOS 12が登場した今でも古びない内容ですので、ARKitの実装を基礎から学びたい方はぜひご検討ください。

4.2.1 検出する平面のタイプを指定する

ARWorldTrackingConfiguration.PlaneDetection 構造体のスタティックプロパティとして **vertical** が追加されました。

```
static var vertical: ARWorldTrackingConfiguration.PlaneDetection
```

これを **ARWorldTrackingConfiguration** の **planeDetection** プロパティに指定して AR セッションを開始することで、垂直平面が検出されるようになります。

```
let configuration = ARWorldTrackingConfiguration()
configuration.planeDetection = .vertical
sceneView.session.run(configuration)
```

なお **ARWorldTrackingConfiguration.PlaneDetection** は **OptionSet** プロトコルに準拠しているので、複数セットし、垂直平面・水平面の両方を検出対象にできます。

```
configuration.planeDetection = [.horizontal, .vertical]
```

垂直平面を検出した場合の挙動は水平面と同様で、アンカーとして **ARPlaneAnchor** が得られます。

```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode,
              for anchor: ARAnchor) {
    guard let planeAnchor = anchor as? ARPlaneAnchor else {fatalError()}
}
```

4.2.2 検出した平面のアラインメントを判別する

ARWorldTrackingConfiguration の **planeDetection** プロパティに **horizontal** と **vertical** の両方を指定した場合、実際に検出した平面が水平・垂直どちらなのかによって処理を分けたいケースがあります。その場合、**ARPlaneAnchor** の **alignment** プロパティが利用できます。

```
var alignment: ARPlaneAnchor.Alignment { get }
```

このプロパティの型である **ARPlaneAnchor.Alignment** は enum として定義されており、case は **horizontal** (水平) と **vertical** (垂直) とがあります。

たとえば得られた平面が垂直平面か水平面かによって色を分けて可視化したい場合は、次のような実装になります。

```
func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode,
              for anchor: ARAnchor) {
    guard let planeAnchor = anchor as? ARPlaneAnchor else {fatalError()}

    // アラインメントによって色をわけ
    let color: UIColor
        = planeAnchor.alignment == .horizontal ? UIColor.yellow : UIColor.blue
}
```

4.3 平面ジオメトリの取得

iOS 11.3 (ARKit 1.5) より、**検出した平面の形状を取得**できるようになりました。この形状は矩形には限りません。たとえば次のように、丸テーブルを平面検出し、その円形（に近い）ジオメトリを取得し、可視化できます。



なお、水平面だけでなく、ARKit 1.5で検出可能になった**垂直平面にも本機能は有効**です。まずは関連する新クラス、新プロパティについて解説し、実装方法を示します。

■ サンプルコード: 02_PlaneGeometry

第5章 デプス

堤 修一 / @shu223

5.1 はじめに

デプス（深度）が取得可能になったのはiOS 11から、つまり比較的最近のことです。従来のカメラやGPSがデジタルの世界と我々が生きる現実世界を繋ぐ重要な役割を担い、アプリ開発者に多くの創造性を与えてくれたのと同様に、モバイルデバイスで「奥行き」が分かるようになったというのはアプリ開発の次元がひとつ増えたようなものです。本章ではそんなデプスについて、iOS 12の新APIだけでなく、基礎から応用まで順序立てて解説していきます。

5.2 デプスとは？

デプスとは、「奥行き」を示す情報です。1チャンネルの画像データで表され、



● 図 5.1 左：カラー画像 / 右：デプスマップ

このデータは**デプスマップ**とも呼ばれます。各ピクセル値が奥行きを表し、0.0（近い）～1.0（遠い）、あるいは0（近い）～255（遠い）といった値をとります。従来のカラー画像も値だけみると同じようなものに見えますが、各ピクセル値はR、G、B、Aあるいはグレイといった「色」の度合いを表すので、意味するところはまったく違います。

1チャンネル画像なので、上の画像のようにグレースケールで描画されることが多いですが、ヒートマップのように赤～青で色付けされることもあります。

5.3 Disparity (視差) と Depth (深度)

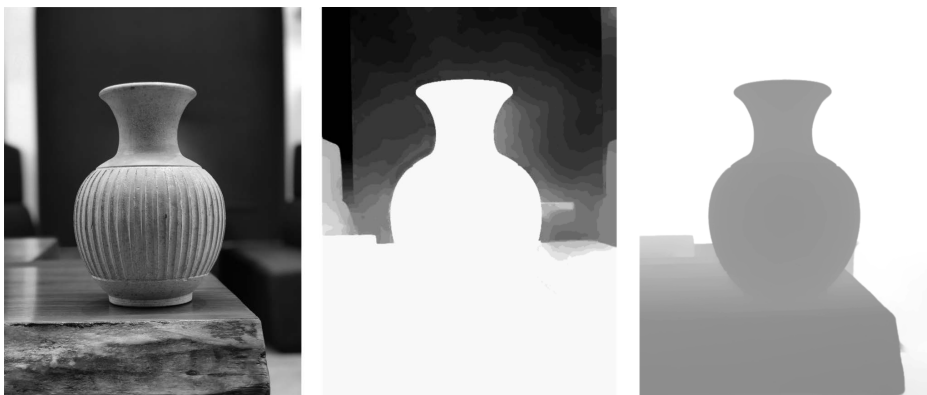
iPhone 7 Plus、8 Plus、X、XSといったiOSデバイスは背面に2つのレンズをもつデュアルカメラを搭載していますが、このデュアルカメラではその2つのレンズの視差を利用して世界の奥行きを推定できます。

