
Яндекс

Яндекс

URL Loading System

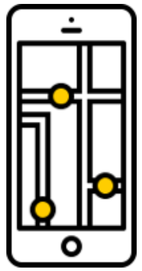
Роман Парадеев

Сегодня в лекции

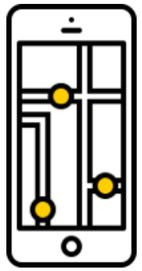
- Зачем iOS-разработчику HTTP?
- Как загрузить картинку
- Как сделать POST-запрос
- Как настроить кэширование
- Как управлять таймаутами и отменять запросы
- Как реализовать аутентификацию
- Как догрузить большой файл в фоне

HTTP

Клиент-серверная архитектура



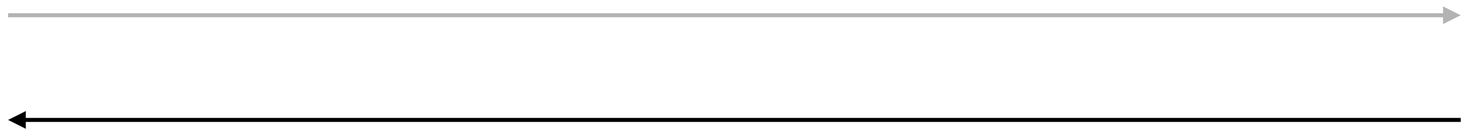
Клиент-серверная архитектура



Мне нужен список сообщений!

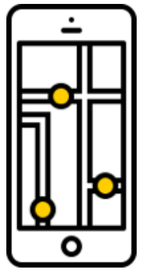
Клиент-серверная архитектура

Мне нужен список сообщений!



На, держи!

Клиент-серверная архитектура



GET /messages HTTP/1.1

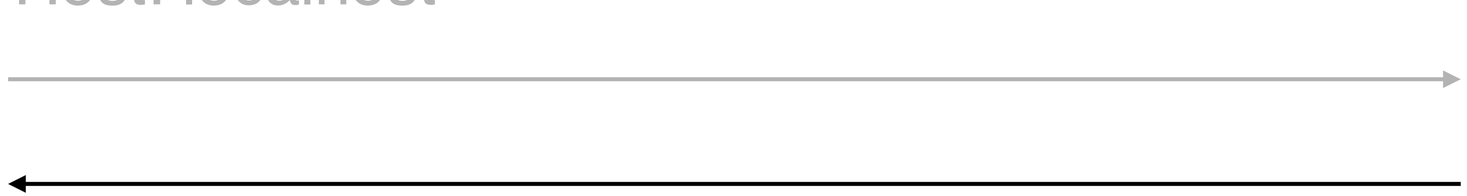
Host: localhost

На, держи!

Клиент-серверная архитектура

GET /messages HTTP/1.1

Host: localhost



HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!”]

Структура протокола

GET /messages HTTP/1.1

Host: localhost

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!’]

