Как быстро сократить вес приложения: переносим картинки и строки локализации на сервер

Евтухов Александр
Teamlead iOS

открытие

Как быстро сбросить вес без регистрации и смс

Евтухов Александр
Teamlead iOS

открытие

План доклада

- Простые способы оптимизации веса пакета приложения
  - Флаги компиляции
  - Статическая и динамическая линковка
  - Бинарные символы
- Перенос ресурсов приложения на CDN
  - XCAssets
  - Строки локализации

Статистика проекта
Отправная точка

Вес приложения – 283мб

Вес ipa файла – 168мб

Кол-во ассетов – 2500 штук

Кол-во строк локализации – 7100

Поддерживаемые языки - русский

Простые меры сокращения размера приложения

1

Флаги компиляции для продакшен сборки проекта

Debug

SWIFT_COMPILATION_MODE = singlefile // Incremental

SWIFT_OPTIMIZATION_LEVEL = -Onone // None

GCC_OPTIMIZATION_LEVEL:= 0 // None

Production

SWIFT_COMPILATION_MODE = wholemodule // Whole module

SWIFT_OPTIMIZATION_LEVEL = -O // Speed

GCC_OPTIMIZATION_LEVEL = -Os // Fastest, Smallest

Вес приложения	283 Мб
Вес ipa	168 Мб

Статическая линковка

MACH_O_TYPE = staticlib

Pros

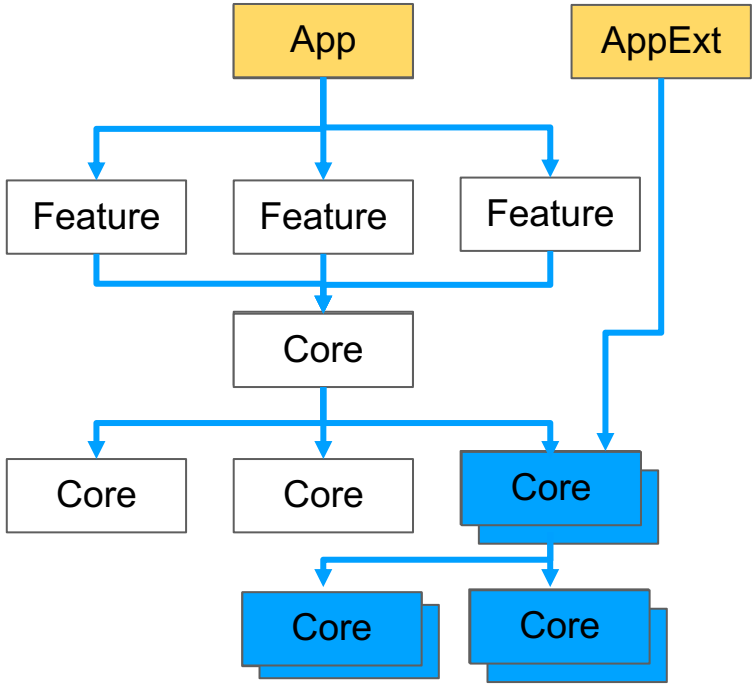
Увеличивает компилятору область видимости кода

Улучшает работу оптимизаций по удалению неиспользуемого кода

Ускоряет запуск приложения

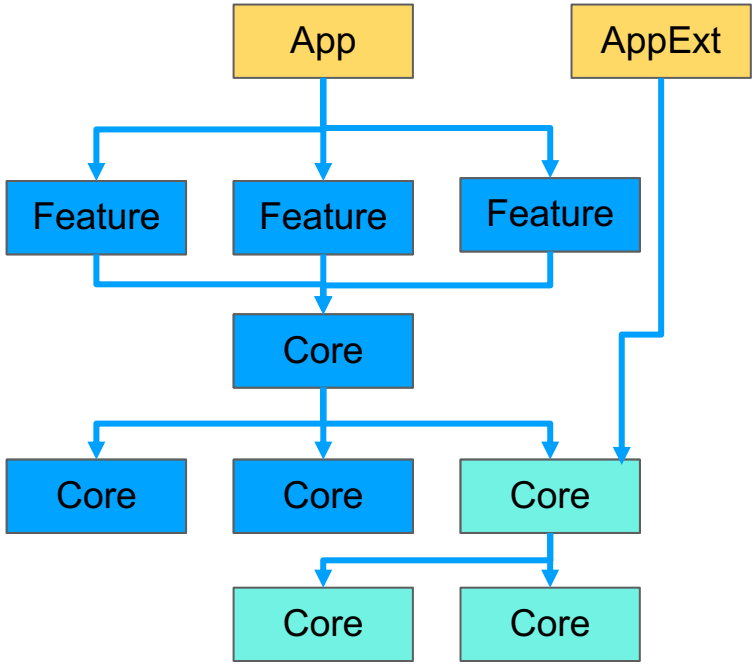
Cons

Требует повышенного внимания при работе с extensions



Таргеты которые будут дублироваться

Статическая линковка



Таргеты использующие статическую линковку

Таргеты использующие динамическую линковку

Вес приложения 283 –> 239 -15%
Вес ipa 168 –> 158 -5%

Флаги компиляции для продакшен сборки проекта

Production

SWIFT_COMPILATION_MODE = wholemodule // Whole module

SWIFT_OPTIMIZATION_LEVEL = -O -> -Osize // Speed -> Size

GCC_OPTIMIZATION_LEVEL = -Os // Fastest, Smallest

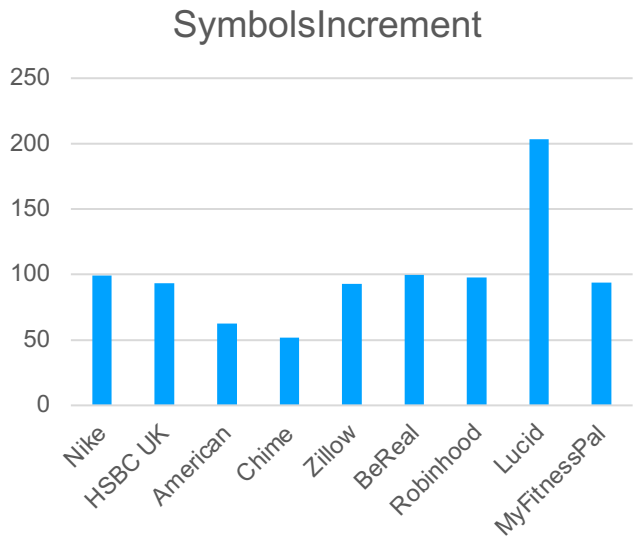
Дополнительные флаги

LLVM_LTO = -YES // Monolithic Link-Time Optimization

Вес приложения 239 –> 235 -1.4%
Вес ipa 158 –> 157 -0.6%

Binary Symbols при переходе на Xcode 14

- С Xcode 14 больше не поддерживается биткод
- Одной из функций биткода было удаление бинарный символов из приложения



Type	Name	Size
Modified	CommerceUI.framework/CommerceUI/DYLD.String Table	↑ 14.6 MB
Modified	ProductKit.framework/ProductKit/DYLD.String Table	↑ 11.6 MB
Modified	AirshipKit.framework/AirshipKit/DYLD.String Table	↑ 6.1 MB
Modified	SharedUI.framework/SharedUI/DYLD.String Table	↑ 5.7 MB
Modified	SharedCore.framework/SharedCore/DYLD.String Table	↑ 4.7 MB
Modified	Adyen.framework/Adyen/DYLD.String Table	↑ 3.9 MB

* материалы взяты из статьи EmergeTools

Вариант 1
Настройки таргетов

Флаги требующие изменения

DEPLOYMENT_POSTPROCESSING = YES // Deployment Postprocessing

STRIP_INSTALLED_PRODUCT = YES // Strip Linked Product

STRIPFLAGS = -rSTx // Additional Strip Flags

Прочие Strip флаги без изменений

SWIFT_ COPY_PHASE_STRIP = NO

STRIP_STYLE = All Symbols

STRIP_SWIFT_SYMBOLS = YES

DEAD_CODE_STRIPPING = YES

Вариант 2
Запуск скрипта после сборки приложения

Ссылка на скрипт

Статья EmergeTools

Вес приложения 235 –> 234 -0.3%
Вес ipa 157 –> 156 -0.6%

Затягивание OpenSource исходными кодами

Dev сборки у разработчиков

Используют весь OpenSource собранный в XCFrameworks

Сборки идут быстрее

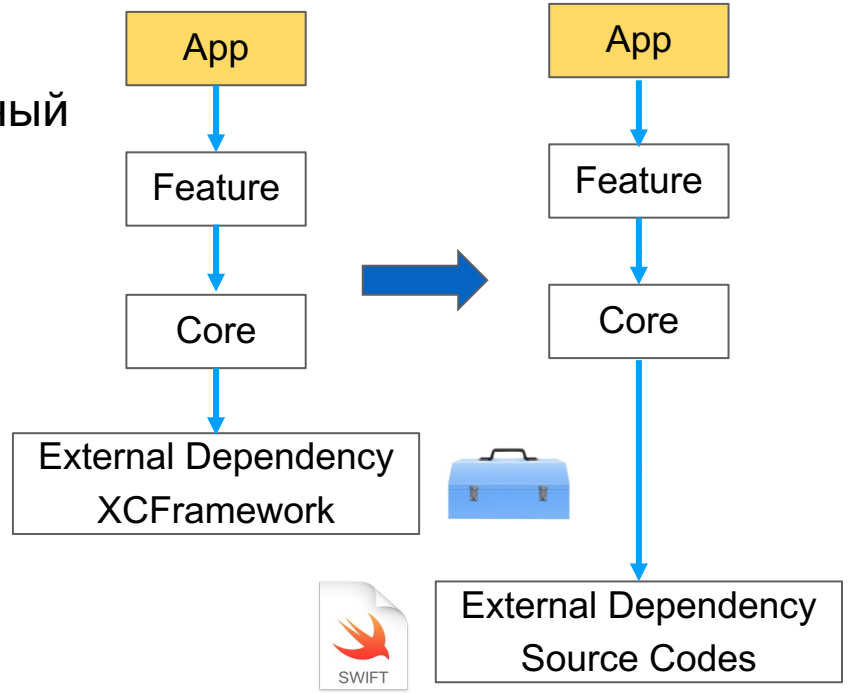
Используется динамическая линковка

Production сборки

Весь OpenSource исходными кодами

Используется статическая линковка

Вес приложения 234 –> 232 -0.7%
Вес ipa 156 –> 155 -0.7%



Установка более агрессивных значений в настройках сборки проекта

Возможные варианты

GCC_OPTIMIZATION_LEVEL = -Oz // Smallest, Aggressive

GCC_THREADSAFE_STATICS = False // Возможны краши

SWIFT_REFLECTION_METADATA_LEVEL = none // Ломает SwiftUI

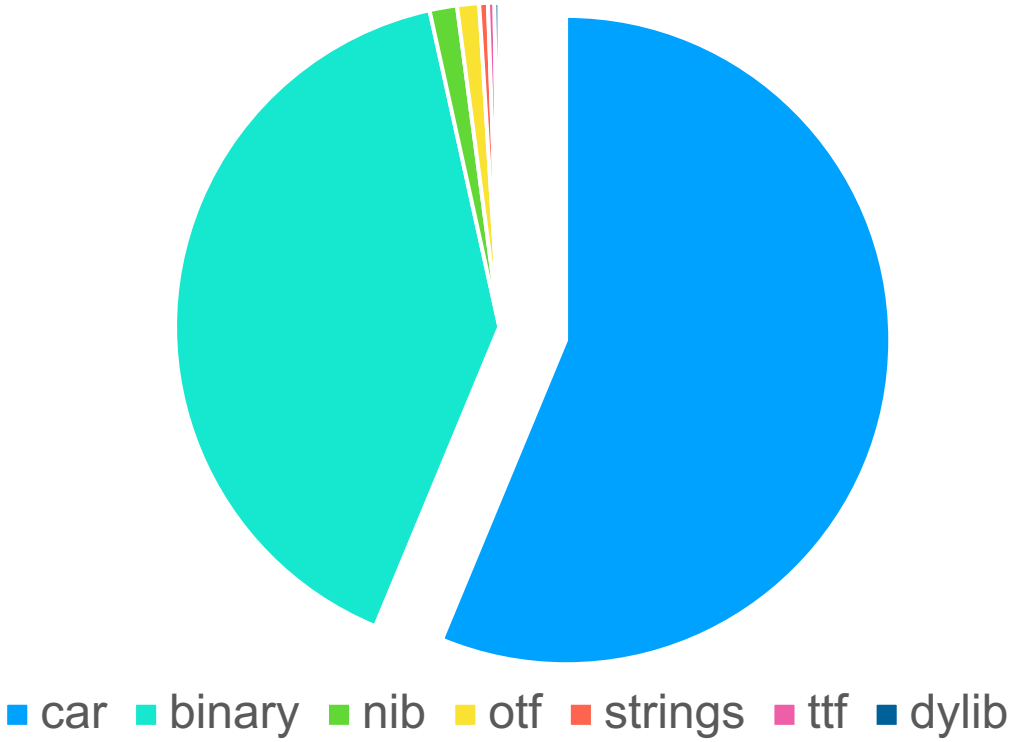
DEV_MENTAL_HEALTH_SUPPORT = MnogoValedola

-experimental-hermetic-seal-at-link

Все эти флаги практически не дадут никакого весомого сжатия ваших бинарных файлов. При этом их использование уже начинает представлять определённую опасность

Распределение веса по пакету приложения после оптимизаций

Тип файла	Вес в Мб
car	130,8
binary	93,7
nib	3,2
otf	2,6
strings	1,0
ttf	0,7
dylib	0,6



Перенос asset-ов приложения на backend

2

Несколько дисклеймеров

- Мы не будем рассматривать как построить систему с экспортом картинок из фигмы в которой есть некоторая дизайн система.
- Главный критерий – минимум рефакторинга
- Используется SwiftGen или его аналог
- Мы не будем рассматривать работу с векторной графикой.
- Это не гайд по построению эталонной дизайн системы

Проектирование интерфейса библиотеки для загрузки картинок

```
protocol ImageDownloaderProtocol {
    /// Проверяет наличие картинки в кеше и возвращает его тип
    /// .none / .disk / .inMemory
    func getCacheType(for url: URL) -> CacheType

    /// Возвращает картинку из inmemory cache
    func obtainMemoryCachedImage(for url: URL) -> Result<UIImage, Error>

    /// Возвращает картинку с диска/
    func obtainDiskCachedImage(
        for url: URL,
        completion: @escaping (Result<UIImage, Error>)->()
    ) -> Operation

    /// Грузит картинку из сети
    func obtainImageFromNetwork(
        by url: URL,
        completion: @escaping (Result<UIImage, Error>)->()
    ) -> Operation
}
```

Экстеншен для работы с UIImage

В приложении должен присутствовать extension к UIImage через который централизованно выполняются преобразования над картинками.

