ozon{tech

@inlinable оптимизация в Swift

Максим Гришутин
Руководитель группы разработки мобильных приложений iOS
Ozon

Немного обо мне

ozon{tech

- Успел поработать над:
  - White label приложением для банков (ВТБ, ОТП, Абсолют и еще 100+)
  - Приложение желтого банка
- Я отвечаю за iOS приложение для селлеров в Ozon
- Преподаватель iOS курсов по SwiftUI в Ozon School Route 256
- Стараюсь внедрять и продвигать новые технологии, а еще с педантичностью подходить к вопросам оптимизации

О чем мы поговорим

ozon{tech

🔎 Рассмотрим проблему

🏎️ Оптимизации кода

⚙️ Примеры работы выстраивания

💪 Результаты оптимизации

☝️ Выводы

Немного предыстории

ozon{tech

- Нашел proposal в репозитории swift об inlinable
- Провел немного тестов, прочитал огромное количество обсуждений на форуме swift
- Начал использовать у себя в коде

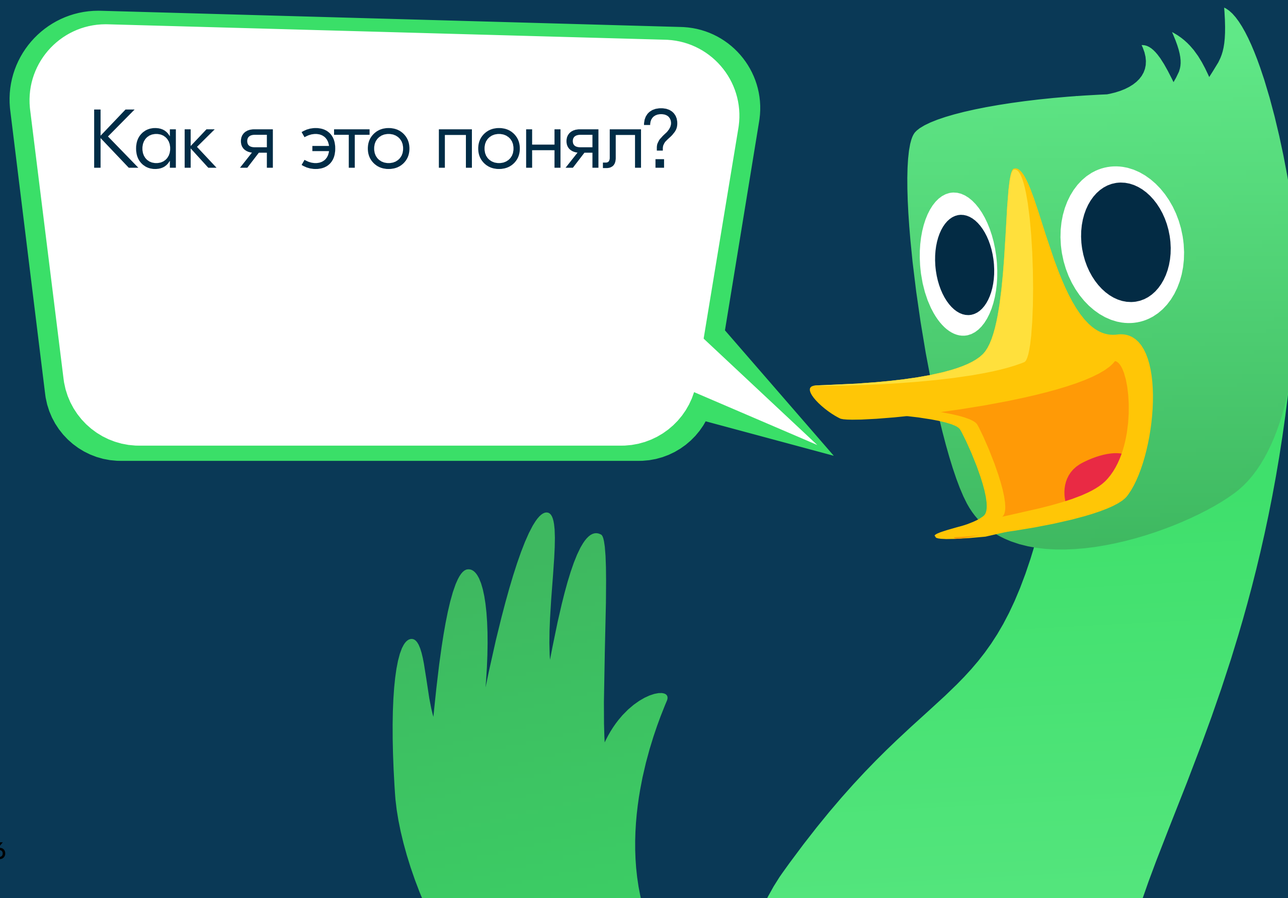
ozon{tech

У нас проблема 👨‍🚀

Основная проблема

ozon{tech

Встраивание почти не используют



Я решил проверить

ozon{ech

- Фреймворки, которые я использую
- Фреймворки с большим количеством ⭐⭐⭐⭐⭐

Примеры зависимостей не использующих встраивание:

ozon{ech

- Alamofire
- Nuke
- Charts
- И многие другие сторонние фреймворки

Только RxSwift использовал 😅

Почему inlinable не используют?

ozon{ech

- Многие считают inlinable очень сложным
- Нет понимания отличия **@inlinable** и **@inline(__always)**

И так сойдет!

Оптимизации

Какие оптимизации

ozon{tech

- Их очень много (100+)
- Обычно не заметны для разработчика

Сегодня мы подробно рассмотрим оптимизацию встраивания

Статья об оптимизациях в LLVM

https://medium.com/@JMangia/swift-c-llvm-compiler-optimization-842012568bb7

Примеры оптимизаций

ozon{tech

Встраивание:

```
func calculateAndPrint(_ value: String) {
    let valueForPrinting = value + value
    myPrint(valueForPrinting)
}

func myPrint(_ string: String) {