Структура протокола

GET /messages HTTP/1.1

Host: localhost

Стартовая строка

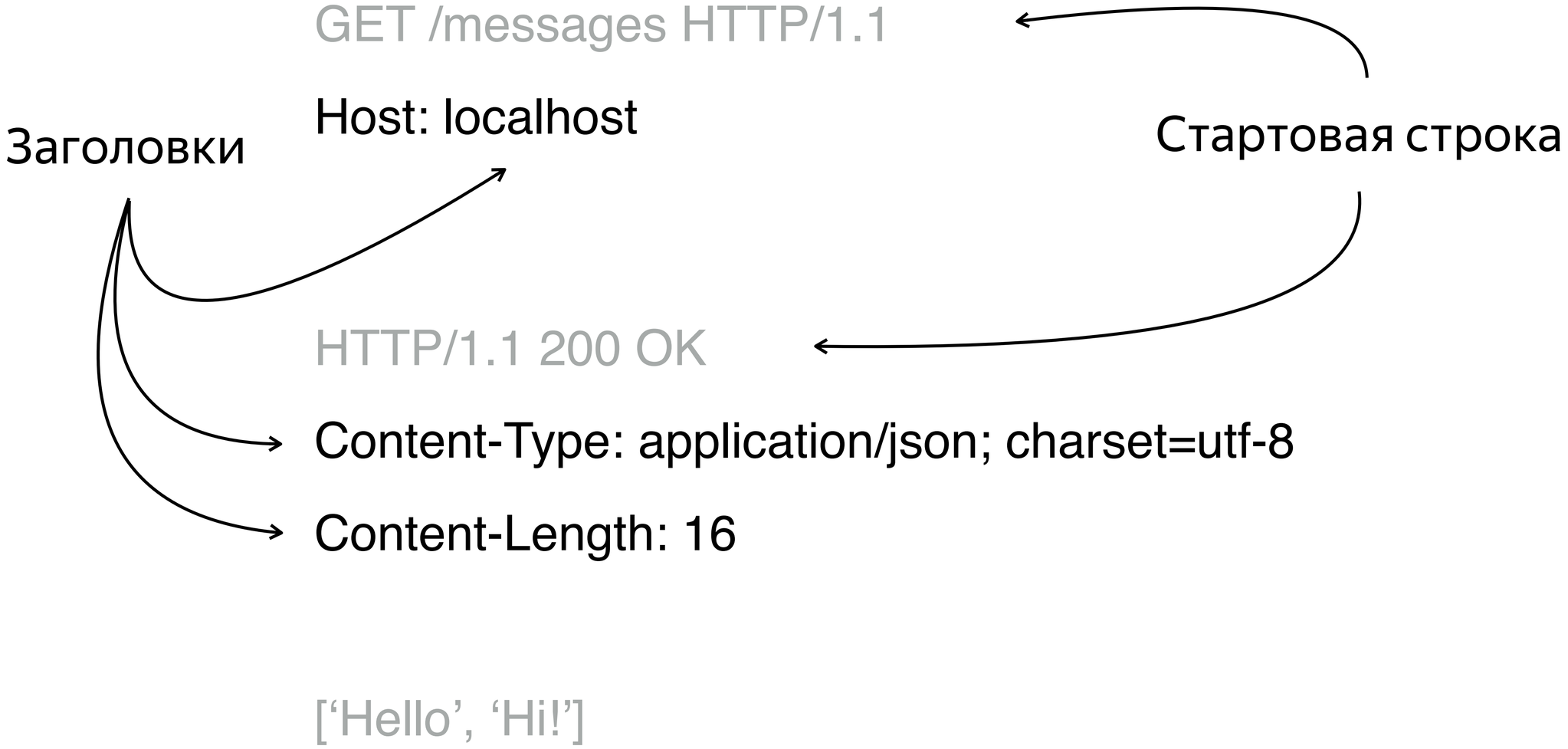
HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