視差は対象物体が近いほど大きく、遠いほど小さくなります。つまり、「視差」(Disparity) と奥行きを示す「深度」(Depth) は逆数の関係にあります^{注1)}。

またiPhone XやXS、XRといったiOSデバイスの前面に搭載されているTrueDepthカメラは、視差方式と違い赤外線を用いて直接的に深度を計測します。

これらの理由から、iOSで取得できるデプスデータにはDisparity (視差) と Depth (深度) の2種類があります。DisparityとDepthは取得後も相互変換でき、上述したように逆数の関係にあります。本章ではこれら「Disparity」と「Depth」を総称して「デプス」と表現することにします。

逆数の関係にあるということは、デプスマップを1チャンネルのグレースケール画像として可視化した場合に、Disparityは近くにあるものほど白くなり、Depthは遠くにあるものほど白くなる、といった真逆の結果になるということです。この違いはデプスマップをマスクとして使用する場合などに重要になってきます^{注2)}。



● 図 5.2 カラー (左)、Disparity (中)、Depth (右)

注 1) この視差と深度の関係は、WWDC 2017 のセッション「Capturing Depth in iPhone Photography」でわかりやすく解説されています。

注 2) 「5.9 デプス応用1：背景合成」

5.4 AVDepthData

iOSにおけるデプスの取得方法の解説に入る前に、もっとも重要なクラスをひとつ紹介します。

AVDepthDataは、デプスデータを表すクラスです。以降の節でさまざまなフレームワークでのデプス取得方法を示しますが、どの方法を使用するにせよ、(ほとんどの場合は)最終的にデプスデータをこの型で得ることになります。iOS 11以降で利用可能です。

AVDepthDataは多くのプロパティやメソッドを持ちますが、もっとも重要なのは**depthDataMap**プロパティです。

```
var depthDataMap: CVPixelBuffer { get }
```

このプロパティはデプスマップのピクセルデータを**CVPixelBuffer**型で保持します。**CVPixelBuffer**はiOS 4の頃から存在し、多くの画像を扱うフレームワークがこの型をサポートしています。ですので、デプスデータを**AVDepthData**として取得してしまえば、あとはそのデプスマップをCore ImageでもMetalでも、従来の画像処理方法で好きなように処理可能です。

得られたデプスデータがDisparityなのかDepthなのかといった「デプスデータの種別」は、**depthDataType**プロパティより確認できます。

```
var depthDataType: OSType { get }
```

型が**OSType**となっていますが、次のいずれかのタイプが得られます。DisparityかDepthか、またビット深度が16か32か、の4種類です。

- **kCVPixelFormatType_DisparityFloat16**
- **kCVPixelFormatType_DisparityFloat32**
- **kCVPixelFormatType_DepthFloat16**
- **kCVPixelFormatType_DepthFloat32**

AVDepthDataでは、次のメソッドを使用してDisparityとDepthの相互変換が可能です。引数には変換したいデプスデータタイプを指定します。

```
func converting(toDepthDataType depthDataType: OSType) -> Self
```