    print("My: \(string)")
}
```

```
func calculateAndPrint(_ value: String) {
    let valueForPrinting = value + value
    print("My: \(valueForPrinting)")
}
```

Удаление переменных:

```
func calculateAndPrint(_ value: String) {
    let valueForPrinting = value + value
    print(valueForPrinting)
}
```

```
func calculateAndPrint(_ value: String) {
    print(value + value)
}
```

Важно

ozon{tech

@inlinable и другие оптимизации не работают, если не включить нужный уровень оптимизации LLVM

Optimization Level	Optimize for Speed [-O]
Debug	Optimize for Speed [-O]
Release	Optimize for Speed [-O]

Не забудьте это сделать, когда будете тестировать свой код 😓

Что дает встраивание

ozon{tech

Преимущества 💪:

- Нет затрат на вызов функций
- Нет затрат на создание отдельного пула для функции
- Нет затрат на копирование

Недостатки 🚨:

- Увеличивает размер приложения
- Из-за размера, процессор может забить кеш и работал медленнее

Что дает встраивание

ozon{ech

Затраты на копирование

```
struct Container {
    let value: String
}
```

Что дает встраивание

ozon{tech

Затраты на копирование

```
struct Container {
    let value: String
}

/// Вывод адреса структуры
func printAddress<T>(for value: T) {
    withUnsafePointer(to: value) {
        print($0)
    }
}
```

```
func process(container: Container) {
    printAddress(for: container)
}
```

Что дает встраивание

ozon{tech

Затраты на копирование

```
struct Container {
    let value: String
}

/// Вывод адреса структуры
func printAddress<T>(for value: T) {
    withUnsafePointer(to: value) {
        print($0)
    }
}

let container = Container(value: "")
printAddress(for: container)  // → 0x000000016f363100
process(container: container) // → 0x000000016f3630b0
printAddress(for: container)  // → 0x000000016f363100
```

```
func process(container: Container) {
    printAddress(for: container)
}
```