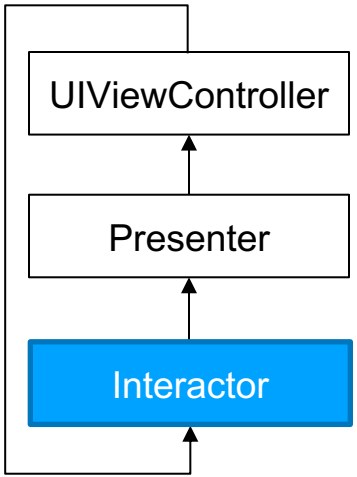
```
protocol UIImageModifier {
    /// Возвращает картинку затинтованую цветом
    func tinted(with color: UIColor) -> UIImage

    /// Возвращает картинку с измененным размером
    func resized(to size: CGSize) -> UIImage
}

extension UIImage: UIImageModifier {
    // ...
}
```

Класс для получения URL и описания картинки из path

```
@frozen public enum SwiftGen {
    public enum Provide {
        public static let test = ImageAsset(name: "Provide/test")
    }
    //....
    public struct ImageAsset {
        let bundleID = "ru.bundle.id"
        let name: String
        var image: UIImage { ... }
    }
}
```

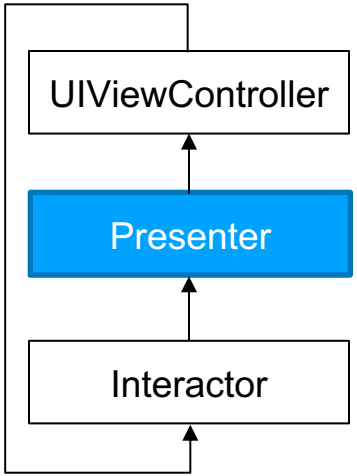
```
protocol ImageUrlFactoryProtocol {
    /// Возвращает MetaInfo (URL) для скачивания картинки с CDN компании
    func getImageDescription(for path: String, bundleID: String) -> ImageDescription?
}
```

Флоу работы с картинками Interactor



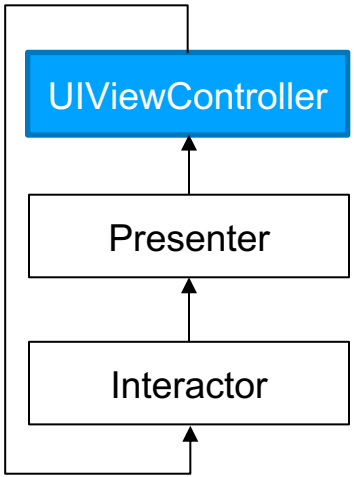

```
struct Interactor {
    var presenter: Presenter

    func doSomeWork() {
        makeRequest { success in
            presenter.presentImage(isFailImage: success)
        }
    }
}
```

Флоу работы с картинками Presenter




```
struct Presenter {
    weak var viewController: ViewController?

    func presentImage(isFailImage: Bool) {
        let color: UIColor = isFailImage ? .green : .red
        let image = SwiftGen.Provide.test.image
        let tintedImage = image.tinted(with: color)
        let viewModel = ViewController.ViewModel(
            image: tintedImage
        )
        viewController?.configure(with: viewModel)
    }
}
```

Флоу работы с картинками ViewController

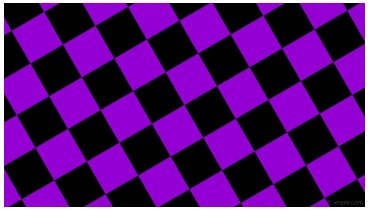



```
class ViewController: UIViewController {
    var imageView: OImageView = OImageView(frame: .zero)

    struct ViewModel {
        var image: UIImage
    }

    func configure(with viewModel: ViewModel) {
        self.imageView.image = viewModel.image
    }
```

Жизненный цикл серверной картинки

ServerImage объект
Template UIImage+URL

URL = https://…./.image



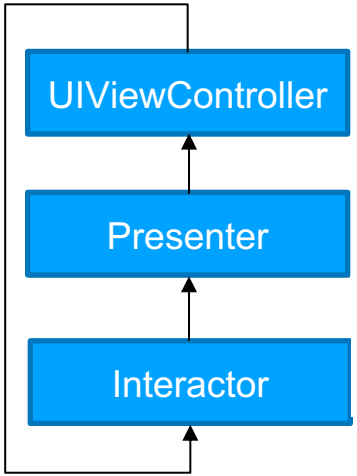
Утилита загрузки картинок ImageLoader

UIImage

- не содержит графики
- Есть URL для загрузки
- Запрещены модификации
  - tinting
  - resizing

- содержит графику
- URL удален
- Разрешены модификации

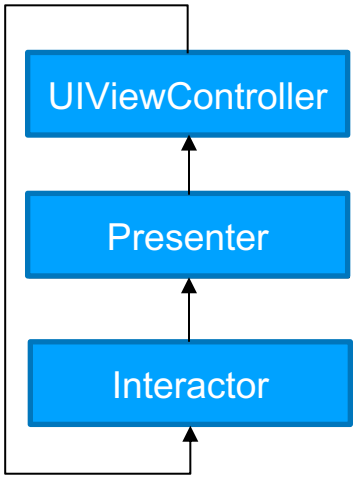
Флоу работы с картинками




```
let image = SwiftGen.Provide.test.image

let tintedImage = image.tinted(with: color)

let viewModel = ViewController.ViewModel(image: tintedImage)

imageView.image = viewModel.image
```

Все методы синхронные
Это значит что их не получится заменить на асинхронную загрузку картинок

Определяем момент для загрузки картинки

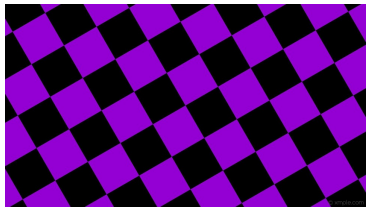



```
let image = SwiftGen.Provide.test.image          // <-- ServerImage

let tintedImage = image.tinted(with: color)      // <-- ServerImage

let viewModel = ViewModel(image: tintedImage)    // <-- ServerImage

imageView.image = viewModel.image // {           // <-- ServerImage
// Показать плейсхолдер                             <-- ServerImage
// Загружаем картинку                        UIImage <-- ServerImage
// Показываем загруженную картинку           UIImage
//}
```

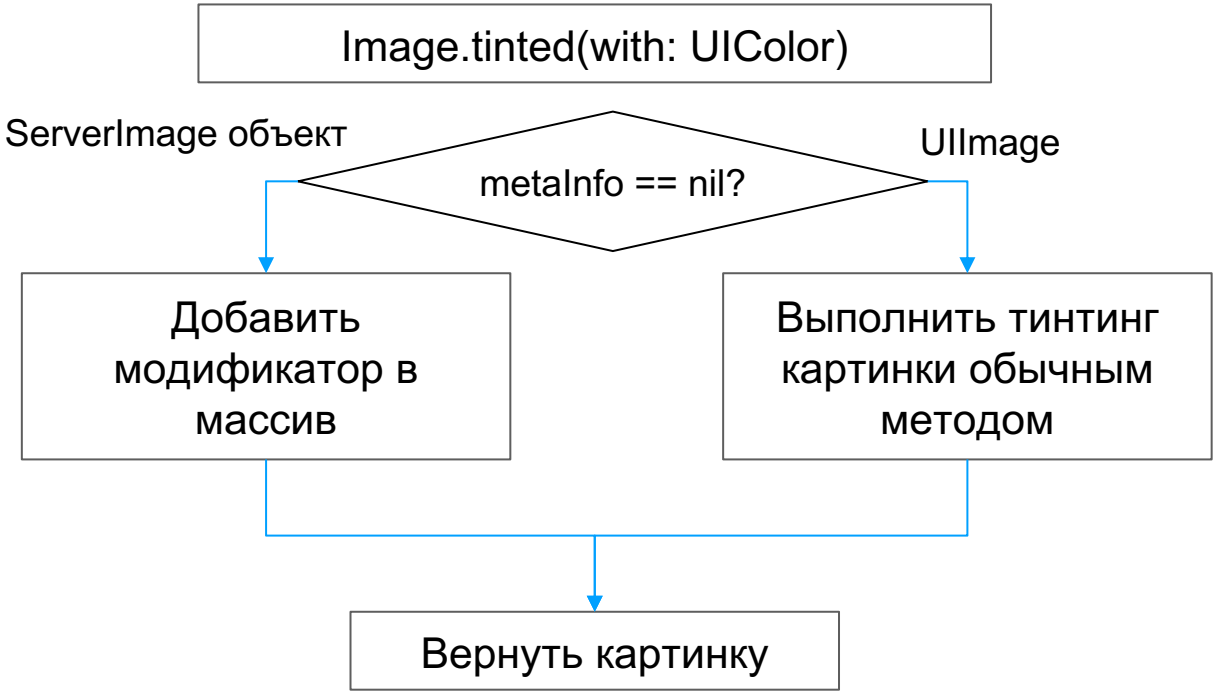
Единственное место где можно безопасно перейти к исполнению асинхронного кода – метод set у свойства image

Применение модификаторов при работе с ServerImage

ServerImage объект
URL = https://…./.image



```
modifiers = [
    .tinted(…),
    .resized(…)
]
```



Модификации базовых классов UIKit

UIImage

Необходимо добавить поля

```
var imageDescription: ImageDescription?
var modifiers: [ImageModifier]
```

UIImageView

Необходимо переопределить

```
var image: UIImage?
```

Модификации базовых классов UIKit Вариант 1

UIImage + objc_set\getAssociatedObject

UIImageView + variable swizzing

Pros

Самый простой и малоинвазивный вариант
Установка картинок в UIButton работает без доп. условий

Cons

Использование сразу 2х приемов «черной» магии
Необходимо использовать динамическую линковку

Модификации базовых классов UIKit Вариант 2

UIImage + Наследование

UIImageView + Наследование

Pros

Никаких запрещенных приемов

Cons

Нужно менять код используя Find\Replace
Нужно менять IB файлы используя regexp
Для работы UIButton потребуются дополнительные приседания

Модификации базовых классов UIKit Вариант 2

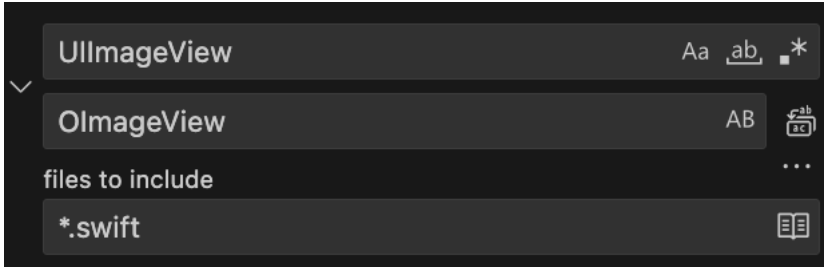
```
open class OImageView: UIImageView {
    override open var image: UIImage? {
        get { return super.image }
        set { /* some magic */ }
    }
```

```
<subviews>
    <imageView .... id="cFV-9Q-Kcp">
    </imageView>
</subviews>
```