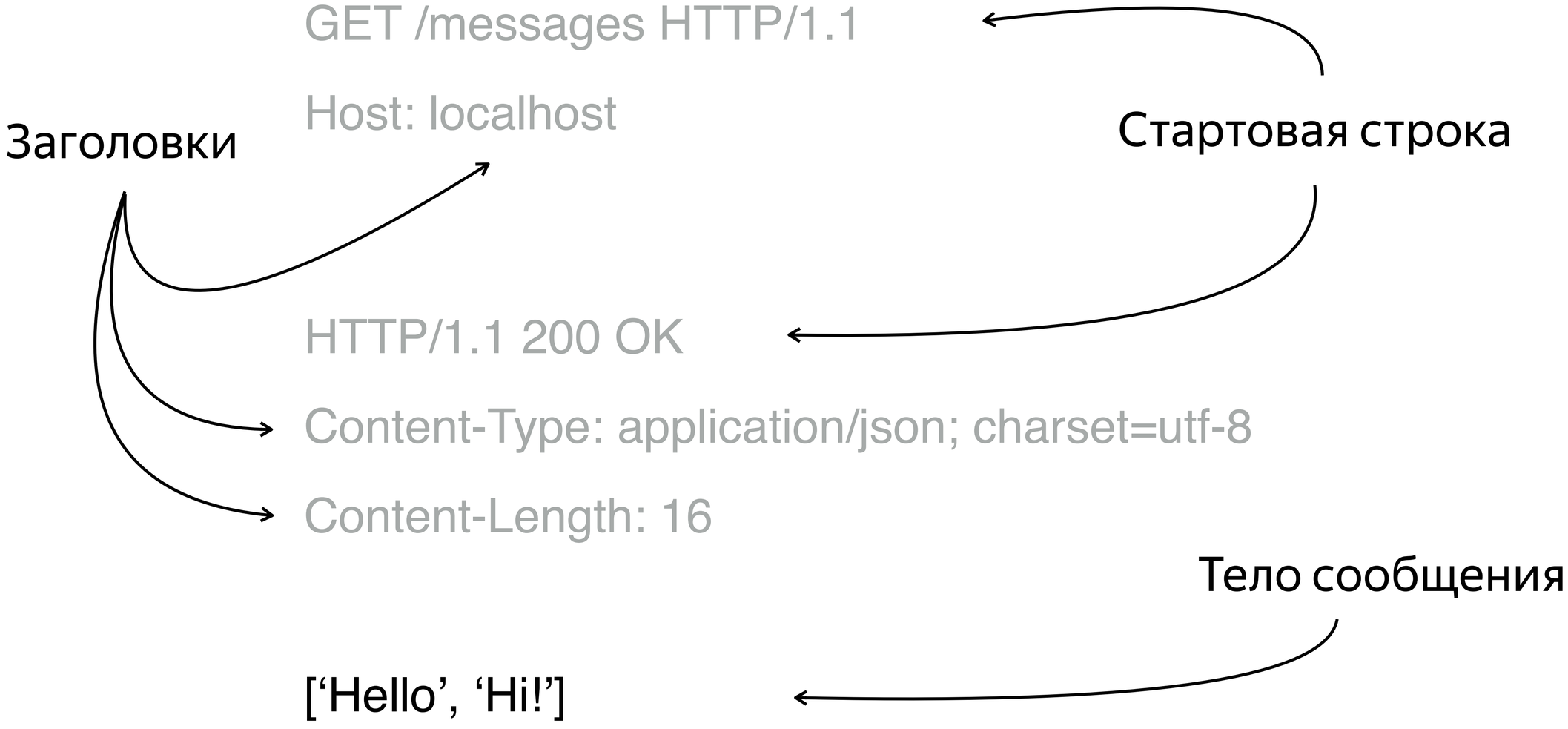
Content-Length: 16

[‘Hello’, ‘Hi!’]

Структура протокола



Структура протокола



Методы запроса и коды ответа

```
GET /messages HTTP/1.1
Host: localhost


HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Content-Length: 16

['Hello', 'Hi!']
```

Методы запроса и коды ответа

GET – read PUT – update

POST – create DELETE – delete

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 16

[‘Hello’, ‘Hi!’]

Методы запроса и коды ответа

GET – read PUT – update

POST – create DELETE – delete

200..<300 – Success

300..<400 – Redirection

400..<500 – Client errors

500..<600 – Server errors

Как загрузить картинку?