Что дает встраивание

ozon{tech

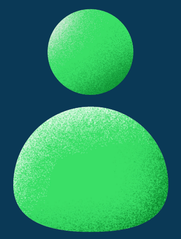
Затраты на копирование

```
struct Container {
    let value: String
}

/// Вывод адреса структуры
func printAddress<T>(for value: T) {
    withUnsafePointer(to: value) {
        print($0)
    }
}

let container = Container(value: "")
printAddress(for: container)  // → 0x000000016f363100
process(container: container) // → 0x000000016f3630b0 ← Скопированно
printAddress(for: container)  // → 0x000000016f363100
```

```
func process(container: Container) {
    printAddress(for: container)
}
```

Поговорим об inline ☝️


Ходит за кофе
Кофейня делает кофе
Любитель кофе
Кофейня

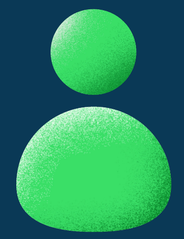


Любитель кофе
компилятор

Кофейня
func makeCoffee()

Любитель кофе хочет делать кофе сам, но не знает как

ozon{tech

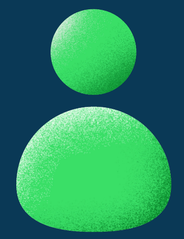


Любитель кофе
компилятор

Кофейня
func makeCoffee()

Любитель кофе хочет делать кофе сам, но не знает как

ozon{tech



Любитель кофе
компилятор

Ходит за кофе

Кофейня делает кофе

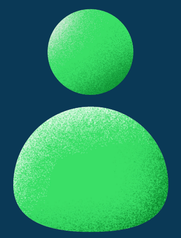
Кофейня
func makeCoffee()

Рецепт:

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

Он узнаёт как готовится его кофе (@inlinable)

ozon{tech



Любитель кофе
компилятор

Ходит за кофе

Кофейня делает кофе

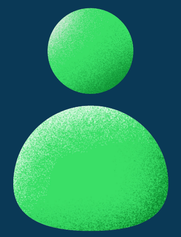
Кофейня
@inlinable func makeCoffee()

Рецепт:

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

И теперь может делать кофе где угодно

ozon{tech



Любитель кофе
компилятор

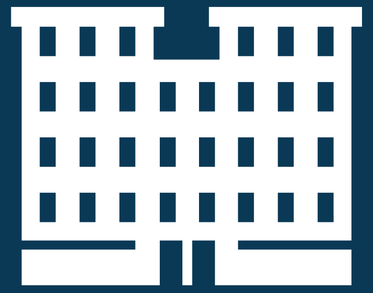
Ходит за кофе

Кофейня делает кофе

Кофейня
@inlinable func makeCoffee()

Рецепт:

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

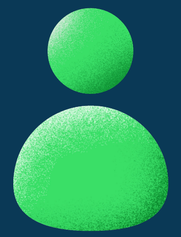


Дом

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

Решает экономить и никогда не ходить в кофейню

ozon{tech



Любитель кофе
компилятор

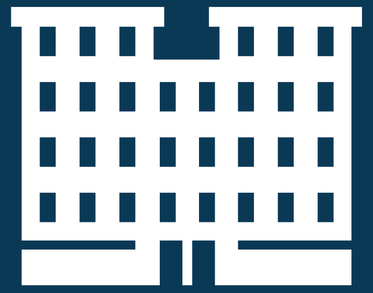
Ходит за кофе

Кофейня делает кофе

Кофейня
@inlinable func makeCoffee()

Рецепт:

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

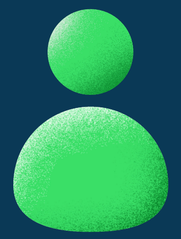


Дом

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

Решает экономить и никогда не ходить в кофейню @inline(__always)

ozon{tech



Любитель кофе
компилятор

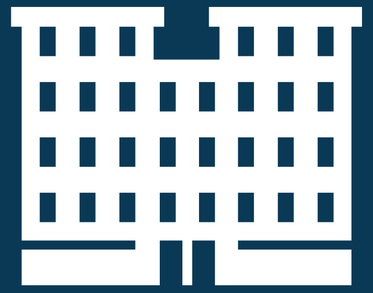
Ходит за кофе

Кофейня делает кофе

Кофейня
@inline(__always)
@inlinable func makeCoffee()