В IB файлах осуществляется замена класса UIImageView на новый

```
<subviews>
    <imageView  id="NNU-aZ-Jpv" customClass="OImageView" customModule="TestInheritance">
    </imageView>
</subviews>
```

Модификации базовых классов UIKit Вариант 2



UIImage + Наследование

UIImageView + Наследование

Pros

Никаких запрещенных приемов

Cons

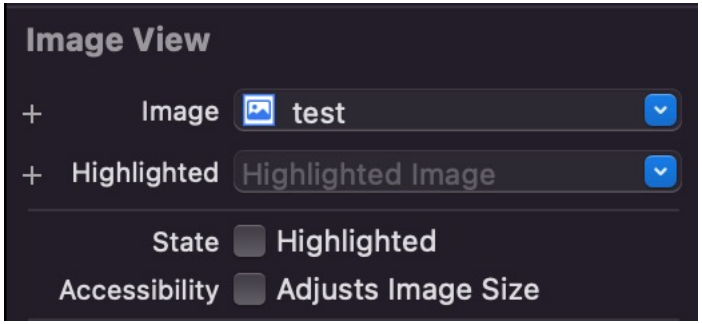
Нужно менять код используя Find\Replace
Нужно менять IB файлы используя regexp
Для работы UIButton потребуются дополнительные приседания

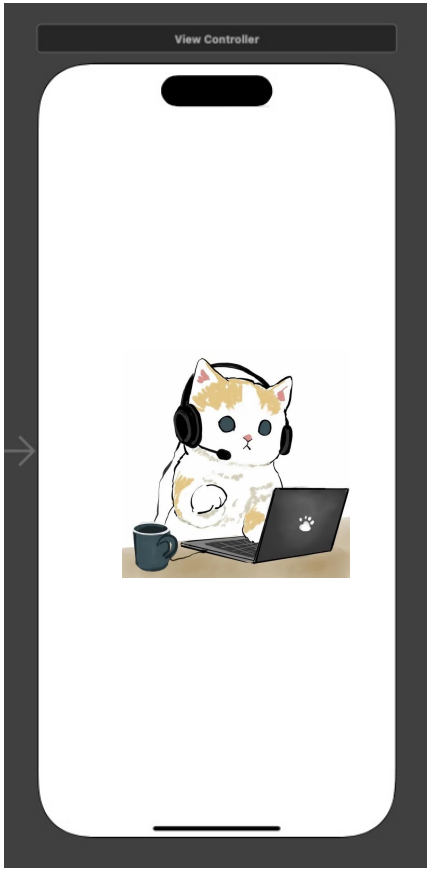
Модификации базовых классов UIKit
Как унаследовать UIImage

```
class ServerImage: UIImage {
    var imageDescription: ImageDescription?
    var modifiers: [ImageModifier] = []

    public convenience init(
        imageDescription: ImageDescription,
        modifiers: [ImageModifier]
    ) {
        self.init(cgImage: .placeholder)
        self.imageDescription = imageDescription
        self.modifiers = modifiers
    }
}
```

```
open class OImageView: UIImageView {
    override open var image: UIImage? {
        get { return super.image }
        set { /* some magic */ }
    }
```

Решение проблемы с установкой картинок из IB



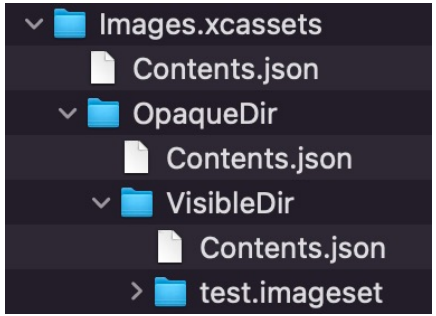


Использование IB для установки картинок при их удалении приводит к возникновению ошибки

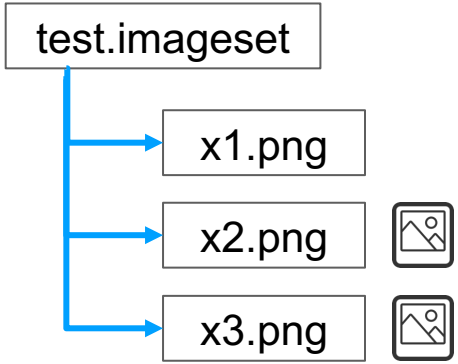
2023-10-11 22:17:06.819626+0300 TestProject[13758:11566312] Could not load the "test" image referenced from a nib in the bundle with identifier "*****"

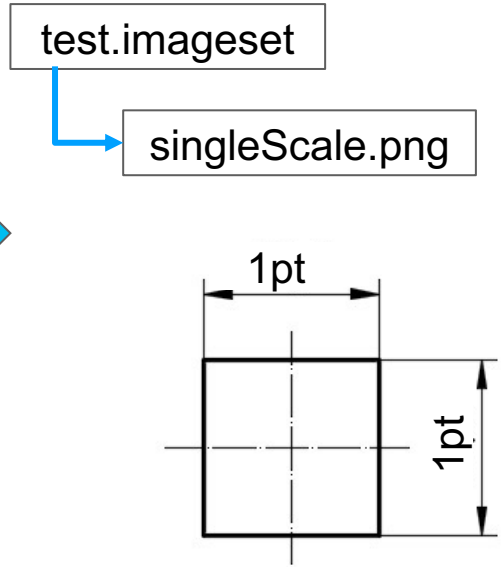
Вариант 1
Сохранение структуры XCAsset

- Структура XCAssets сохраняется в приложении
- Все картинки переводятся на singleScale
- Все ассеты заменяются пустыми изображениями
- Переопределяется конструктор UIImageView.init(coder:)



Структура XCAssets сохраняется в приложении





Вариант 1
Сохранение структуры XCAsset

- Переопределяется конструктор UIImageView.init(coder:)

```
static func createImage(from coder: NSCoder) -> ServerImage? {
    guard
        let image = (coder.decodeObject(forKey: "UIImage")) as? NSObject,
        let asset = image.value(forKey: "_imageAsset") as? NSObject,
        let path = asset.value(forKey: "_assetName") as? String,

        let bundle = asset.value(forKey: "_containingBundle") as? Bundle,
        let bundleID = bundle.bundleIdentifier
    else { return nil }
```

Вариант 1
Сохранение структуры XCAsset

- Переопределяется конструктор UIImageView.init(coder:)

```
static func createImage(from coder: NSCoder) -> ServerImage? {
        ...
        let path = asset.value(forKey: "_assetName") as? String,
        ...
        let bundleID = bundle.bundleIdentifier

        let imageDescription = ImageUrlFactory.shared.getImageDescription(
            for: path,
            bundleID: bundleID
        )!
        return ServerImage(imageDescription: imageDescription, modifiers: [])
}
```

Вариант 1
Сохранение структуры XCAsset

- Переопределяется конструктор UIImageView.init(coder:)

```
required public init?(coder: NSCoder) {
    super.init(coder: coder)
    if let coderImage = Self.createImage(from: coder) {
       self.image = coderImage
    }
}
```

Вариант 2
Перенос path в IBInspectable поле

- Структура XCAssets удаляется
- Path картинки переносится в userDefinedAttribute

```
<imageView ...>
    <userDefinedRuntimeAttributes>
        <userDefinedRuntimeAttribute type="string" keyPath="ibImageKey" value="Provide/test"/>
    </userDefinedRuntimeAttributes>
</imageView>
```

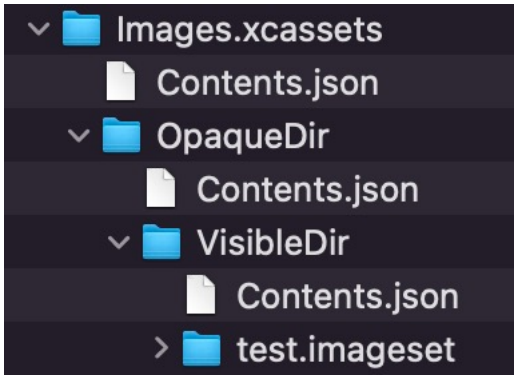
Вариант 2
Перенос path в IBInspectable поле

- Структура XCAssets удаляется
- Path картинки переносится в userDefinedAttribute

```
@IBInspectable
var ibImageKey: String? {
    didSet {
        guard
            let ibImageKey = ibImageKey
        else { return }
        let imageDescription = ImageUrlFactory.shared.getImageDescription(
            for: ibImageKey,
            bundleID: Bundle.main.bundleIdentifier!
        )!
        let oImage = ServerImage(imageDescription: imageDescription, modifiers: [])
        self.image = oImage
    }
```

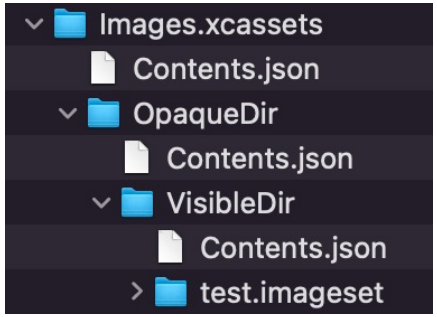
Скрипт для подготовки xcasset к выгрузке на бекенд
Задачи

- Получение данных для класса ImageURLFactory
- Подготовка пакета к выгрузке на CDN
- Извлечение полезных метаданных из imageset



Скрипт для подготовки xcasset к выгрузке на бекенд

Рекурсивный поиск imageset

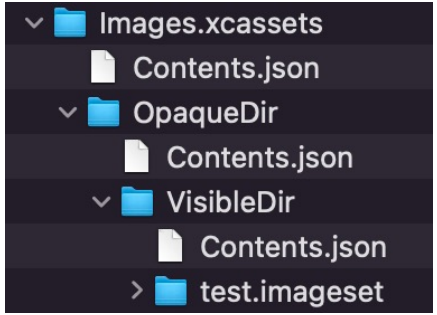


Скрипт для подготовки xcasset к выгрузке на бекенд

Рекурсивный поиск imageset

↓

Формирование path на базе анализа Contents.json



module = bundleIdentifier
path = VisibleDir/test
cdnPath = Images/OpaqueDir/VisibleDir/test

```
{
  ...
  "properties" : {
    "provides-namespace" : true
  }
}
```

Скрипт для подготовки xcasset к выгрузке на бекенд

Рекурсивный поиск imageset

↓

Формирование path на базе анализа Contents.json

↓

Проверка использования asset в проекте

↓

Удаление лишних ассетов

module = ModuleX
path = VisibleDir/test
cdnPath = Images/OpaqueDir/VisibleDir/test

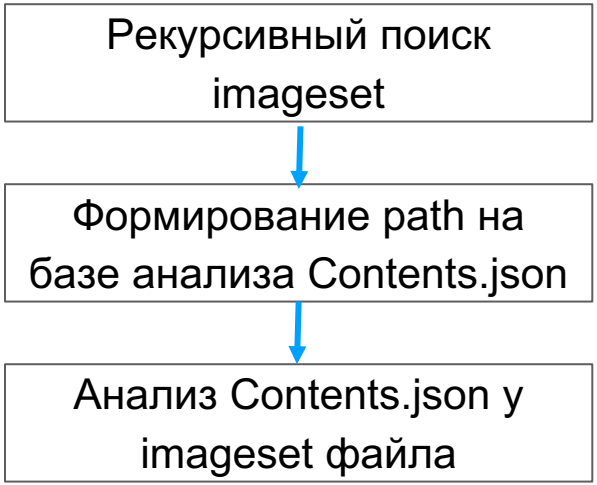
path = VisibleDir/test

Images.VisibleDir.test

Поиск строки в файлах проекта

Поиск path в IB файлах проекта

Скрипт для подготовки xcasset к выгрузке на бекенд



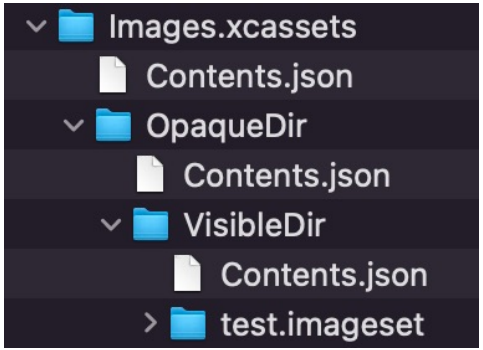

```
{
  "images": [
    {
      "idiom": "universal",
      "scale": "1x"
    },
    {
      "filename": "qrcode-6.png",
      "idiom": "universal",
      "scale": "2x"
    },
    {
      "idiom": "universal",
      "scale": "3x"
    }
  ],
  "properties": {
    "template-rendering-intent": "template"
  }
}
```

path = VisibleDir/test
cdnPath = Images/OpaqueDir/VisibleDir/test/qrcode-6.png
renderAsTemplate = true

Скрипт для подготовки xcasset к выгрузке на бекенд

- Файлы копируются в новую директорию
- Удаляются лишние расширения

path = VisibleDir/test
cdnPath = Images/OpaqueDir/VisibleDir/test/qrcode-6.png
renderAsTemplate = true