たとえば次のように実装することで、DisparityをDepthへ、DepthをDisparityへビット深度を維持しつつ変換できます。

iOS 12 Programming

加藤 尋樹
@cockscomb

佐藤 紘一
@justin999_

石川 洋資
@_ishkawa

堤 修一
@shu223

吉田 悠一
@sonson_twit

池田 翔
@ikesyo

佐藤 剛士
@hatakenokakashi

大榎 一司
@tamadeveloper

所 友太
@tokorom

編集
横田 孝次郎
@macneko_ayu

編集
永野 哲久
@7gano



『iOS12 Programming』
電子版: ¥3,200 製本版: ¥3,500

購入して続きを読む



Core ML 2, Create ML

吉田 悠一 / @sonson_twit

6.1 はじめに

WWDC 2018で**Core ML 2**が発表されました。本章では、Core ML 2およびCreate ML、前書『iOS 11 Programming』^{注1)}で紹介したCore MLから、iOS 11.2以降の差分について説明します。以降、iOS 11.2以降のCore MLを**Core ML 1.2**と呼び、iOS 11.2以前にサポートされていたものを**Core ML 1**と書きます。また、Core ML全般のことを**Core ML**と書くことにします。Core MLは、iOS 11以降でサポートされる機械学習のモデルをデプロイするためのフレームワークです。Core MLを使うと、開発者は、Appleがあらかじめ実装したランタイムにモデル^{注2)}とパラメーターを流し込むだけで機械学習の機能を実現できます。開発者は、**scipy**^{注3)}や、**Keras**^{注4)}といったツールから**Core ML Tools**^{注5)}を使って**mlmodel ファイル**^{注6)}をエクスポートし、Xcodeに読み込ませるだけで機械学習を機能として組み込むことができます。また、Core MLはデバイス毎に最適化され、GPUやSIMDを適切に利用し、高速に運用できるという利点を持ちます。

Core MLの本質的な課題は、学習を他のツールに依存していること、Appleが実装した機械学習のモデル以外を開発者が使えないことにありました。具体的には、Core ML 1は、雨後の筍のように出てくる新しく提案されたニューラルネットワークの手法を、簡単に実装・デプロイできないという短所を持っていました。しかし、Core MLのマイナーバージョンアップと、iOS 12 (Core ML 2) の登場で、これらの課題は部分的に、あるいは全面的に解決されました。さらに、Core ML 2ではmlmodelファイルを圧縮する機能が追加され、より強力なフレームワークになりました。

注 1) <https://peaks.cc/iOS11>

注 2) 機械学習における推論の形式、線形回帰、ロジスティック回帰、サポートベクターマシン、ニューラルネットワークなどのこと。形式だけに止まらず、パラメーターや次元の数までもモデルに含まれることがある。

注 3) Python のための科学・工学のためのオープンソース・ライブラリ。

注 4) Python で書かれたニューラルネットワークのライブラリ。

注 5) 各種ライブラリで作った機械学習のモデルとパラメーターを Core ML で取り扱うためのオープンソースのツール。

注 6) Core ML で機械学習のモデルやそれに付随するパラメーターをシリアルライズしたファイルのこと。詳しくは前書の iOS 11 Programming をご参照ください。

Core ML 1.2以降、「Appleが実装した機械学習のモデル以外を開発者が使えない」という課題は、ニューラルネットワークを利用する場合に限り、解決されました。Core ML 1.2以降でサポートされるカスタムレイヤーは、Core MLで実装されていないニューラルネットワークのレイヤーを、開発者が独自に実装してコードで肩代わりできるフレームワークです。ニューラルネットワークを使った手法は、今や日進月歩をとおり越して、秒進分歩ともいうべきスピードで、研究開発が進んでいます。初期の段階では、典型的な行列計算と非線形関数を組み合わせて実現できるニューラルネットワークがほとんどでした。しかし今では、ひと昔前では考えられないようなアイデアを取り入れた手法がどんどん提案されています。Core ML 1.2でカスタムレイヤーが実装されたことによって、この大きな短所が解決されたのです。

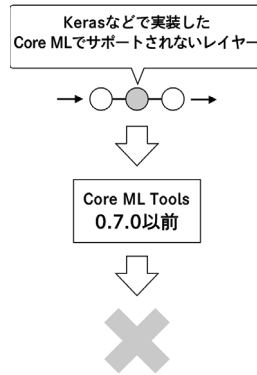
6.2.1 カスタムレイヤーの動作原理

ニューラルネットワークをCore MLでデプロイするときの処理の流れは次のようになります。

- Keras, Caffeなどのツールでニューラルネットワークを実装する
- データを使い、パラメーターを学習する
- Core ML Toolsで、モデルとパラメーターをmlmodelファイルにエクスポートする
- Xcodeでプロジェクトにmlmodelファイルをインポートする
- iOSのコードからmlmodelファイルを使って、予測のためのインスタンスを生成する

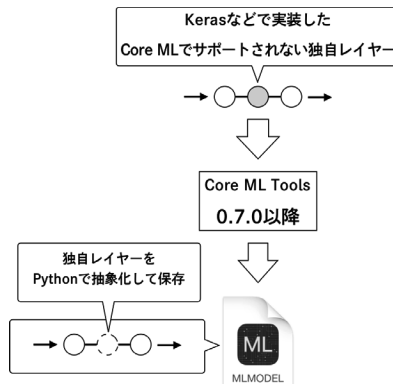
Core ML 1では、Core MLがサポートしない活性化関数等を含んだモデルを作成した場合^{注7)}、mlmodelファイルにエクスポートするときにエラーが発生します（図6.1）。

注7) ニューラルネットワークに非線形性を持たせるために各レイヤーごとに挿入する関数のこと。詳しくは、機械学習の専門書、あるいは前書『iOS 11 Programming』を参照してください。