Рецепт:

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

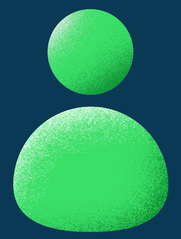


Дом

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок

Теперь он всегда будет делать кофе у себя дома

ozon{tech

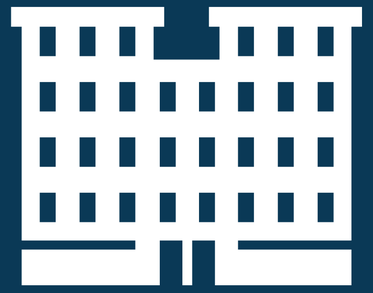


Любитель кофе
компилятор

Кофейня
@inline(__always)
@inlinable func makeCoffee()

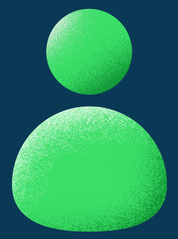
Рецепт:

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок



Дом

1. Молят зерна
2. Варят кофе
3. Добавляют молоко
4. Делают рисунок



Любитель кофе
компилятор

@inlinable — мы узнаем рецепт и можем его использовать когда и где захотим

@inline(__always) — мы говорим что всегда будем использовать наш рецепт там, где мы находимся, например дома

`@inlinable` — атрибут, позволяющий сделать реализацию открытой

ozon{ech

- Своеобразный аналог **public** модификатора для реализаций
- Не говорит что метод должен быть встроен
- Может указать на ошибки мешающие встраиванию. Например:
  - использование **private var** в реализации

@inlinable

ozon{ech

Можно применять для:

- Функций
- Вычисляемых свойств
- subscript
- init / deinit

Рассматриваемые атрибуты

ozon{ech

Атрибуты, позволяющие функции поддерживать встраивание:

- @inlinable
- @usableFromInline

Атрибуты, говорящие компилятору, нужно встраивать или нет:

- @inline(__always)
- @inline(__never)

Поддержка встраивания

ozon{tech

Атрибуты, позволяющие реализациям поддерживать встраивание:

- @inlinable
- @usableFromInline

- Выступают как протоколы, но для реализаций.
- Влияют на ABI Stability *(но сегодня не об этом)*
- Компилятор сам решает когда нужно произвести встраивание.
- Атрибуты с able приставкой просто добавляют возможность, в данном случае встраивания

Указание встраивания

ozon{tech

Атрибуты, говорящие компилятору, нужно встраивать или нет:

- @inline(__always)
- @inline(__never)

Мы решаем за компилятор, будет встраиваться или нет

Немного о других языках

ozon{ech

Другие языки тоже поддерживают встраивание

C++

```
inline int sum(int a, int b) {
    return a + b;
}
```

Kotlin

```
inline fun sum (a: Int, b: Int): Int {
        return a + b
}
```

А Objective C не поддерживает 😅, если только не использовать C реализацию (.mm файлы)

Рассмотрим примеры

Пример

@inlinable

ozon{ech

Возьмем код:

```
extension Int {
    static let one = 1

    func increased() → Int {
        self + .one
    }
}
```

Пример

@inlinable

ozon{ech

Добавим @inlinable

```
extension Int {
    static let one = 1

    @inlinable
    func increased() → Int {
        self + .one
    }
}
```

Пример

@inlinable

ozon{ech

Получаем ошибку 🛑

```
extension Int {
    static let one = 1

    @inlinable
    func increased() → Int {
        self + .one
    }
}
```

Static property 'one' is internal and cannot be referenced from an '@inlinable' function

Пример

@usableFromInline

ozon{tech

Все компилируется и работает 🚀

```
extension Int {
    @usableFromInline
    static let one = 1

    @inlinable
    func increased() → Int {
        self + .one
    }
}
```