Скрипт для подготовки xcasset к выгрузке на бекенд

Для хранения метаданных о картинке будем использовать простую структуру

- path
- serverPath – путь к картинке на сервере
- isTemplateRenderMode – используется ли рендеринг как шаблон

```
public struct ImageDescription: Codable {
    public var path: String
    public var serverPath: String
    public var isTemplateRenderMode: Bool
}
```

Скрипт для подготовки xcasset к выгрузке на бекенд

Структуры записываются в генерируемый swift файл содержащий 2 вложенных словаря

1й ключ – bundleID модуля
2й ключ - path ассета

```
var container: [String: [String: ImageDescription]] = [
    "ru.moduleA.identifier": [
        "Provide/test": ImageDescription(...)
    ]
]
```

Еще раз разберем весь пусть работы с картинкой

UIView

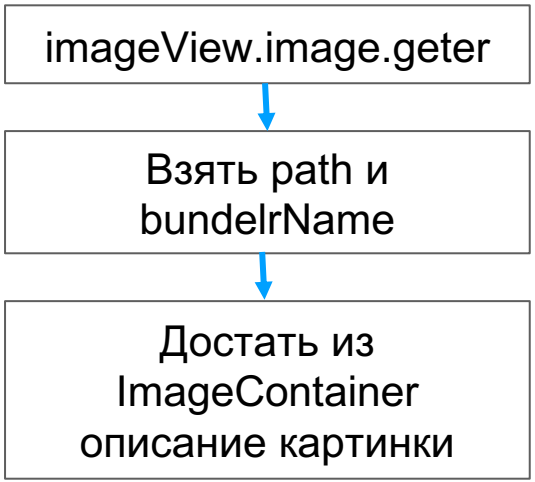
UIViewController

Presenter

Interactor

```
let image = SwiftGen.Provide.test.image          // <-- ServerImage

let tintedImage = image.tinted(with: color)      // <-- ServerImage

let viewModel = ViewModel(image: tintedImage)    // <-- ServerImage

imageView.image = viewModel.image // {           // <-- ServerImage
// Показать плейсхолдер                             <-- ServerImage
// Загружаем картинку                        UIImage <-- ServerImage
// Показываем загруженную картинку           UIImage
//}
```

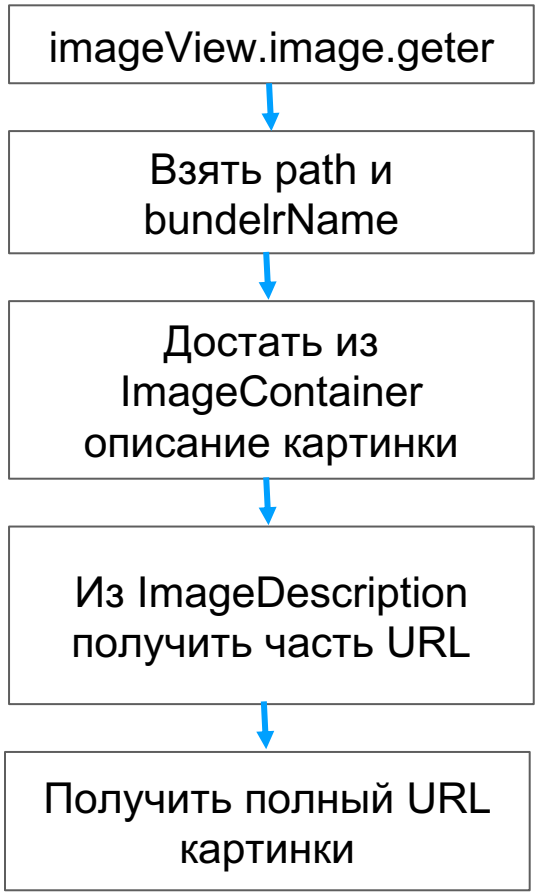
SwiftGenFile




```swift
@frozen public enum SwiftGenServer {
    public enum Provide {
        public static let test = ImageAsset(name: "Provide/test")
    }
    //....
    public struct ImageAsset {
        let name: String
        let bundleID = "ru.moduleA.identifier"
    }
}
```

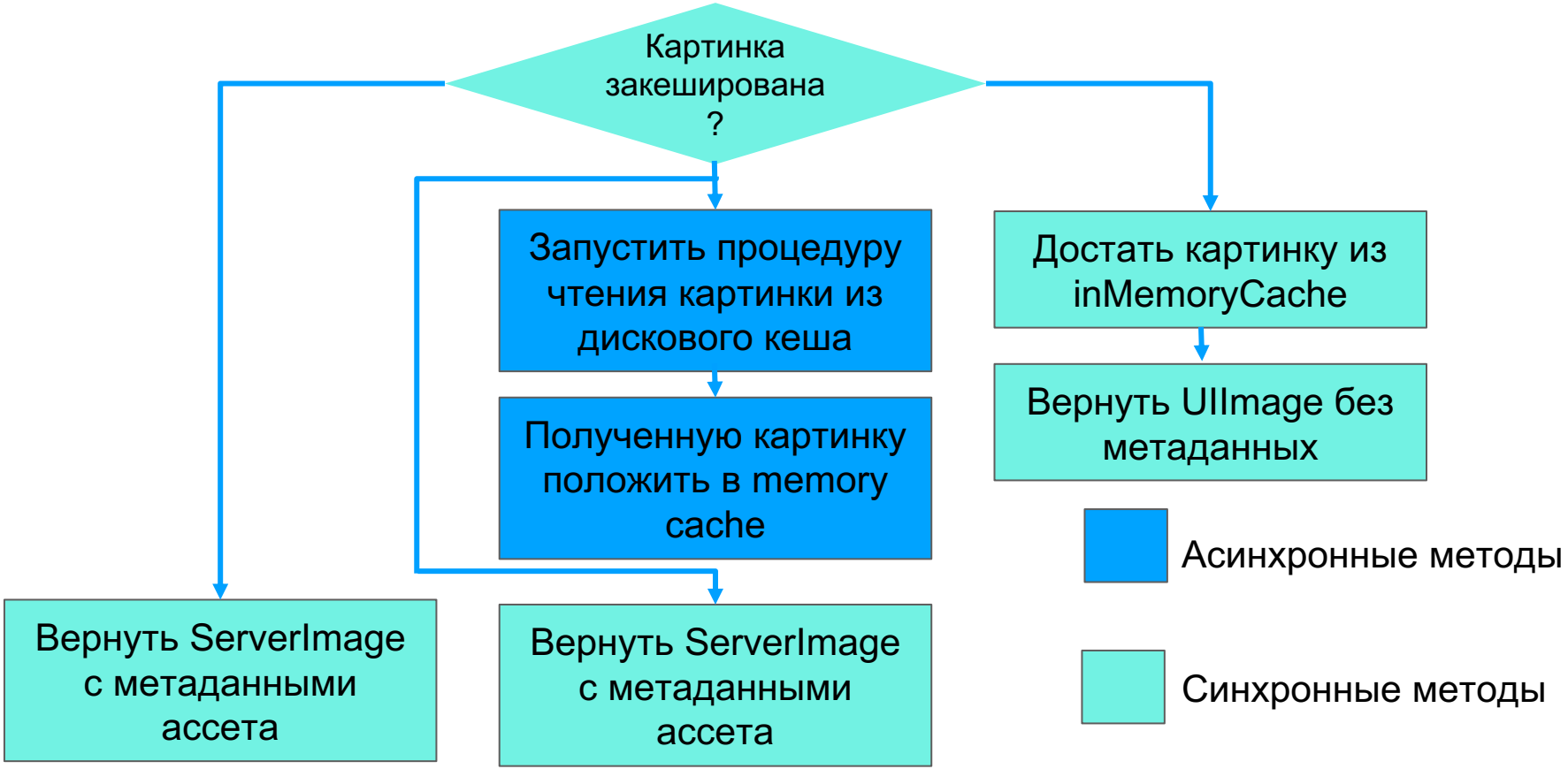
```swift
var container: [String: [String: ImageDescription]] = [
    "ru.moduleA.identifier": [
        "Provide/test": ImageDescription(...)
    ]
]
```

SwiftGenFile




```swift
extension ImageDescription {
    var url: URL {
        let baseURL = "https://cdn.company.ru/"
        let cdnPart = self.serverPath // bundleID/Provide/test/test.png
        let fullPath = baseURL + cdnPart
        // https://cdn.company.ru/bundleID/Provide/test/test.png
        return URL(string: fullPath)!
    }
}
```

SwiftGenFile



Следующий шаг обработки картинки

UIView

UIViewController

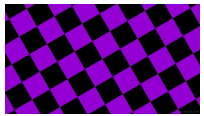
Presenter

Interactor

```
let image = SwiftGen.Provide.test.image          // <-- ServerImage

let tintedImage = image.tinted(with: color)      // <-- ServerImage

let viewModel = ViewModel(image: tintedImage)    // <-- ServerImage

imageView.image = viewModel.image // {           // <-- ServerImage
// Показать плейсхолдер                             <-- ServerImage
// Загружаем картинку                        UIImage <-- ServerImage
// Показываем загруженную картинку           UIImage
//}
```

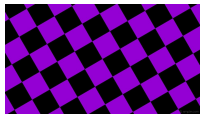
Модификация картинки ServerImage

ServerImage объект



modifiers = [ ]

ServerImage объект



modifiers = [
.tinted(…),
]

tint(with color: UIColor)

ServerImage

self as? ServerImage

UIImage

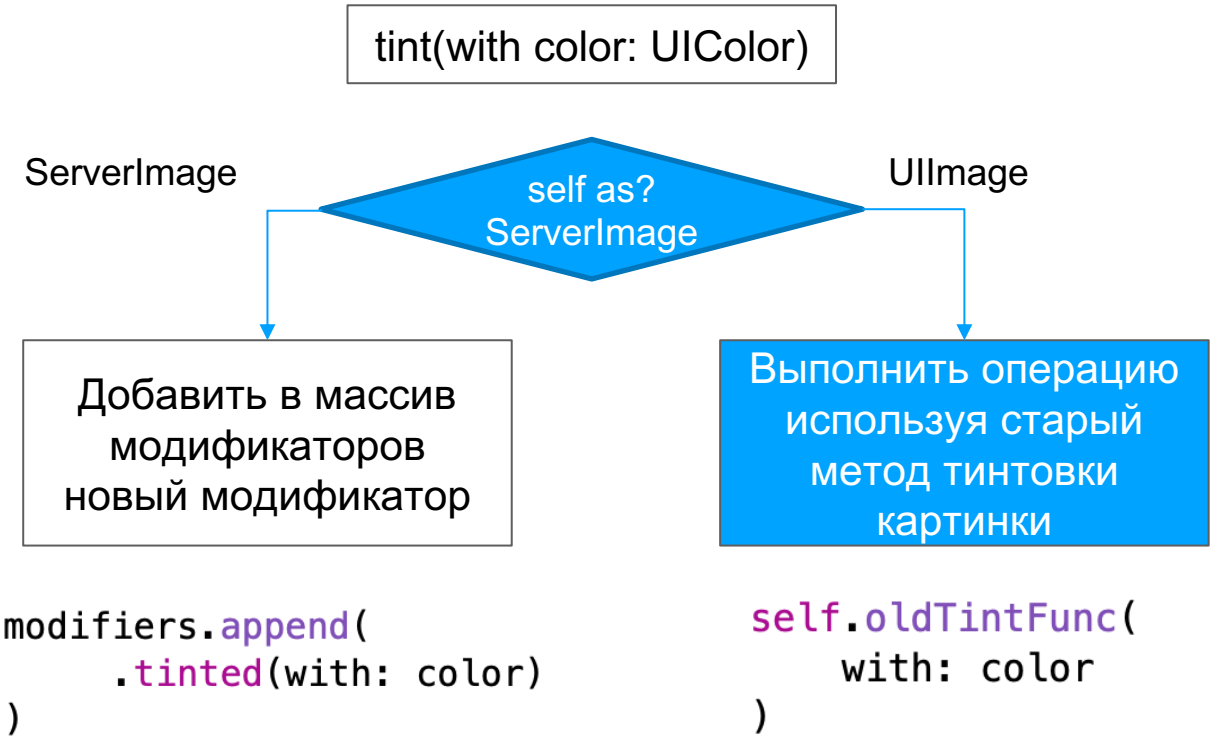
Добавить в массив модификаторов новый модификатор

Выполнить операцию используя старый метод тинтовки картинки

```
modifiers.append(
    .tinted(with: color)
)
```

```
self.oldTintFunc(
    with: color
)
```

Модификация картинки ServerImage




```
modifiers.append(
    .tinted(with: color)
)
```

```
self.oldTintFunc(
    with: color
)
```

Следующий шаг обработки картинки

UIView

UIViewController

Presenter

Interactor

```
let image = SwiftGen.Provide.test.image          // <-- ServerImage

let tintedImage = image.tinted(with: color)      // <-- ServerImage

let viewModel = ViewModel(image: tintedImage)    // <-- ServerImage

imageView.image = viewModel.image // {           // <-- ServerImage
// Показать плейсхолдер                             <-- ServerImage
// Загружаем картинку                        UIImage <-- ServerImage
// Показываем загруженную картинку           UIImage
//}
```