● 図 6.1 Core ML 1.2 以前では独自実装のレイヤーはサポートされない

Core ML 1.2では、サポートされない活性化関数やレイヤーを抽象化してmlmodelファイルにエクスポートし、ランタイム時に開発者が自前で実装したSwiftやObjective-Cで実装したクラスにマッピングできます（図6.2）。



● 図 6.2 Core ML Tools 0.70 による独自レイヤーの変換

マッピングされるクラスは、開発者が**MLCustomLayerプロトコル**に従い実装することになります。MLCustomLayerプロトコルは、ニューラルネットワークのレイヤーの初期化、実際の推論における計算などのメソッドからなり、Core ML 1.2と開発者の独自レイヤーのインターフェースの役割を果たします（図6.3）。

Swift 4.0 から Swift 4.2 へのアップデート

池田 翔 / @ikesyo

7.1 はじめに

本章では、2017年にXcode 9.0とともにリリースされたSwift 4.0からのアップデートとして、Swift 4.1とSwift 4.2のアップデートについて解説します。

解説する項目は、Swift 4.1^{注1)}とSwift 4.2^{注2)}、それぞれのCHANGELOGやswift-evolutionの各プロポーザルに基づいています（プロポーザルはSE-0200のように、SE-xxxxという形式で記載されます）。実装されたすべてのプロポーザルを扱っているわけではないので、取り上げられなかった項目に興味がある方は、ぜひswift-evolutionのリポジトリをチェックしてみてください^{注3)}。

Swift 4.0のCodable^(SE-0166, SE-0177)とSmart KeyPaths^(SE-0161)について復習したい方は、前書の『iOS 11 Programming』の第4章を読むことをお勧めします^{注4)}。

7.2 ABI安定化とSwift 5のリリース延期

2017年のSwift 4まで、Swiftのメジャーアップデートは毎年、iOSとXcodeの新しいメジャーバージョンに合わせてリリースされてきました。その流れからすると、2018年もiOS 12・Xcode 10とともにSwift 5がリリースされるところですが、実際にはSwift 4.2というバージョンになりました。

Swift 5をリリースする前にSwift 4.2というバージョンもリリースする予定であることは、Swift 4.1リリースの少し前からSwift.orgのブログで公表されていました^{注5)}。この公表の時点では、WWDC 2018でXcode 10のベータがリリースされる前にSwift 4.2がリリースされるだろうと想像できましたが、WWDC 2018でSwift 5のリリースは2019年初頭に延期されることが発表されました^{注6)}。

なぜこのようなリリースプロセスになったのでしょうか？それはSwift 5の最大の目標であるABI (Application Binary Interface) の安定化（バイナリ互換性の提供）の実現と関係しています。

注1) <https://github.com/apple/swift/blob/master/CHANGELOG.md#swift-41>

注2) <https://github.com/apple/swift/blob/master/CHANGELOG.md#swift-42>

注3) <https://github.com/apple/swift-evolution>

注4) <https://peaks.cc/iOS11#chapter4>

注5) <https://swift.org/blog/4-2-release-process/>

注6) <https://developer.apple.com/videos/play/wwdc2018/401/>

ABIの安定化とは、異なるバージョンのSwiftコンパイラーでビルドされた複数のバイナリ間での、バイナリフォーマットの互換性を保証することです。ABI安定化が達成されると、Objective-Cがそうであるように、Swiftの標準ライブラリとランタイムをOSに同梱できるようになります。

現在のSwiftはOSに標準ライブラリとランタイムが含まれず、Swiftでビルドした各アプリにそれらが同梱されるようになっていました。Swiftで書かれたアプリが、Objective-Cで書かれたアプリよりもファイルサイズが大きいのは、これが理由です。

OSに標準ライブラリとランタイムを同梱できるようになると、アプリごとにそれらを同梱する必要がなくなり、Swiftで書かれたアプリをApp Storeからダウンロードするときのファイルサイズを小さくできます。結果として、アプリのダウンロード時間を短縮できたり、アプリが占有するディスク容量を減らせませす。

ご存知の方も多いかもしれませんが、ABI安定化は当初Swift 3で達成される予定でした。しかし実際には予想以上に時間が掛かっており、Swift 4にも間に合わず、ABI安定化はSwift 5での最重要事項とされてきました。もし2018年の秋に間に合わず、ABI安定化の達成をSwift 6に延期してしまうと、さらに1年待たされてしまうことになります。

そこでAppleは、iOS・Xcodeのメジャーリリースと歩調を合わせることにこだわらず、ABIの安定化をSwift 6に延期する代わりに、Swift 5自体のリリースを数ヶ月延期して、延期の期間を最小化することを選択したと考えられます。

7.3

Swift 4.1における変更点

Swift 4.1は、Xcode 9.3とともに2018年3月にリリースされました^{注7)}。

Swift 4.1はジェネリクス関係の機能追加やコードサイズを削減するための最適化モードの導入^{注8)}、Swift 5に向けたABI安定化のための進捗などを含むリリースとなっています。