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Формирование URL

```
let url = URL(string: "http://localhost/messages")!
```

Формирование URL

```
var urlComponents = URLComponents()

urlComponents.scheme = "http"
urlComponents.host = "localhost"
urlComponents.path = "/messages"
urlComponents.queryItems = [
    URLQueryItem(name: "user", value: "Roman Paradeev"),
    URLQueryItem(name: "limit", value: "10"),
]

let url = urlComponents.url!
// http://localhost/messages?user=Roman%20Paradeev&limit=10
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

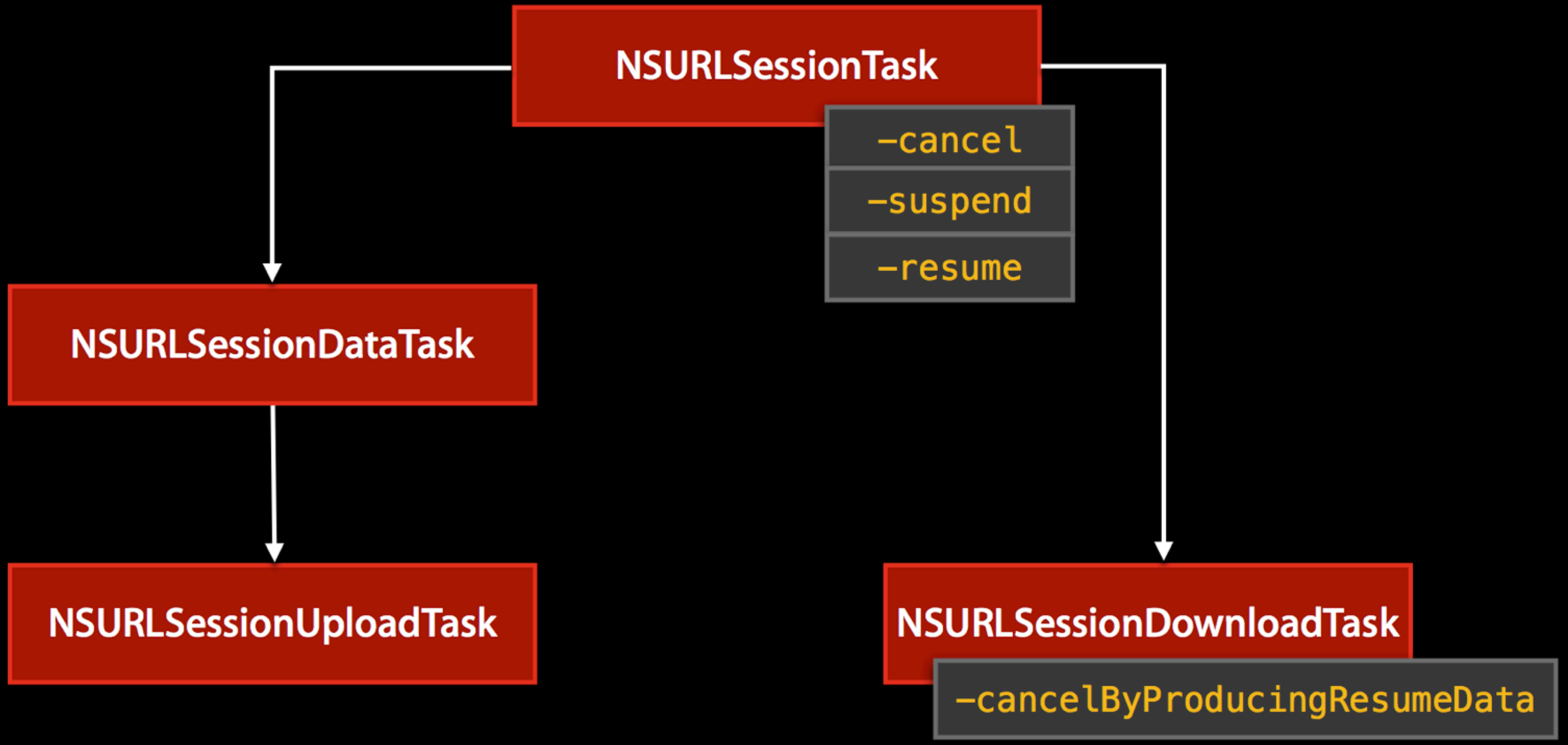
Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

Создание задачи