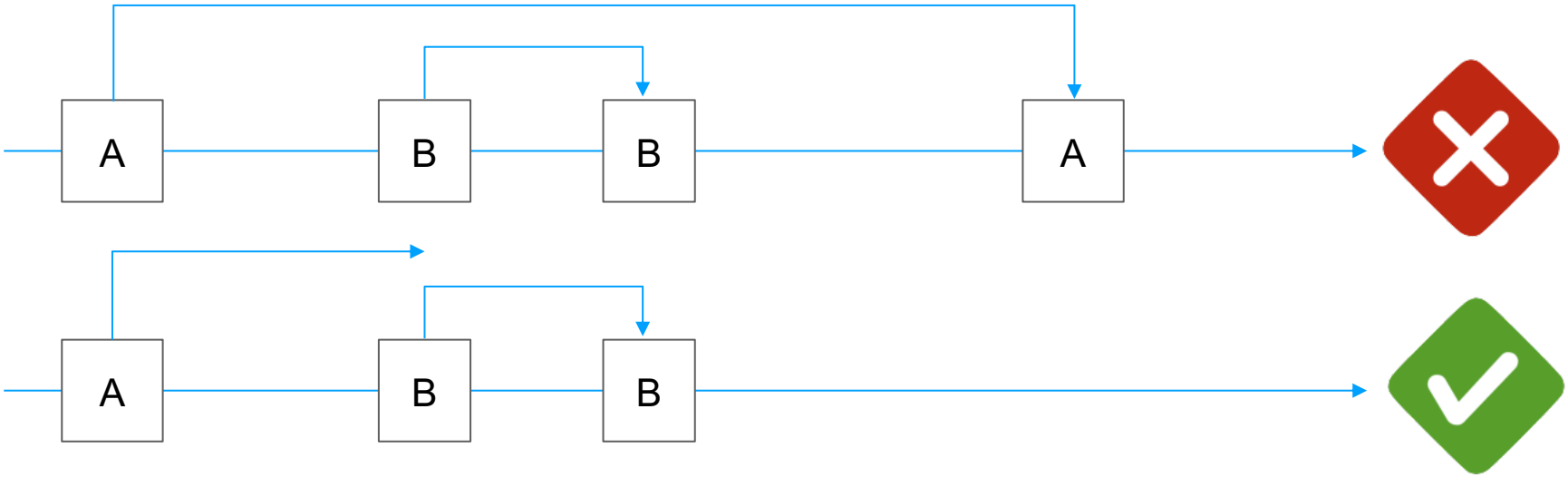
Установка картинки в OImageView

```
override open var image: UIImage? {
        get {
            return super.image
        }

        set {
            cancelCurrentOperation()
            guard let newValue = newValue else {
                super.image = nil
                return
            }
            if let serverImage = newValue as? ServerImage {
                setServerImage(serverImage)
            } else {
                super.image = newValue
            }
        }
    }
```

Поддержка отмены заданий на загрузку картинок

Возможны задержки (непоследовательность) в загрузке картинок по следующим причинам:

- Одна картинка может быть тяжелее другой
- Или могут измениться условия мобильного интернета



Установка картинки в OImageView

```
override open var image: UIImage? {
        get {
            return super.image
        }

        set {
            cancelCurrentOperation()
            guard let newValue = newValue else {
                super.image = nil
                return
            }
            if let serverImage = newValue as? ServerImage {
                setServerImage(serverImage)
            } else {
                super.image = newValue
            }
        }
    }
```

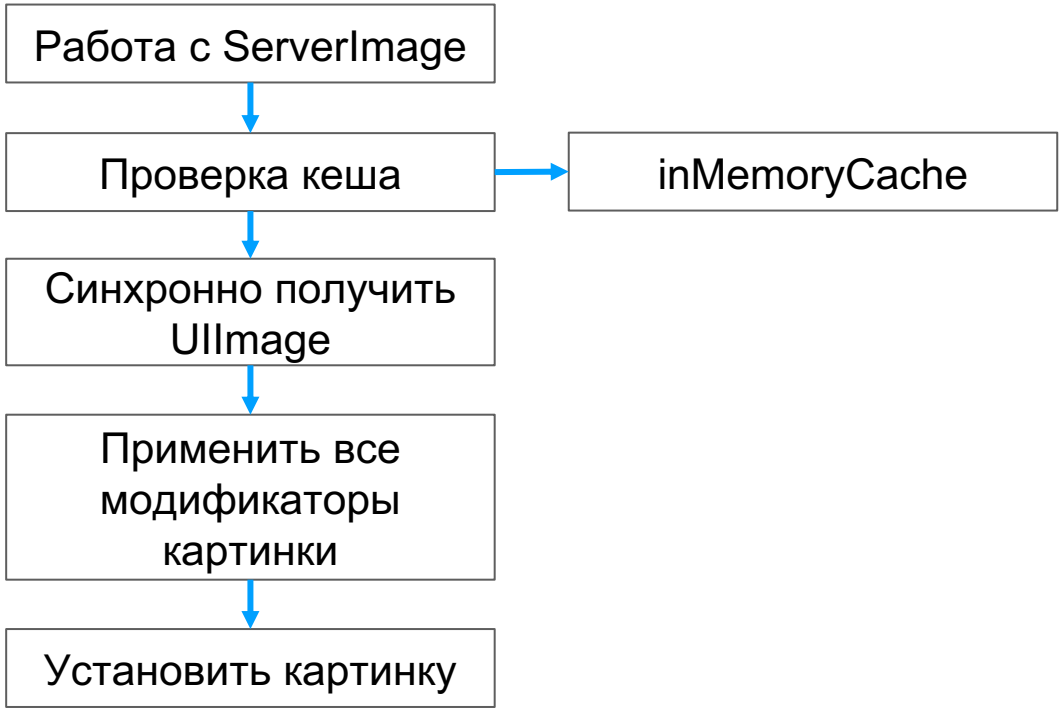
Установка картинки в OImageView

```
override open var image: UIImage? {
        get {
            return super.image
        }

        set {
            cancelCurrentOperation()
            guard let newValue = newValue else {
                super.image = nil
                return
            }
            if let serverImage = newValue as? ServerImage {
                setServerImage(serverImage)
            } else {
                super.image = newValue
            }
        }
    }
```

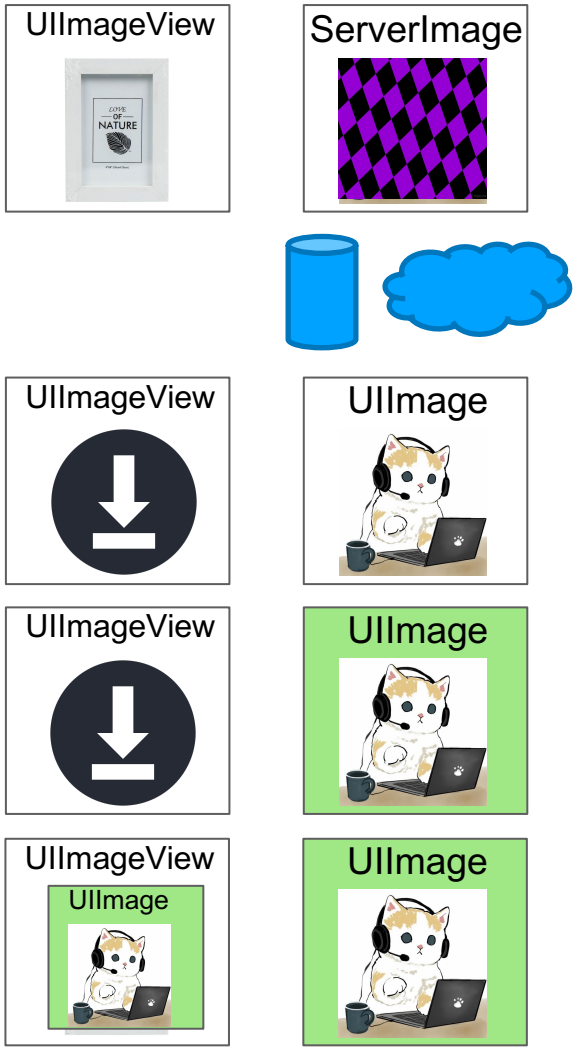
Установка картинки в OImageView

```
override open var image: UIImage? {
        get {
            return super.image
        }

        set {
            cancelCurrentOperation()
            guard let newValue = newValue else {
                super.image = nil
                return
            }
            if let serverImage = newValue as? ServerImage {
                setServerImage(serverImage)
            } else {
                super.image = newValue
            }
        }
    }
```

Установка изображения



Установка изображения



UIButton + ServerImages

Swizzling + UIImageView

Так как UIButton под капотом содержит UIImageView в этом случае дополнительных действий не потребуется.

UIImageView + наследование

Необходимо реализовывать наследника UIButton с обработкой кастомной логики установки серверных картинок

Или использовать swizzling методов установки картинок

Исключения

- Использование картинок в NSAttributedString
- UIBarButtonItem
- UITabBarItem

Включение в пакет приложения

Так как картинок для этих элементов не так много, то их можно перевести на векторую графику и включить в пакет приложения

Предварительная загрузка

Такие ассеты можно в блокирующем режиме скачивать при старте приложения.

Версионирование

- К этому решению не должно применяться версионирование
- Но при необходимости можно добавлять версию пакета картинок в путь до CDN

URL = baseCDNPath + “/” cdnPath

URL = baseCDNPath + “1_0” + “/” cdnPath

Итоги

Затраты

- 2 недели на разработку и прототипирование
- 1 неделя на отладку и тестирование
- Мы не проводили каких-то широкомасштабных рефакторингов приложения. Максимум – подмена класса UIImageView на наш кастомный OImageView, но мы это делали просто через replaceAll в редакторе кода.
- Около 20 картинок (стрелочки назад, иконки таб бара, плейсхолдеры итд) были включены в пакет приложения в отдельный фреймворк.
- На старте загружаются около 40 наиболее часто встречающихся в приложении картинок.

Итоги

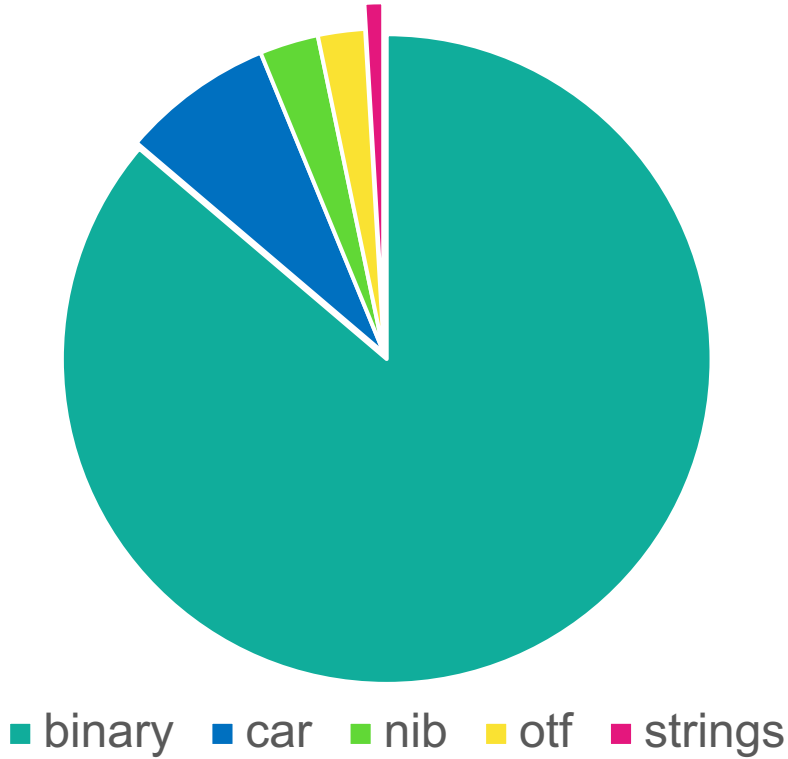
Результаты

- Минимальные изменения UX
- -120 мб ассетов
- Сокращение времени компиляции (- 40 секунд на сборку car файлов)