Но всегда ли будет встраиваться данный код? 🤔

Пример

@inline(__always)

ozon{ech

Что бы функция всегда встраивалась - добавим **@inline(__always)**

```
extension Int {
    @usableFromInline
    static let one = 1

    @inlinable
    @inline(__always)
    func increased() → Int {
        self + .one
    }
}
```

Результаты оптимизации

Как я тестировал

ozon{tech

- Каждый тест прогонялся 100 раз
- Брался средний результат
- Все тесты выполнялись синхронно

```
let t0 = DispatchTime.now().uptimeNanoseconds
// Тестируемый код
let t1 = DispatchTime.now().uptimeNanoseconds
print(Double(t1-t0) / 1e9, "seconds")
```

Начнем с простых примеров

ozon{ech

- Примеры простые и без контекста
- Позволяют увидеть разницу скорости выполнения

Пример

ozon{tech

Функция нахождения факториала

```
public func factorial(_ value: Int) → Int {
    if value == 1 {
        return 1
    } else {
        return value * factorial(value - 1)
    }
}
```

```
@inline(__always)
@inlinable
public func factorial(_ value: Int) → Int {
    if value == 1 {
        return 1
    } else {
        return value * factorial(value - 1)
    }
}
```

Среднее время одного запуска:

public	inline
0.113	0.109

Разница: **3.5%**

Еще пример

ozon{tech

```
func randomState(state: UInt64 = .random(in: .min ... .max)) -> () -> UInt64 {
    var state = state
    func nextRandomState() -> UInt64 {
        state &+= 0xa0761d6478bd642f
        let mul = state.multipliedFullWidth(by: state ^ 0xe7037ed1a0b428db)
        return mul.high ^ mul.low
    }
    return nextRandomState
}
```

0.94s

```
@inline(__always)
func randomState(state: UInt64 = .random(in: .min ... .max)) -> () -> UInt64 {
    var state = state
    @inline(__always)
    func nextRandomState() -> UInt64 {
        state &+= 0xa0761d6478bd642f
        let mul = state.multipliedFullWidth(by: state ^ 0xe7037ed1a0b428db)
        return mul.high ^ mul.low
    }
    return nextRandomState
}
```

0.05s

Примеры на реальных фреймворках

ozon{ech

- Alamofire
- BetterCodable

Alamofire

@inlinable

ozon{ech

Найдем функцию которая:

- Будет вызываться много раз
- Внутри мало кода

Alamofire

@inlinable

ozon{ech

Возьмём функцию map для response:

```
public func map<NewSuccess>(_ transform: (Success) → NewSuccess)
    → DownloadResponse<NewSuccess, Failure> {
    DownloadResponse<NewSuccess, Failure>(request: request,
                                          response: response,
                                          fileURL: fileURL,
                                          resumeData: resumeData,
                                          metrics: metrics,
                                          serializationDuration: serializationDuration,
                                          result: result.map(transform))
}
```

Alamofire

ozon{tech

@inlinable

Установим для неё встраивание:

```
@inlinable
@inline(__always)
public func map<NewSuccess>(_ transform: (Success) → NewSuccess)
    → DownloadResponse<NewSuccess, Failure> {
    DownloadResponse<NewSuccess, Failure>(request: request,
                                          response: response,
                                          fileURL: fileURL,
                                          resumeData: resumeData,
                                          metrics: metrics,
                                          serializationDuration: serializationDuration,
                                          result: result.map(transform))
}
```

Alamofire

ozon{tech

Получаем такой результат:

Среднее время одного запуска:

Без встраивания	С встраиванием
~0.006	~0.004

Разница не огромная, но получилось все, же быстрее 😎

BetterCodable

ozon{tech

Здесь мы будем смотреть обратный результат, так как встраивание уже добавили

Возьмем код, который:

- Исполняется часто
- Не имеет много логики
- 🚨 Имеет в себе объявление и вызов объявленных функций