7.3.1 SE-0075 : Adding a Build Configuration Import Test

SE-0075^{注9)}では、`#if canImport`という検査で、モジュールの存在確認とインポート可否に応じた条件付きコンパイルができるようになりました。

たとえばUIKitとAppKit両方をサポートするライブラリを作る場合、Swift 4.0まではOSがmacOSなのか、iOSなのかtvOSなのか、はたまたLinuxなのかなど、どのOSなのかを判別するために、リスト7.1のように書く必要がありました。

注7) <https://swift.org/blog/swift-4-1-released/>

注8) <https://swift.org/blog/osize/>

注9) <https://github.com/apple/swift-evolution/blob/master/proposals/0075-import-test.md>

● リスト 7.1 OS 判別の実装例

```
#if os(iOS) || os(tvOS)
    import UIKit
    class MyView: UIView {}
#elseif os(macOS)
    import AppKit
    class MyView: NSView {}
#else
    class MyView: CustomView {}
#endif
```

Swift 4.1からは`#if canImport`を使うとリスト 7.2のように書けます。

● リスト 7.2 SE-0075 : `canImport`を使用した実装例

```
#if canImport(UIKit)
    import UIKit
    class MyView: UIView {}
#elseif canImport(AppKit)
    import AppKit
    class MyView: NSView {}
#else
    class MyView: CustomView {}
#endif
```

モジュールが使用可能かどうかだけに着目し、OSを気にする必要がなくなったことが分かります。

7.3.2 SE-0143 : Conditional conformances (Swift 4.1)

SE-0143^{注10)}は**Conditional conformances**と呼ばれる、プロトコルへの条件付き準拠の機能です。一例としてはArrayの要素がEquatableであれば、そのArrayもまたEquatableになる、というものです（リスト 7.3）。

● リスト 7.3 Array の Conditional conformances

```
extension Array: Equatable where Element: Equatable {
    ...
}
```

`extension`でプロトコルへの準拠を宣言する際に`where`節で準拠するための条件を指定します。

Swift 4.1ではConditional conformancesを利用し、標準ライブラリの次のジェネリックなデータ型が、要素がEquatableである時にEquatableに準拠するようになりました。

注 10) <https://github.com/apple/swift-evolution/blob/master/proposals/0143-conditional-conformances.md>

iOS 12 Programming

加藤 尋樹
@cockscomb

佐藤 紘一
@justin999_

石川 洋資
@_ishkawa

堤 修一
@shu223

吉田 悠一
@sonson_twit

池田 翔
@ikesyo

佐藤 剛士
@hatakenokakashi

大榎 一司
@tamadeveloper

所 友太
@tokorom

編集
横田 孝次郎
@macneko_ayu

編集
永野 哲久
@7gano



『iOS12 Programming』
電子版: ¥3,200 製本版: ¥3,500

購入して続きを読む



Xcode 10の新機能

佐藤 剛士 / @hatakenokakashi

8.1 はじめに

WWDC 2017で発表されたXcode 9では多くのアップデートがありました。エディターの機能向上ではSwift向けのリファクタリング、ソースコード管理機能としてはGitHubとの統合機能が追加、デバッグ機能としてはワイヤレスデバッグの追加やビューデバッグの改善が行われました。WWDC 2018で発表された**Xcode 10**では去年に負けず劣らず、大幅な機能改善や開発効率向上が見込まれる機能が追加されました。複数カーソル選択の追加や、BitbucketやGitLabの統合、**Signposts**によるパフォーマンス計測など、開発効率だけでなく、アプリの品質を高められる機能が数多く追加されました。筆者もXcode 10は使っていてとても気持ちよく開発できるIDEと感じています。それほど今回のアップデートは注目すべきものです。

本章では次の新機能や改善点について解説します。

- UIの変更点
- エディター機能のアップデート
 - 複数カーソル
 - Code folding ribbon
 - Over-scroll
- Playgroundのステップ毎実行
- Run Script Phaseの改善
- ソースコード管理機能
 - Bitbucket アカウントとGitLabのアカウントの設定
 - エディター上でGitステータス
- SignpostsとInstrumentsによるパフォーマンス計測
 - 同期的なメソッドの実行時間を計測する
 - 非同期での実行時間計測
 - Signpost Events
 - OSLogのDisable設定
 - Auto Layoutのパフォーマンスを向上させる
- エナジーログの収集

また、Xcode 10のリリースに合わせて、Xcodeのヘルプページも刷新されました。

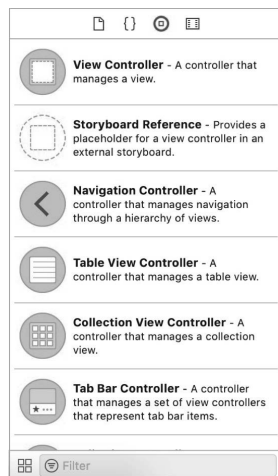
困ったことがあればこちらを参照しましょう。

<https://help.apple.com/xcode/mac/10.0/index.html>

8.2 UIの変更点

Xcode 10になり、ライブラリペインが廃止されました。UIの変更により戸惑う開発者の方も多いかと思います。同じ機能がXcode 10でどのように利用できるのかを解説していきます。