Вес приложения 232 –> 112 - 42%
Вес ipa 155 –> 46 -70%

Распределение веса по пакету приложения после переноса xcasset на backend

Тип файла	Вес в Мб
binary	93,7
car	8,2
nib	3,2
otf	2,6
strings	1,0



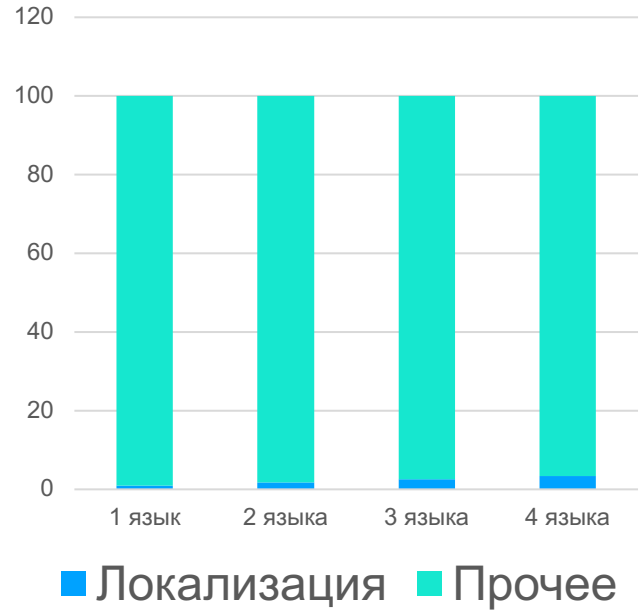
Перенос локализации на backend

3

Отношение веса локализации к остальному весу пакета

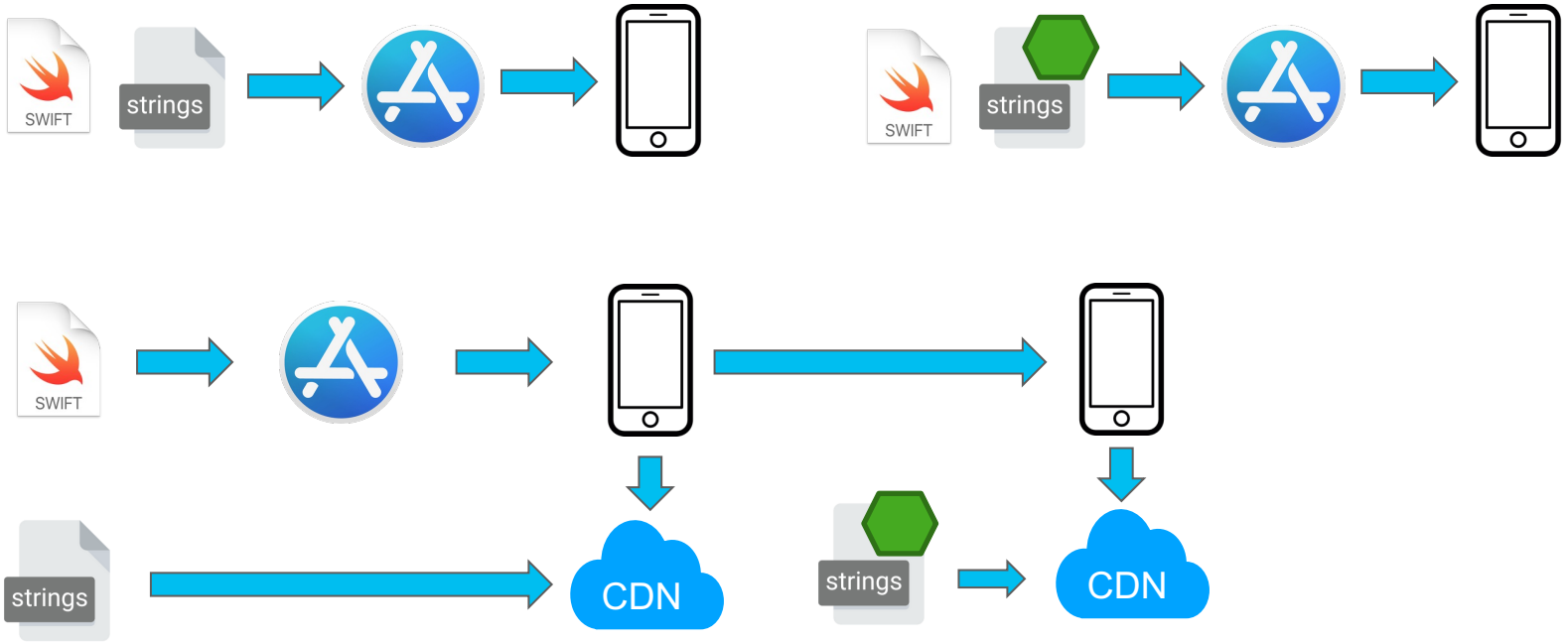
- % веса локализации в приложении растет
- Но все еще не превышает 4% для для локализации на 4 языка

Вес пакета приложения		112
Локализация	Вес файлов	Относительный вес локализации в %
1 язык	1	0,88
2 языка	2	1,75
3 языка	3	2,61
4 языка	4	3,45



Главный плюс серверной локализации

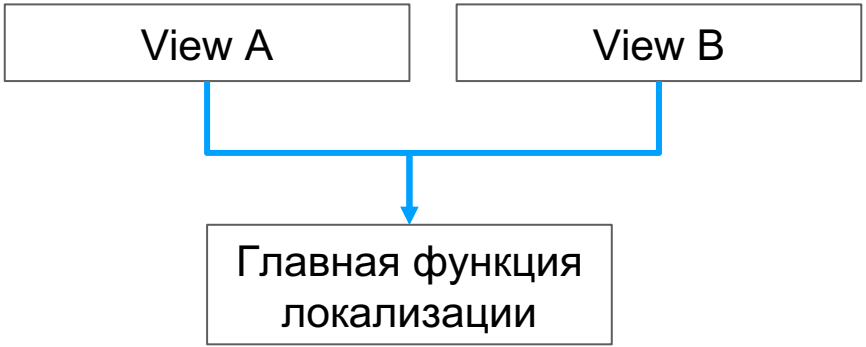
Возможность исправить какие-то ошибки без релиза в AppStore



Исходные требования к вашему приложению

- Использование Swiftgen или его аналога
- IB файлы не используют локализацию напрямую

Главное – наличие центральной точки где ключ локализации превращается в конкретную строку



Как мы работаем с локализацией

UIView

UIViewController

Presenter

```
let title = TextEx.title
let subtitle = TextEx.subtitle("Hello!!!")
let text = title + "\n" + subtitle
label.attributedText = NSAttributedString(
    string: text,
    attributes: [
        .font: UIFont.systemFont(ofSize: 17),
        .foregroundColor: UIColor.red,
        .paragraphStyle: someParagraphStyle
    ]
)
```

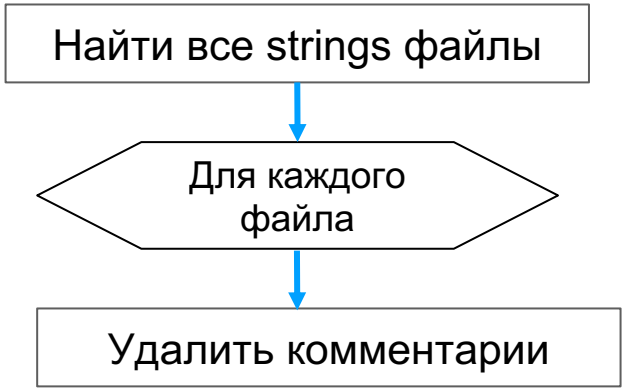
```
extension Text {
    private static func tr(_ table: String, _ key: String, _ args: CVarArg...) -> String {
        let bundle = BundleToken.bundle
        let format = bundle.localizedString(forKey: key, value: nil, table: table)
        return String(format: format, locale: Locale.current, arguments: args)
    }
```

Скрипт для подготовки файлов к выгрузке на CDN

Найти все strings файлы

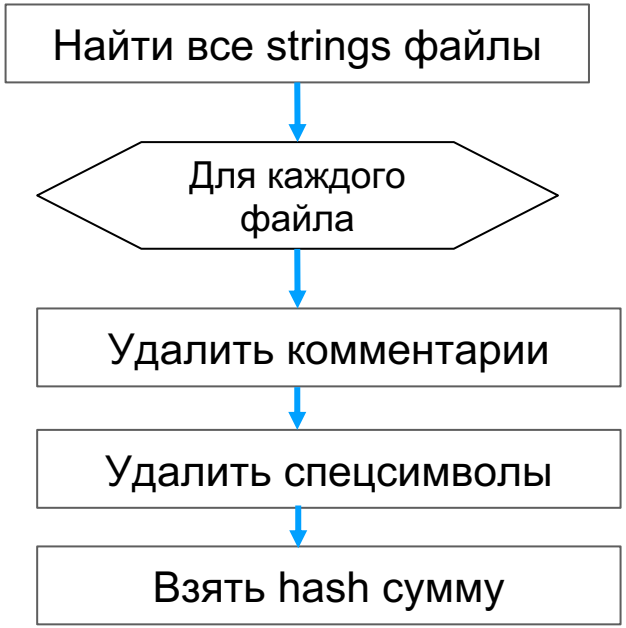

```
// Строка 1
"StringKey1" = "Строка1";

// Строка 2
"StringKey2" = "Строка2";

// Строка 3
"StringKey3" = "Строка3";
```

Скрипт для подготовки файлов к выгрузке на CDN



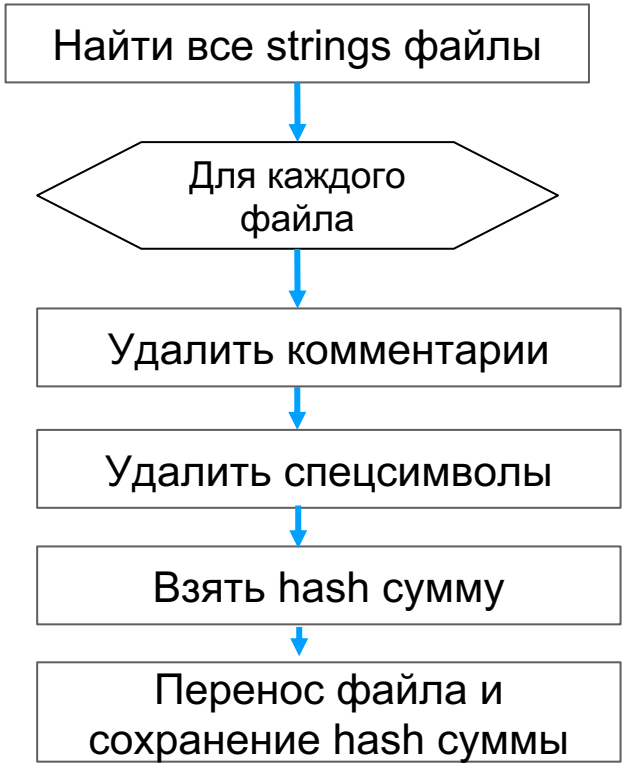

```
"StringKey1" = "Строка1";
"StringKey2" = "Строка2";
"StringKey3" = "Строка3";
```

Скрипт для подготовки файлов к выгрузке на CDN




```
StringKey1#Строка1
StringKey2#Строка2
StringKey3#Строка3
```

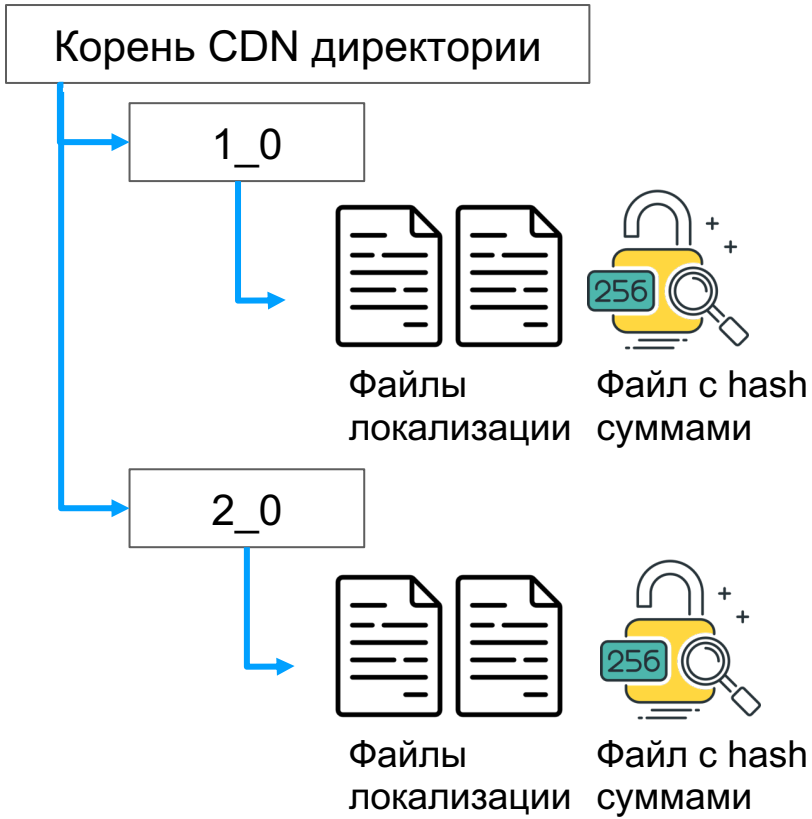
Скрипт для подготовки файлов к выгрузке на CDN



Файл с hash суммами

Файлы локализации

Скрипт для подготовки файлов к выгрузке на CDN



Структура локального хранилища локализации

Структуры записываются в генерируемый swift файл содержащий 2 вложенных словаря

1й ключ – имя модуля (или его bundleID)
2й ключ – ключ локализации

```
var container: [String: [String: String]] = [
    "ru.moduleA.bundleIdentifier": [
        "LocaliseKey": "Русская локализация строки"
    ]
]
```