```
let task = URLSession.shared.dataTask(with: url) {
    data, response, error in

    DispatchQueue.main.async {
        self.label.text = "ok"
    }
}

task.resume()
```

NSURLSessionTask



Обработка ошибок

```
if let error = error {
    print(error.localizedDescription)
    return
}

if let response = response as? HTTPURLResponse {
    switch response.statusCode {
    case 200..<300: break
    default:
        print("Status: \(response.statusCode)")
    }
}
```

Обработка ошибок: ошибки соединения

```
if let error = error {
    print(error.localizedDescription)
    return
}

if let response = response as? HTTPURLResponse {
    switch response.statusCode {
    case 200..<300: break
    default:
        print("Status: \(response.statusCode)")
    }
}
```

Обработка ошибок: ошибки приложения

```
if let error = error {
    print(error.localizedDescription)
    return
}

if let response = response as? HTTPURLResponse {
    switch response.statusCode {
    case 200..<300: break
    default:
        print("Status: \(response.statusCode)")
    }
}
```

Application Transport Security

- Политика безопасности, требующая работать с сетевыми ресурсами по защищённому соединению (HTTPS)
- Запрещает устаревшие и небезопасные протоколы и алгоритмы шифрования
- Обязательна к соблюдению (почти)
- Гибко настраивается

Конфигурация ATS в plist

NSAppTransportSecurity

- NSAllowsArbitraryLoads
- NSAllowsArbitraryLoadsForMedia
- NSAllowsArbitraryLoadsInWebContent
- NSAllowsLocalNetworking
- NSExceptionDomains

Как сделать POST-запрос?

HTTP-методы

GET – чтение

POST – создание

PUT – изменение

DELETE – удаление

OPTIONS, HEAD, PATCH, …

Создание ресурса

POST http://localhost/api/message

Host: localhost

Тело запроса

POST http://localhost/api/message

Host: localhost

Content-Type: text/plain

Hello world

Тело запроса

POST http://localhost/api/message

Host: localhost

Content-Type: application/x-www-form-urlencoded

text=Hello%20world

Тело запроса

POST http://localhost/api/message

Host: localhost

Content-Type: application/json

{"message": "Hello world"}

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Создание HTTP-запроса

```
var request = URLRequest(url: url)

request.httpMethod = "POST"

request.setValue("application/json",
                 forHTTPHeaderField: "Content-Type")

request.httpBody = try! JSONSerialization.data(
                   withJSONObject: ["message": "Hello"])
```

Создание задачи из запроса

```
let url = URL(string: "http://localhost:8080/message")!
var request = URLRequest(url: url)

let task = session.dataTask(with: request) {
    data, response, error in

    if error != nil {
        print("ok")
    }
}

task.resume()
```

Как настроить кэширование?

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

URLSessionConfiguration

- Cache, Cookies, Credential stores
- Cell usage, network service type
- Number of connections
- Resource and network timeouts
- TLS protocols
- HTTP proxies, cookies, pipelining, headers
- Protocol handlers

Типовые конфигурации

- URLSessionConfiguration.default
  - Настройки по умолчанию
- URLSessionConfiguration.ephemeral
  - Cache, Credentials и Cookie хранятся в памяти
- URLSessionConfiguration.background(withIdentifier: String)
  - Для запросов в фоновом режиме

Создание конфигурации

```
let config = URLSessionConfiguration.default

config.urlCache = URLCache(memoryCapacity: 100 * 1024 * 1024,
                           diskCapacity: 100 * 1024 * 1024,
                           diskPath: nil)

config.requestCachePolicy = .returnCacheDataElseLoad
```

Политики кэширования

- useProtocolCachePolicy
  - Согласно HTTP-заголовкам
- reloadIgnoringLocalCacheData
  - Только сеть
- returnCacheDataElseLoad
  - Кэш, потом сеть
- returnCacheDataDontLoad
  - Только кэш

Создание сессии