Xcode 9まではユーティリティ領域のライブラリペインがあり、ファイルテンプレートやコードスニペット、オブジェクト、メディアそれぞれのライブラリにアクセスができました。

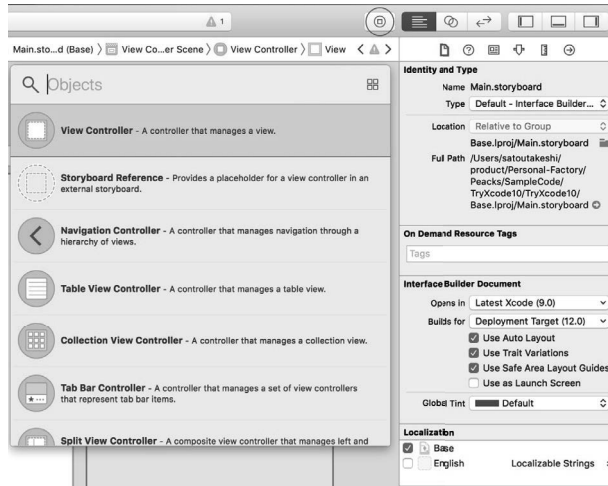


● 図 8.1 Xcode 9 までのユーティリティ領域

Xcode 10ではライブラリペインのそれぞれの機能が他の場所に移動しています。

まず、ファイルテンプレートが、特定のペイン領域に表示されなくなりました。ファイルテンプレートを使用するにはメニューのFile > New > Fileを実行します。

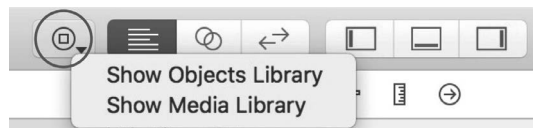
次に、オブジェクトライブラリがワークスペースツールバーに移動されました。ストーリーボードエディターを表示している場合にワークスペースツールバーのエディター設定ボタンの左隣に配置されています。



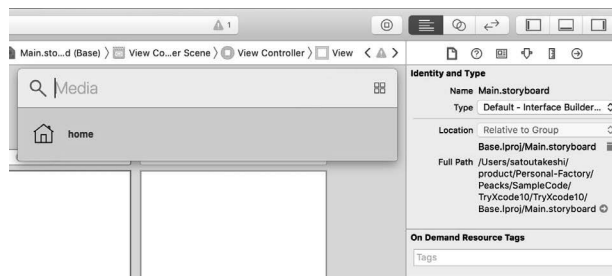
● 図 8.2 オブジェクトライブラリを表示

クリックするとオブジェクトライブラリが別ウィンドウで表示されるので、使用したいオブジェクトをドラッグしましょう。ウィンドウは別領域をクリック、またはオブジェクトをドラッグすると消えてしまいます。ウィンドウが消えないようにするには、**option**キーを押しながらオブジェクトライブラリボタンをクリックするか、オブジェクトをドラッグ中に**option**キーを押します。ウィンドウが固定化されて引き続き操作ができるようになります。ショートカットは`command + shift + L`です。

次にメディアライブラリですが、オブジェクトライブラリボタンを長押しするとメニューが表示されます。そこからメディアライブラリを選択するとリストが表示されます。



● 図 8.3 長押しでオブジェクトライブラリとメディアライブラリを選択



● 図 8.4 メディアライブラリのリスト表示

Mojaveの新機能

大槻 一司 / @tamadeveloper

9.1 はじめに

WWDC 2018でmacOSの新バージョン**Mojave**が発表されました。デスクトップを整理する「スタック」や、Finderの新しい表示方法である「ギャラリービュー」などが追加されました。またQuickLookでは編集が可能となり、スクリーンショット撮影では動画を取り込む機能が追加されています。トラッキング防止機能がSafariに搭載されるなど、セキュリティ一面も強化されました。またMac App Storeはデザインが大幅に刷新され、iOSのApp Storeのようにエディトリアルが表示されるようになりました。

その中でも**ダークモード**はMojaveの特徴的な新機能のひとつです。これまではFinal Cut ProやLogic Proなど主にプロ向けのMacアプリに採用されていたダークモードが、システムレベルでサポートされました。本章では、今後iOSにも搭載される可能性のあるダークモードの実装方法を解説します。