Структура локального хранилища локализации
При мультиязычной локализации

В структуру добавляется дополнительный 3й словарь, который используется для хранения разных языковых версий одной текстовки

3й ключ – код локализации (например ru или en)

```
var container: [String: [String: [String: String]]] = [
    "ru.moduleA.bundleIdentifier": [
        "LocaliseKey": [
            "ru": "Русская локализация строки",
            "en": "English localization string",
        ]
    ]
]
```

Алгоритм загрузки локализации

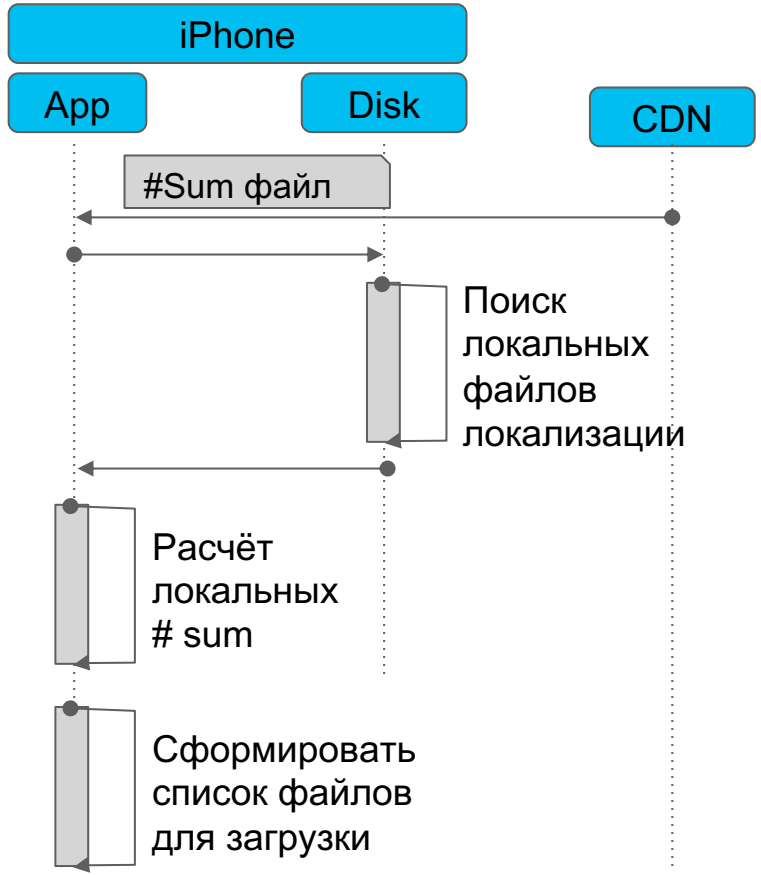
Загрузка hash файла с CDN

↓

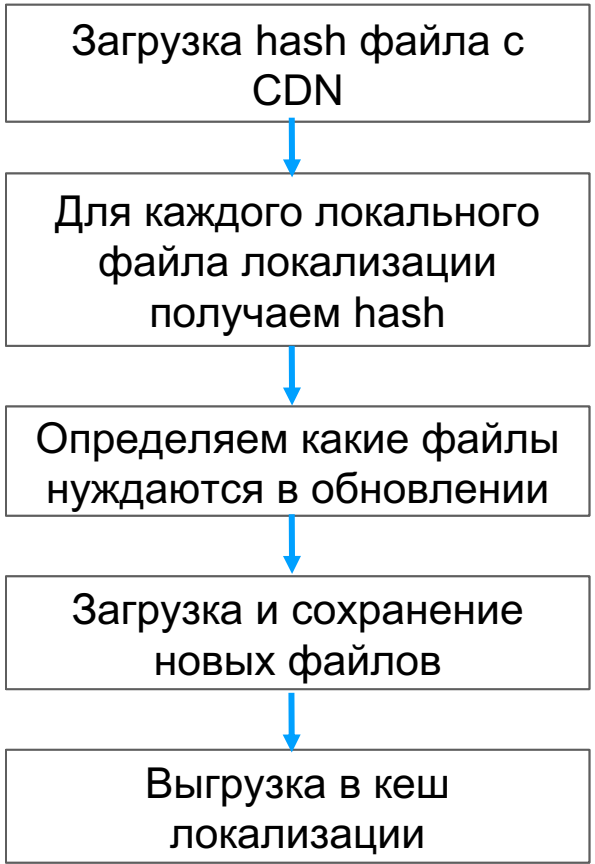
Для каждого локального файла локализации получаем hash

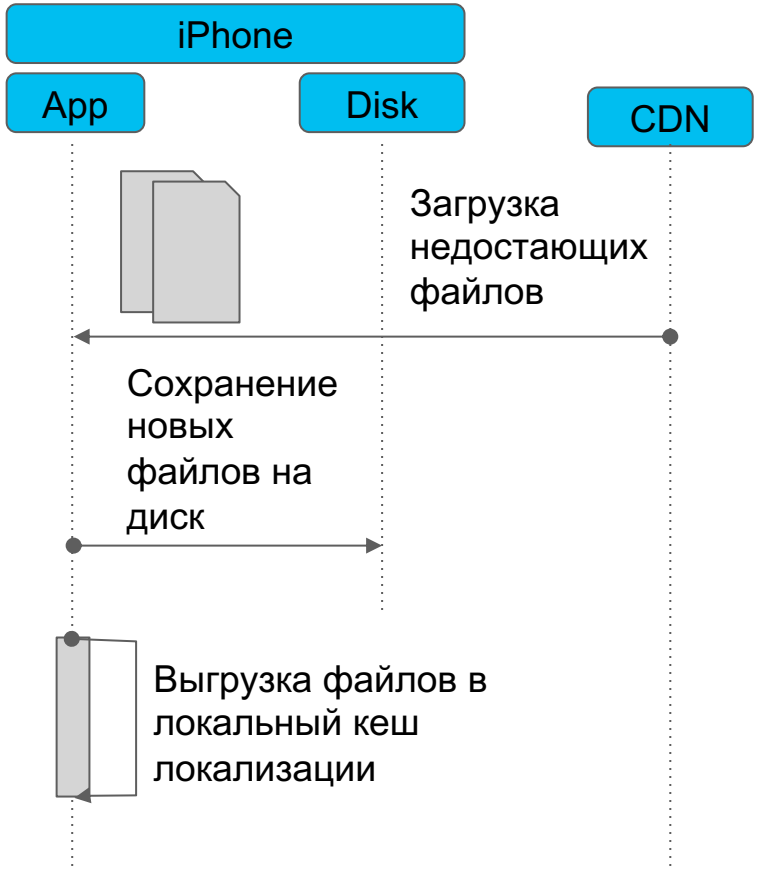
↓

Определяем какие файлы нуждаются в обновлении



Алгоритм загрузки локализации



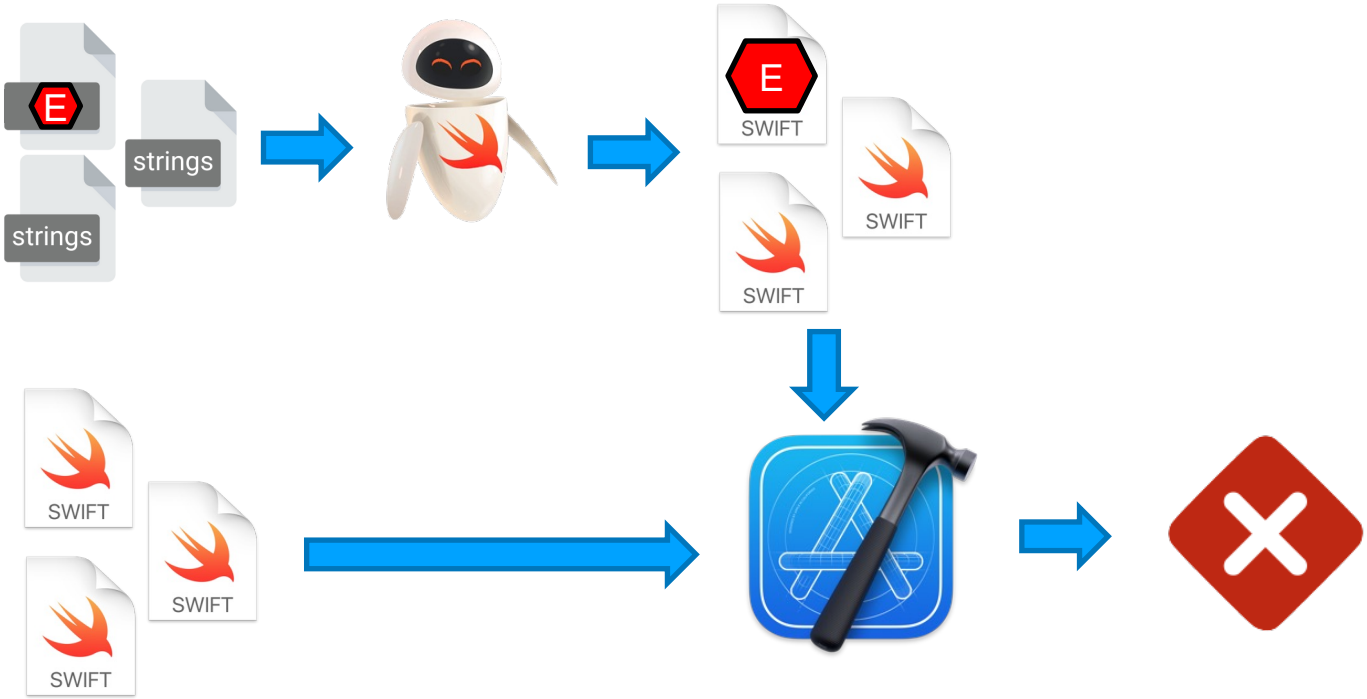


Измененный SwiftGen файл

```swift
extension Text {
    private static func tr(_ table: String, _ key: String, _ args: CVarArg...) -> String {
        let bundle = BundleToken.bundle
        let format = bundle.localizedString(forKey: key, value: nil, table: table)
        return String(format: format, locale: Locale.current, arguments: args)
    }
}

extension Text {
    private static func tr2(_ table: String, _ key: String, _ args: CVarArg...) -> String {
        let bundle = BundleToken.bundle
        let bundleID = bundle.bundleIdentifier!
        let format = LocalizationStore.shared.container[bundleID]?[key] ?? key
        return String(format: format, locale: Locale.current, arguments: args)
    }
}
```

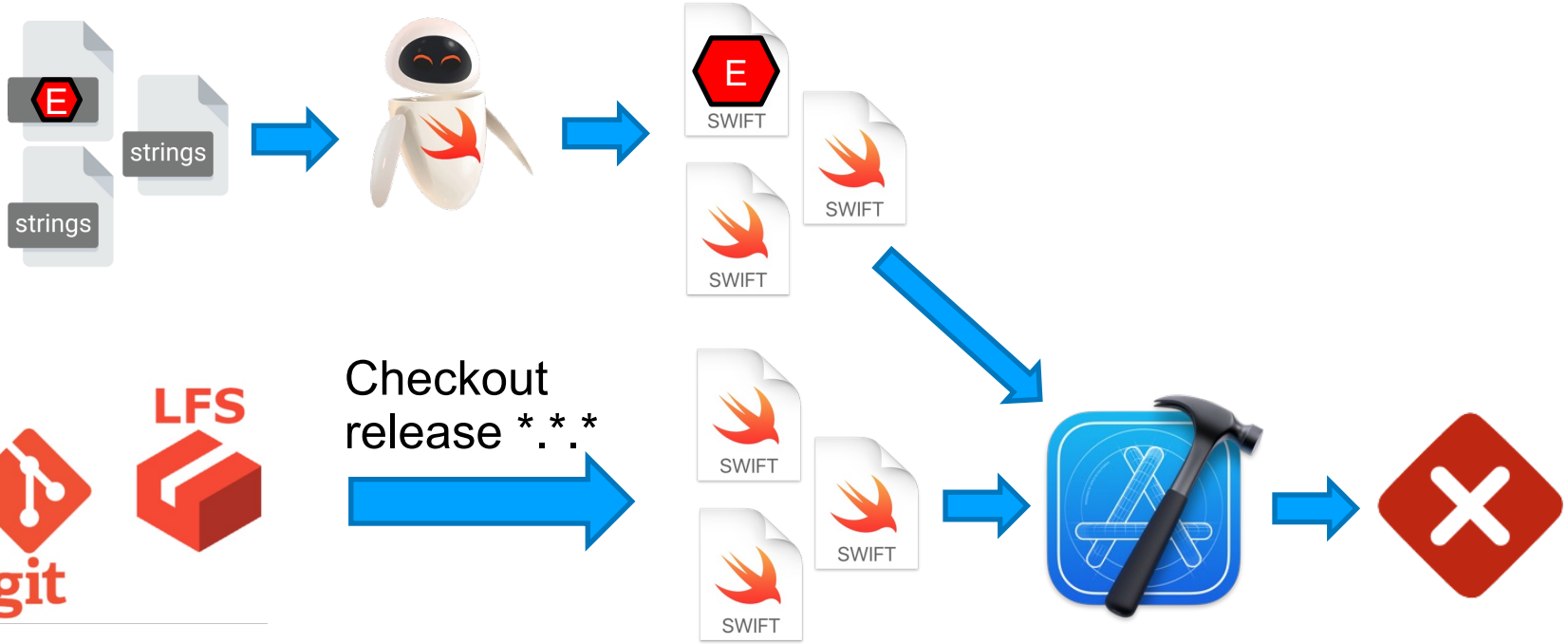
```swift
var container: [String: [String: String]] = [
    "ru.moduleA.bundleIdentifier": [
        "LocaliseKey": "Русская локализация строки"
    ]
]
```