```
lazy var session: URLSession = {
    let configuration = URLSessionConfiguration.default

    configuration.requestCachePolicy =
                                .returnCacheDataElseLoad

    return URLSession(configuration: configuration)
}()
```

Как управлять таймаутами и отменять запросы?

Как реализовать аутентификацию?

Basic Authorization

Basic Authorization

GET /secret HTTP/1.1

Host: localhost

Basic Authorization

GET /secret HTTP/1.1

Host: localhost

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Basic realm="example"

Basic Authorization

GET /secret HTTP/1.1

Host: localhost

HTTP/1.1 401 Unauthorized

WWW-Authenticate: Basic realm="example"

GET /secret HTTP/1.1

Host: localhost

Authorization: Basic YWRtaW46cGFzc3dvcmQ=

Basic Authorization

```
GET /secret HTTP/1.1
Host: localhost
```

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic realm="example"
```

```
GET /secret HTTP/1.1
Host: localhost
Authorization: Basic YWRtaW46cGFzc3dvcmQ=
```

```
HTTP/1.1 200 OK
```

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Работа с URLSession

1. Сформировать URL (URL)
2. Создать HTTP-запрос (URLRequest)
3. Настроить делегата (URLSessionDelegate, ...)
4. Создать конфигурацию (URLSessionConfiguration)
5. Создать сессию (URLSession)
6. Создать задачу (URLSessionTask, ...)

Настройка делегата

```
extension ViewController: URLSessionDataDelegate {
    func urlSession(
        _ session: URLSession,
        task: URLSessionTask,
        didReceive challenge: URLAuthenticationChallenge,
        completionHandler: (AuthChallengeDisposition,
                            URLCredential?) -> Void) {

        completionHandler(.useCredential, credential)
    }
}
```

Настройка делегата

```
extension ViewController: URLSessionDataDelegate {
    func urlSession(
        _ session: URLSession,
        task: URLSessionTask,
        didReceive challenge: URLAuthenticationChallenge,
        completionHandler: (AuthChallengeDisposition,
                            URLCredential?) -> Void) {

        completionHandler(.useCredential, credential)
    }
}
```

Настройка делегата

```
extension ViewController: URLSessionDataDelegate {
    func urlSession(
        _ session: URLSession,
        task: URLSessionTask,
        didReceive challenge: URLAuthenticationChallenge,
        completionHandler: (AuthChallengeDisposition,
                            URLCredential?) -> Void) {

        completionHandler(.useCredential, credential)
    }
}
```

completionHandler(AuthChallengeDisposition)

- performDefaultHandling
  - Сделать вид, что ничего не происходит
- cancelAuthenticationChallenge
  - Отказаться от этой идеи
- useCredential
  - Указать реквизиты доступа

Создание сессии

```
extension ViewController: URLSessionDataDelegate {
    lazy var session: URLSession = {
        return URLSession(configuration: .default,
                          delegate: self,
                          delegateQueue: .main)
    }()
}
```

Создание задачи

```
let url = URL(string: "http://localhost:8080/secret")!
let task = session.dataTask(with: url)

task.resume()
```

Как догрузить большой файл в фоне?

Background Sessions

Overview

Why use background sessions?

Background sessions in app extensions

Discretionary networking

Using background sessions properly

- Handling app launches
- Data tasks
- Pitfalls and best practices

Why Use Background Sessions?