またMojaveでは従来のAppKitに加えて、UIKitでmacOSアプリを開発できるようになりました。まだ「Phase 1」と呼ばれる段階で、Appleのみが利用できる機能ですが、2019年にはすべての開発者が利用できるようになります。iOS開発者が既存のiOSアプリをmacOSに対応させるために、今から何ができるかを考察します。

9.2 ダークモード

Mojaveではユーザーがシステム環境設定でダークモードを選ぶことができます。Xcode 9でビルドされたmacOSアプリはユーザー設定にかかわらずライトモード（非ダークモードである通常モード）で表示されるため、アップデートが必要になります。アプリをダークモード表示に対応させるには、Xcode 10を使ってビルドします。

既存のmacOSアプリがMojaveではうまく動作しない場合、ダークモードには対応せずにとりあえずバグ修正をリリースしたい、という場合もあるでしょう。

このような時はXcode 9を使ってバグを修正するか、もしくはXcode 10を使う場合はInfo.plistファイルにNSRequiresAquaSystemAppearanceキーを追加し、値をtrueにすることで一時的にア

プリをダークモード非対応にできます。

またアプリ起動時に `NSApp.appearance` を指定すると、ユーザー設定にかかわらずアプリをライトモード（もしくはダークモード）で表示できます。たとえばプロ向けの macOS アプリを開発していて、アプリを常にダークモードで表示させたい、などの場合に利用できます。

```
// アプリを常にライトモードで表示
NSApp.appearance = NSAppearance(named: .aqua)

// アプリを常にダークモードで表示
NSApp.appearance = NSAppearance(named: .darkAqua)
```

ただしアプリをダークモードで表示させても、すべてのUIがダークモードで表示されるわけではありません。システムで定義された色や標準コントロール以外はダークモードには対応しておらず、またアプリで使われている画像もダークモード版を用意する必要があります。さらにダークモードの細かいエフェクトに対応させる場合にも作業が必要となります。

9.2.1 色のダークモード対応

AppKitでは表9.1のように、さまざまな色が**システムカラー**として `NSColor` クラスに定義されています。

● 表 9.1 `NSColor` に定義されたシステムカラー（抜粋）

labelColor	secondaryLabelColor
textColor	placeholderTextColor
selectedTextColor	textBackgroundColor
linkColor	separatorColor
controlAccentColor	controlColor
controlBackgroundColor	controlTextColor
disabledControlTextColor	currentControlTint
selectedControlColor	selectedControlTextColor
windowBackgroundColor	windowFrameTextColor
systemBlue	systemBrown
systemGray	systemGreen
systemOrange	systemPink

システムカラーはダークモードに対応しており、たとえば `NSColor.labelColor` はライトモードでは黒を返しますが、ダークモードでは白となります。また `NSColor.systemBlue` はライトモードではカラーコード `#297cf6` を返しますが、ダークモードでは `#3186f7` となり、若干薄い青になります。このよ

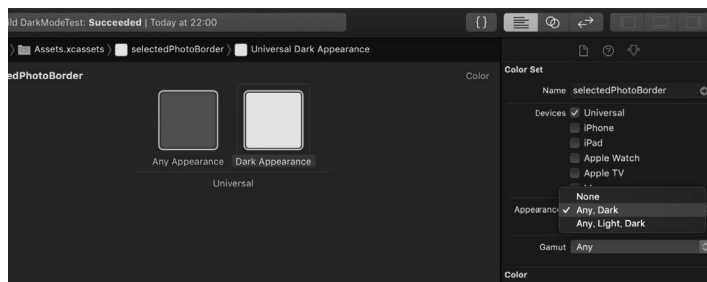
うに、できる限りシステムカラーを使うことで簡単にダークモードに対応できます。

また Mojave では図 9.1 のようにシステム環境設定でアクセントカラーを指定できるようになりました。この色は `NSColor.controlAccentColor` を使って取得できます。アプリで独自の選択（ハイライト）の色を定義している場合は、アクセントカラーを使うように変更するとよいでしょう。



● 図 9.1 システム環境設定でアクセントカラーを指定できる

アプリのブランディングなど、システムカラーに存在しない色を使う場合は**アセットカタログ**で管理することが奨励されています。Xcode 9 からアセットカタログに色を追加できるようになりましたが、Xcode 10 ではダークモード用の色を追加できるようになりました。図 9.2 のようにアセットカタログで色を選択すると、右側のインスペクタに **Appearances** プロパティが表示され、ポップアップ・メニューで **Any, Dark**（もしくは **Any, Light, Dark**）を選択するとダークモード用の色を追加できます。



● 図 9.2 アセットカタログにダークモード用の色を追加する

アセットカタログの色は次のコードで読み込むことができます。

```
let color = NSColor(named: "myColor")
```

アセットカタログを使わない場合でもコードで対応できます。`NSAppearance.current` の値が `.darkAqua` の場合にダークモード用の色を指定します。