Обеспечение Compile-time проверок корректности локализации



Реализация проверок на CI

Новые файлы
локализации



Хранение дефолтной локализации в пакете приложения

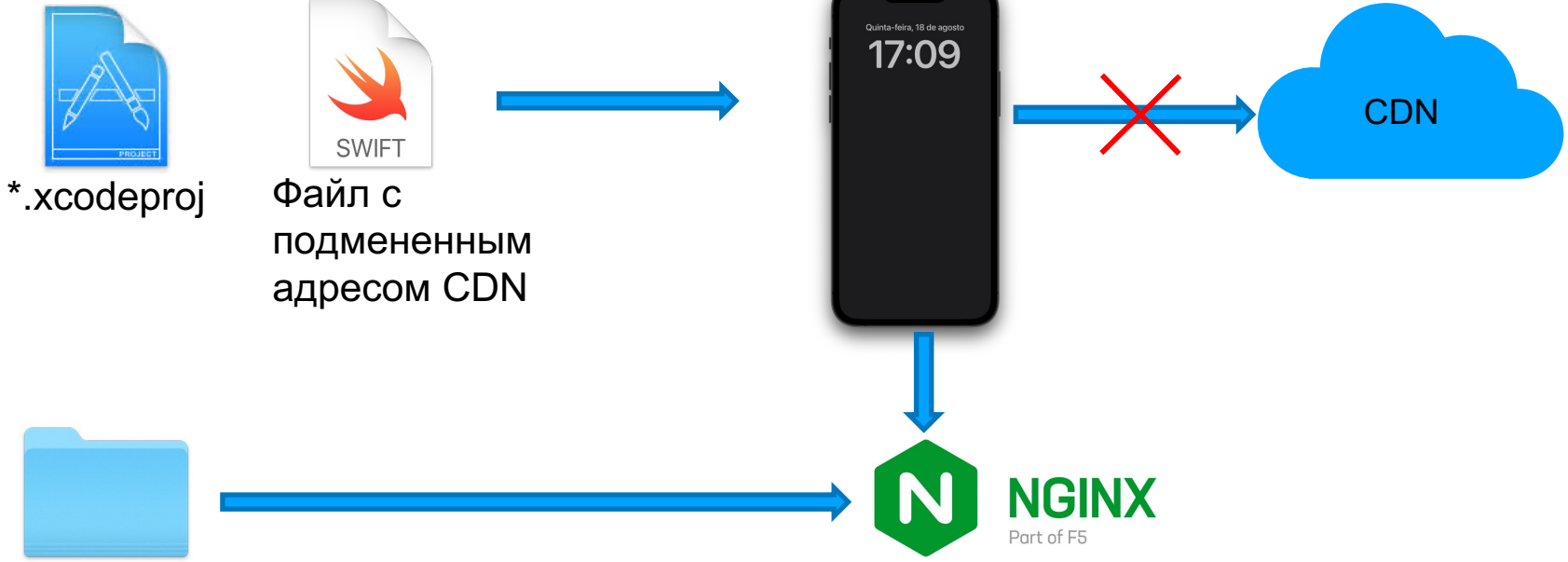
Pros

Отсутствие зависимости от хорошего соединения при первом запуске приложения

Cons

Не получится сэкономить лишние 1-4 мб веса пакета приложения
Больше внимания со стороны Apple

Локальное тестирование механизма серверных текстовок



*.xcodeproj

Файл с подмененным адресом CDN

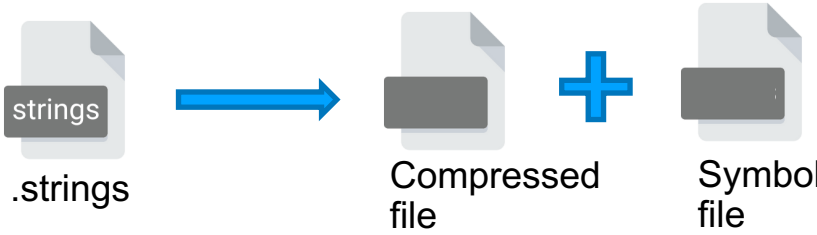
Архивы локализации

Локальный сервер на машине разработчика

Возможные улучшения

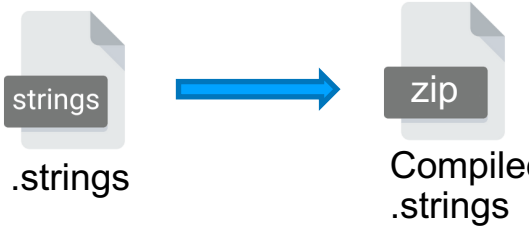
- Сжатие локализации

Huffman алгоритм сжатия



≈ 30% выигрыш

Zip архивация



≈ 75% выигрыш

Возможные улучшения

- Сжатие локализации
- Частичное обновление локализации batch апдейтами

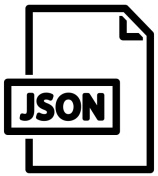
Дефолтная реализация

.strings

При изменении даже одной строки в файле мы вынуждены скачивать его целиком

```
"StringKey1" = "Строка1";
"StringKey2" = "Строка2";
"StringKey3" = "Строка3";

"StringKey1" = "Строка1";
"StringKey2" = "Новая строка";
"StringKey3" = "Строка3";
```

Batch апдейты




```
{
    "ModuleA": {
        "StringKey2": "Новая строка"
    }
}
```

Передача только измененных частей файла

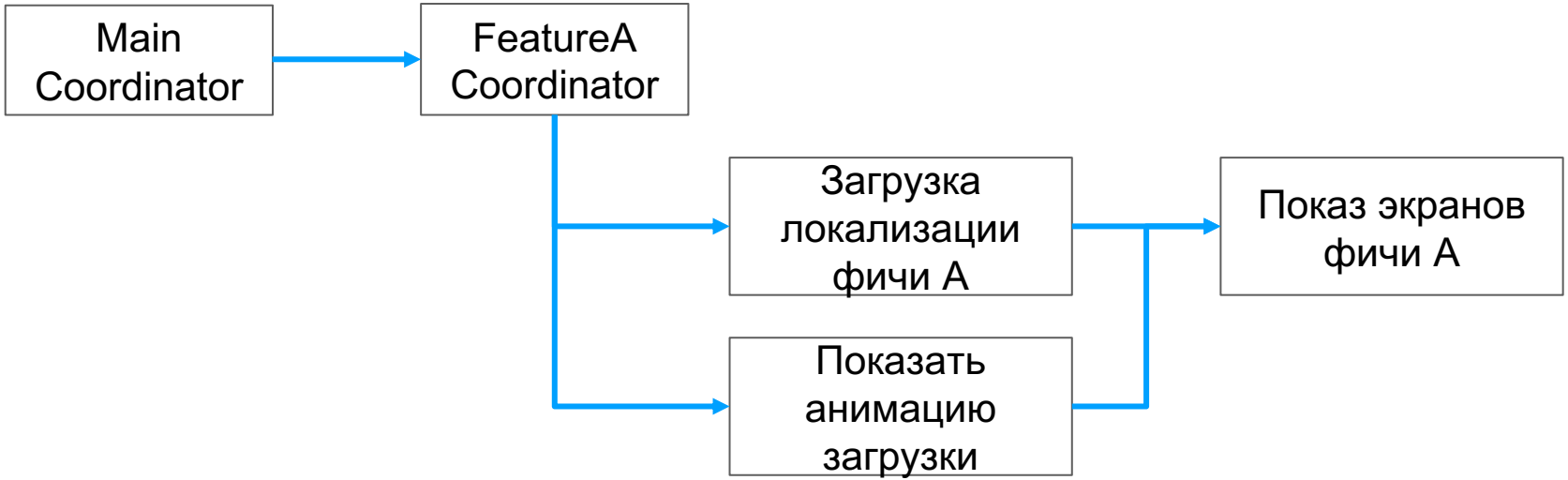
Возможные улучшения

- Сжатие локализации
- Частичное обновление локализации batch апдейтами
- Хранение локализации в БД

- Быстрее чтение
- Удобнее обработка сложных запросов
- Удобное частичное извлечение и обновление данных

Возможные улучшения

- Сжатие локализации
- Частичное обновление локализации batch апдейтами
- Хранение локализации в БД
- Распределенная загрузка локализации



Серверная локализация
Итоги

Затраты

- 2 недели на разработку и прототипирование
- 1 неделя на отладку и тестирование
- Мы не проводили каких-то широкомасштабных рефакторингов приложения.
- На старте приложения загружается локализация

Результаты

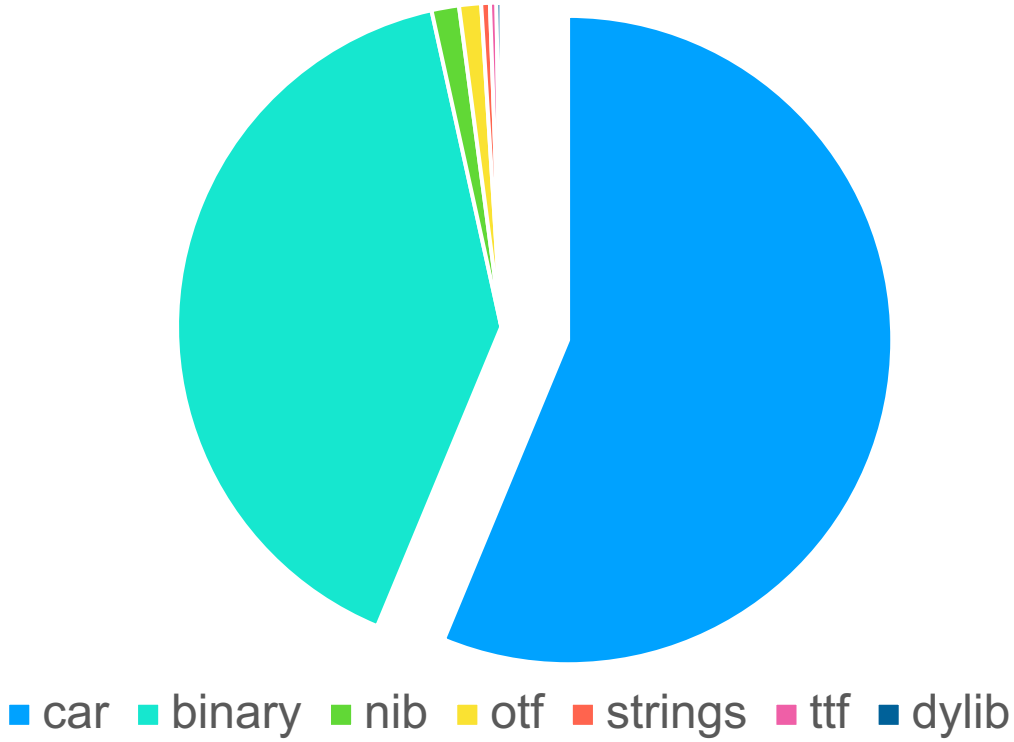
- Минимальные изменения UX
- Упрощение ревью в Apple
- Возможность вносить изменения в локализацию без релизов

Общие итоги

.app 234мб -> 112мб -52%
.ipa 168мб -> 46мб -72%

Большие car файлы – главный фактор и причина большого веса вашего приложения

Тип файла	Вес в Мб
car	130,8
binary	93,7
nib	3,2
otf	2,6
strings	1,0
ttf	0,7
dylib	0,6



Если избавиться от лишних ресурсов, то и другие оптимизации начнут приносить большую пользу

	До удаления ресурсов	После удаления ресурсов
Статическая линковка	15,5	27,0
Swift Optimization to SIZE	1,4	2,5
Удаление бинарных флагов	0,3	0,9
Opensource исходными кодами	0,6	1,8

Если избавиться от лишних ресурсов, то и другие оптимизации начнут приносить большую пользу

	До удаления ресурсов	После удаления ресурсов
Статическая линковка	15,5	27,0
Swift Optimization to SIZE	1,4	2,5
Удаление бинарных флагов	0,3	0,9
Opensource исходными кодами	0,6	1,8
Сокращенная Swift Reflection	0,7	1,3
Отключение Swift Reflection	1,4	2,5

Спасибо за внимание

Евтухов Александр
Банк Открытие

@AlexDarked

Вакула Максим
ООО «КОДЕ»

@VakulaMaksim

Прожарка Tuist