Uploads and downloads continue while app isn't running

- App can be suspended or crash
- App will be woken up to handle auth and completion

We monitor the environment for you

- Network reachability and connectivity
- Automatic retry after network failures
- Battery monitoring
- Bandwidth monitoring

Background Sessions in App Extensions

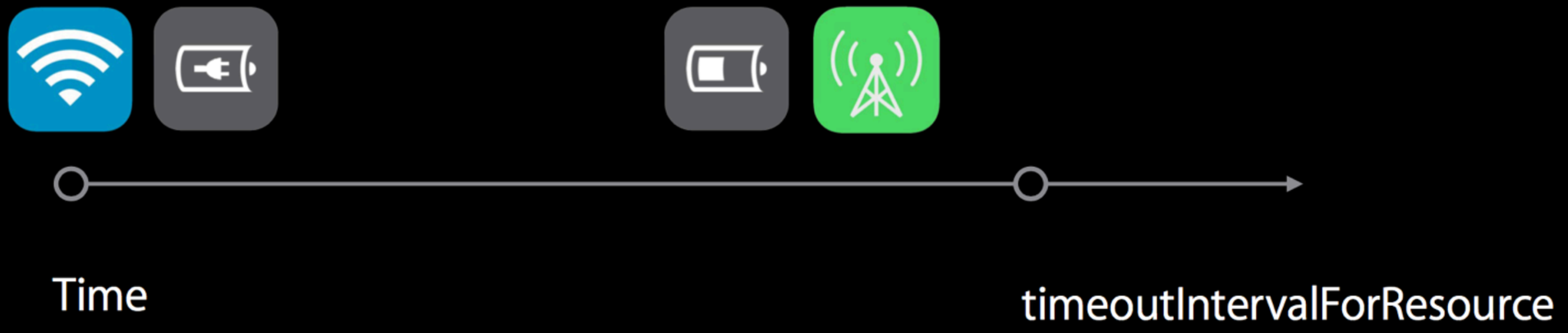
Specifying a shared container


```
NSURLSessionConfiguration *config =
    [NSURLSessionConfiguration backgroundSessionConfigurationWithIdentifier:
                                @“com.mycompany.myapp.bgsession”];

config.sharedContainerIdentifier = @“com.mycompany.mysharedcontainer”;

NSURLSession *session = [NSURLSession sessionWithConfiguration:config
                                                      delegate:self
                                                 delegateQueue:nil];
```

Discretionary Networking

Tasks treated with more urgency as time goes on

- May restrict to WiFi and plugged in to power source at first
- Constraints will relax as timeoutIntervalForResource approaches



Discretionary Networking

Opt-in on NSURLSessionConfiguration:

```
config.discretionary = YES
```

Non-user-initiated tasks

- Pre-fetching the next episode
- Upload syncing

Tasks created while app is running in the background are automatically discretionary

- Work performed during background fetch/push is not user-initiated
- Tasks become non-discretionary when user brings the app to the foreground

Using Background Sessions

Handling app launching

UIApplicationDelegate method:

`-application:handleEventsForBackgroundURLSession:completionHandler:`

Reconnect to the background session

- Create background session with the provided identifier
- Receive delegate messages
- Call the completionHandler when finished handling the events

`-URLSessionDidFinishEventsForBackgroundURLSession:`

Using Background Sessions

Data tasks

Will only run while app is running

- Will fail with `NSURLErrorBackgroundSessionWasDisconnected` when app is suspended or exits

Can convert to download task when response is received

- Will continue after app is suspended

```
- (void)URLSession:(NSURLSession *)session dataTask:(NSURLSessionDataTask
*)dataTask didReceiveResponse:(NSURLResponse *)response completionHandler:
(void (^)(NSURLSessionResponseDisposition disposition))completionHandler;
```

`NSURLSessionResponseBecomeDownload`

Using Background Sessions

Things to avoid

Creating one task at a time

- Tasks created in background will be discretionary
- The system will prevent your app from being launched too frequently

Downloading lots of small assets

- Much more efficient to download one large, zipped asset

Blocking while waiting for transfers to complete

Using Background Sessions

Best practices

Can still use in-process networking while running in the background

- Smaller, time-sensitive assets
- Larger uploads/downloads should use background sessions

Support resumable downloads (Range GET requests)

Handle launch events properly

- Reconnect to your background session when we launch your app
- Call the completion handler