第 10 章

tvOS 12の新機能

所 友太／@tokorom

10.1 はじめに

本章では、tvOS 12のアップデート内容をひとつおり解説します。

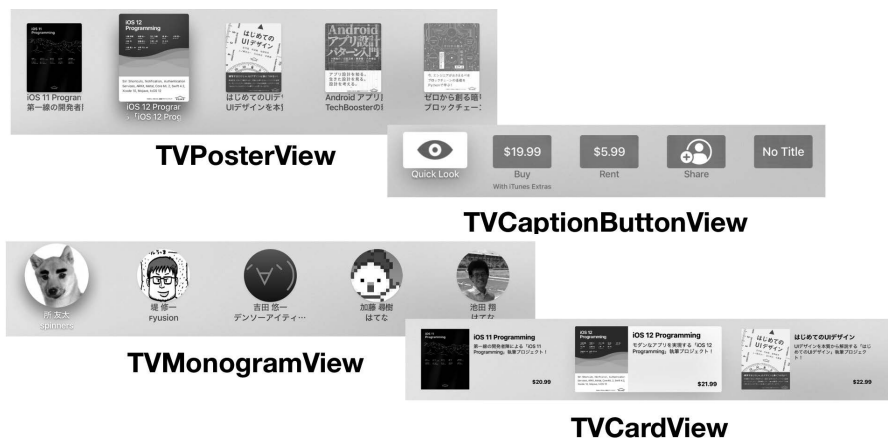
- 「10.2 TVUIKit」では、tvOS 12の目玉となるTVUIKitについて解説します
- 「10.3 UIKitのtvOS向けアップデート」では、UIKitに加わったtvOS用のアップデートを解説します
- 「10.4 Password AutoFill」では、iOSと連携して実現できるようになったtvOSアプリでのPassword AutoFillについて解説します
- 「10.5 フォーカス周りのアップデート」では、フォーカス関連のアップデートについて解説します

10.2 TVUIKit

TVUIKitには次のクラスとプロトコルがあります。本節では、TVUIKitの存在意義について説明したあと、これら全てについて詳細な解説をしていきます。

- 「10.2.2 TVPosterView クラス」
- 「10.2.3 TVCaptionButtonView クラス」
- 「10.2.4 TVCardView クラス」
- 「10.2.5 TVMonogramView クラス」
- 「10.2.6 TVLockupView クラス」
- 「10.2.7 TVLockupHeaderFooterView クラス」
- 「10.2.8 TVLockupViewComponent プロトコル」
- 「10.2.9 TVDigitEntryViewController クラス」

10.2.1 TVUIKit の存在意義



● 図 10.1 TVUIKit

WWDC 2018でTVUIKitが発表されました。TVUIKitは、図10.1のような具体的なUIパーツを提供する、tvOSアプリ用のフレームワークです。ポスター形式のパーツから顔写真を円形に表示するためのパーツまで、従来のUIKitではありえなかった末端のUIパーツを提供しています。「こんな末端のパーツを提供する必要があるのだろうか？」というのが私の第一印象でした。しかしTVUIKitについて深く調べていくうちに、TVUIKitが多くのtvOSアプリ開発者とユーザーにとって有用なフレームワークになると確信しました。

たとえばAppleが提供するtvOSアプリには、Siri Remoteのタッチサーフェイスの操作に合わせて動く、細やかなフォーカスアニメーションが実装されています（図10.2）。フォーカスという概念を扱うtvOSアプリでは、こういったインタラクションが重要であることはよく知られています。



● 図 10.2 細やかなフォーカスアニメーション

ただし、これと同じインタラクションをサードパーティ製アプリで実装するとなると、限られたケース¹⁾をのぞいて難易度が高く、著名なサードパーティ製アプリでも対応しきれていないというのが現状です。

こういった経緯で、tvOS アプリ独特の事情をふまえた tvOS 専用の TVUIKit が新設されたものと考えます。TVUIKit の UI パーツは、これら細やかなインタラクションを元よりサポートしており、最小限のコードで利用できます。また、このインタラクションをカスタマイズするためのインターフェースも多く備えています。

TVUIKit の登場により Apple 純正アプリと同等の UI を実装する敷居が下がり、tvOS アプリの UI のクオリティは着実に向上していくことでしょう。

注 1) UIImageView だけでは `adjustsImageWhenAncestorFocused` プロパティなどこれを簡単に実現する機能がありました

iOS 12 Programming

加藤 尋樹
@cockscomb

佐藤 紘一
@justin999_

石川 洋資
@_ishkawa

堤 修一
@shu223

吉田 悠一
@sonson_twit

池田 翔
@ikesyo

佐藤 剛士
@hatakenokakashi

大榎 一司
@tamadeveloper

所 友太
@tokorom

編集
横田 孝次郎
@macneko_ayu

編集
永野 哲久
@7gano



『iOS12 Programming』
電子版: ¥3,200 製本版: ¥3,500

購入して続きを読む