BetterCodable

ozon{ech

@inlinable

Возьмем готовый код, для декодинга без потерь:

```
public static var losslessDecodableTypes: [(Decoder) → LosslessStringCodable?] {
    @inline(__always)
    func decode<T: LosslessStringCodable>(_: T.Type) → (Decoder) → LosslessStringCodable? {
        return { try? T.init(from: $0) }
    }

    @inline(__always)
    func decodeBoolFromNSNumber() → (Decoder) → LosslessStringCodable? {
        return { (try? Int.init(from: $0)).flatMap { Bool(exactly: NSNumber(value: $0)) } }
    }

    return [
        decode(String.self),
        decodeBoolFromNSNumber()
    ]
}
```

BetterCodable

ozon{ech

@inlinable

Уберем из него @inline(__always)

```
public static var losslessDecodableTypes: [(Decoder) → LosslessStringCodable?] {
    func decode<T: LosslessStringCodable>(_: T.Type) → (Decoder) → LosslessStringCodable? {
        return { try? T.init(from: $0) }
    }

    func decodeBoolFromNSNumber() → (Decoder) → LosslessStringCodable? {
        return { (try? Int.init(from: $0)).flatMap { Bool(exactly: NSNumber(value: $0)) } }
    }

    return [
        decode(String.self),
        decodeBoolFromNSNumber()
    ]
}
```

BetterCodable

ozon{ech

Получаем такой результат:

Без встраивания	С встраиванием
~0.020	~0.003

Разница составила **15%**

Выводы

Что использовать?

ozon{ech

- `@inlinable`: используем безболезненно
- `@inline(__always)`: используем осторожно - может понизить производительность из-за увеличения размеров кода

Когда использовать?

ozon{tech

- Код вызывается часто
- Код не занимает много места
- Код имеет в себе объявление функций
- Код из Core слоя
- Код в модульной архитектуры

Когда использовать?

ozon{tech

Пример из репозитория Swift

```
@inlinable
public func map<T>(_ transform: (Element) throws → T) → [T] {
    let initialCapacity = underestimatedCount
    var result = ContiguousArray<T>()
    result.reserveCapacity(initialCapacity)

    var iterator = self.makeIterator()

    // Add elements up to the initial capacity without checking
    for _ in 0 ..< initialCapacity {
        result.append(try transform(iterator.next()!))
    }
    // Add remaining elements, if any.
    while let element = iterator.next() {
        result.append(try transform(element))
    }
    return Array(result)
}

@inlinable
@inline(__always)
public func _customContainsEquatable(element: Element) → Bool? {
    return nil
}
```

Apple везде использует inlinable оптимизации

В завершении

ozon{tech

- Попробуйте оптимизировать
- Всегда профилируйте свой код перед оптимизациями
- Надеюсь что в будущем больше фреймворков начнут использовать inlinable атрибуты

В завершении

ozon{ech

Взгляд в будущее

Сейчас активно ведутся обсуждения улучшения работы `@inline`(__always) и добавление более безопасного аналога `@inline`(__optimize)

https://forums.swift.org/t/proposal-for-inline-optimize/19320

В завершении

ozon{tech

Взгляд в будущее

Что может дать нам `@inline`(__optimize)?

- Xcode сможет сам подсвечивать места, а может даже сам проставлять **@usableFromInline**
- Весь Swift код сможет полагаться на выбор LLVM и в целом работать сильно быстрее

https://forums.swift.org/t/proposal-for-inline-optimize/19320

Материалы

ozon{tech

- https://github.com/apple/swift/blob/main/stdlib/public/core/Map.swift
- https://github.com/marksands/BetterCodable/
- https://developer.apple.com/videos/play/wwdc2019/416/
- https://forums.swift.org/t/effect-of-inline-on-nested-functions/50202
- https://forums.swift.org/t/when-should-both-inlinable-and-inline-always-be-used/37375/2
- https://swiftrocks.com/understanding-inlinable-in-swift
- https://github.com/apple/swift-evolution/blob/master/proposals/0193-cross-module-inlining-and-specialization.md
- https://developer.apple.com/videos/play/wwdc2019/416/

Репозиторий с ссылками:

ozon{ech

Всем спасибо

GitHub: BarredEwe

Telegram: @BarredEwe

LinkedIn: BarredEwe