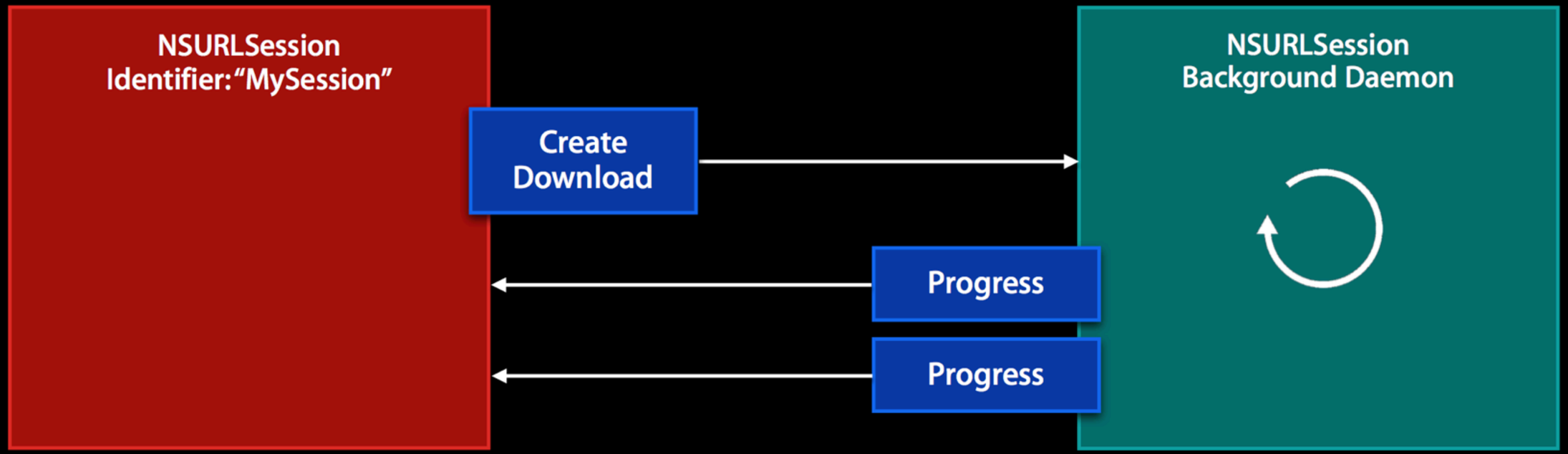
NSURLSession

Background Transfers

- Supports upload and download using HTTP(S)
- Requires a delegate for event delivery
  - Uses same Upload and Download task delegates as in-process
- Redirections are always taken
- `–discretionary` configuration property
  - Available on iOS, only applies to background transfers
  - Optimizes for power and network

NSURLSession

Background Transfers



NSURLSession

Out-of-process Transfers

- Delegate messages received while you're running
- Your app will be launched in the background…
  - to service auth requests
  - when all tasks complete
- Creating a session from same identifier "reconnects" you to existing background session

```
-(void) getTasksWithCompletionHandler:^(NSArray* dataTasks,
                                        NSArray* uploadTasks,
                                        NSArray* downloadTasks)
                                        completion;
```

Сегодня мы узнали

1. Как сформировать URL и создать HTTP-запрос
2. Какие возможности предоставляет конфигурация сессии
3. Как работать с кэшом и авторизацией
4. Как обращаться с фоновой сессией

Ссылки

Демо-приложение и демо-сервер

- https://github.com/sameoldmadness/urlsession-example-server
- https://github.com/sameoldmadness/urlsession-example-app

Отладка

- http://docs.mitmproxy.org/en/latest/mitmproxy.html

Ссылки

Сессии WWDC

- https://developer.apple.com/videos/play/wwdc2016/711/
- https://developer.apple.com/videos/play/wwdc2015/711/
- https://developer.apple.com/videos/play/wwdc2014/707/
- https://developer.apple.com/videos/play/enterprise/707/

Спасибо!