КРАТКИЙ ВЗГЛЯД НА URLSESSIONWEBSOCKETTASK В IOS 13

Вчерашний В.Е.

Вчерашний Владислав Евгеньевич - руководитель отдела,
отдел разработки мобильного приложения,
Компания Parler LLC,
преподаватель,
курсы разработки приложений для Apple iOS,
г. Киев, Украина

Аннотация: *в статье анализируется работа технологии WebSockets в операционной системе Apple iOS 13.*

Ключевые слова: *apple, iOS, websockets, Swift.*

Использование WebSockets в iOS никогда не было простым и прозрачным механизмом, как того хотелось бы. Но настал момент, и на ежегодной конференции для разработчиков WWDC 2019 компания Apple показала нам новый инструмент для работы с WebSockets — URLSessionWebSocketTask (который является наследником класса URLSessionTask).

В этой небольшой статье мы познакомимся с данным нововведением и узнаем — что изменится в будущем и станет ли теперь работа с WebSockets проще и понятнее.

Вступление.

В наше время существует огромное количество приложений и большинство из них так или иначе использует REST API для “общения” с сервером. Мы уже давно привыкли, что для отправки или получения данных с сервера нужно просто отправить запрос нужного типа и ожидать ответа. В большинстве случаев этого достаточно, верно?

А что если вам необходимо реализовать “real-time” чат (некоторые умудряются сделать его с помощью Push Notification (!) но это совсем другая история) или автообновляемую ленту новостей (как в Twitter)? Да, достичь этих целей, конечно, можно и с помощью “повторяемых” запросов с определённой периодичностью, вот только недостатков у такого решения намного больше, чем преимуществ (ресурсы, трафик, да и совсем нет гарантий по скорости получения ответа от сервера).

К счастью для решения подобных задач у нас есть WebSockets — где между клиентом и сервером существует постоянное и непрерывное соединение. Простыми словами — клиент подключается к выделенному порту, который прежде открыл сервер. После чего клиент может отправлять ему сообщения или “слушать” его на входящие сообщения.

Но самое главное — каждый подключенный “клиент” мгновенно получает сообщения от сервера и без каких-либо дополнительных запросов.

URLSession — что нового?

Давайте посмотрим в класс **URLSession**. Как вы могли заметить — здесь появилось 3 новых метода, которые необходимы для подключения к серверу:

```
@available(iOS 13.0, *)
open func webSocketTask(with url: URL) -> URLSessionWebSocketTask

@available(iOS 13.0, *)
open func webSocketTask(with url: URL, protocols: [String]) -> URLSessionWebSocketTask

@available(iOS 13.0, *)
open func webSocketTask(with request: URLRequest) -> URLSessionWebSocketTask
```

Приступая к работе

Для начала нам необходимо подключиться к серверу. Поскольку **URLSessionWebSocketTask** является наследником **URLSessionTask** — их набор API весьма похожий:

```
func connect() {
    guard !self.isConnected else { return }
    self.session = URLSession(configuration: .default, delegate: self, delegateQueue: nil)
    self.webSocketTask = self.session.webSocketTask(with: URL(string: self.baseURL)!)
    self.webSocketTask.resume()
  }
```

! Чтобы наблюдать за состоянием соединения не забудьте реализовать методы делегата **URLSessionWebSocketDelegate**. В нем есть всего 2 метода:

optional func urlSession(_ session: URLSession, webSocketTask: URLSessionWebSocketTask, didOpenWithProtocol protocol: String?)

optional func urlSession(_ session: URLSession, webSocketTask: URLSessionWebSocketTask, didCloseWith closeCode: URLSessionWebSocketTask.CloseCode, reason: Data?)

Отправка и получение сообщений

Для отправки сообщений следует использовать метод **Send**, который в качестве аргумента принимает **URLSessionWebSocketTask.Message**. Отправить можно только *String* или *Binary data*, а для получения – – **Receive**.

```
public enum Message {
    case data(Data)
    case string(String)
}

public func send(_ message: URLSessionWebSocketTask.Message, completionHandler: @escaping (Error?) -> Void)

public func receive(completionHandler: @escaping (Result<URLSessionWebSocketTask.Message, Error>) -> Void)
```

Пример реализации методов Send и Receive

```
func sendSignal(message: String) {
    self.webSocketTask.send(.string(message)) { [weak self] error in
        guard let error = error else { return }
        self?.handleError(error.localizedDescription)
    }
}

private func receive() {
    self.webSocketTask.receive { [weak self] result in
        switch result {
        case .failure(let error):
            self?.handleError("Failed to receive message: \(error.localizedDescription)")
        case .success(let message):
            switch message {
            case .string(let text):
                self?.logSignal("Received text message: \(text)")
            case .data(let data):
                print("Received binary message: \(data)")
            @unknown default:
                fatalError()
            }

            self?.receive()
        }
    }
```

Обратите внимание: метод Receive вызывается *только один раз*, по этому для получения других сигналов вам нужно вызывать метод Receive рекурсивно.

Закрытие соединения

Чтобы отключиться от сервера достаточно вызвать метод **Cancel.** Он принимает два параметра – – код закрытия (для информирования о причине закрытия соединения) и reason payload.

```
func disconnect() {
    guard self.isConnected else { return }
```

```
        self.webSocketTask.cancel(with: .goingAway, reason: nil)
    }
```

Сыграем в Ping-Pong?

Иногда сервер может разорвать соединение если он не получает сигнал с определенной периодичностью. Чтобы “разбудить” сервер нужно использовать метод **sendPing**.

```
    func sendPing() {
        self.webSocketTask.sendPing { [weak self] error in
            guard let `self` = self else { return }
            if let error = error {
                self.handleError(error.localizedDescription)
            }
        }
    }
```

Заключение

Использование WebSockets в iOS 13 стало действительно простым, к тому же без необходимости подтягивания сторонних библиотек. Но главный момент, что новый API доступен только для iOS 13. Для более старых версий — извините, пользуйтесь дополнительными библиотеками (Starscream, Socket.IO или SocketRocket).

Мне нравится то, как развивается Network в iOS. Путь, который выбрала компания, не идеальный, но, как по мне, он идет в правильном направлении. В конечном счете мы придем к тому, что у нас будет доступно все из коробки.

Список литературы

1. Документация Apple / URLSessionWebSocketTask. [Электронный ресурс], 2019. Режим доступа: https://developer.apple.com/documentation/foundation/urlsessionwebsockettask/ (дата обращения: 08.01.2021).
2. Документация Apple / URLSessionTask. [Электронный ресурс], 2019. Режим доступа: https://developer.apple.com/documentation/foundation/urlsessiontask/ (дата обращения: 08.01.2021).
3. Репозиторий GitHub / StarScream. [Электронный ресурс], 2019. Режим доступа: https://github.com/daltoniam/Starscream/ (дата обращения: 18.01.2021).
4. Репозиторий GitHub / Socket.IO. [Электронный ресурс], 2019. Режим доступа: https://github.com/socketio/socket.io-client-swift/ (дата обращения: 18.01.2021).
5. Репозиторий GitHub / SocketRocket. [Электронный ресурс], 2019. Режим доступа: https://github.com/facebook/SocketRocket/ (дата обращения: 18.01.2021).