Article

Fetching website data into memory

Receive data directly into memory by creating a data task from a URL session.

Overview

For small interactions with remote servers, you can use the `URLSessionDataTask` class to receive response data into memory (as opposed to using the `URLSessionDownloadTask` class, which stores the data directly to the file system). A data task is ideal for uses like calling a web service endpoint.

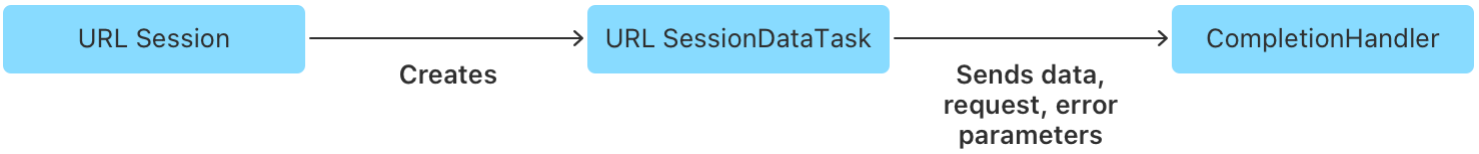
You use a URL session instance to create the task. If your needs are fairly simple, you can use the `shared` instance of the `URLSession` class. If you want to interact with the transfer through delegate callbacks, you'll need to create a session instead of using the shared instance. You use a `URLSessionConfiguration` instance when creating a session, also passing in a class that implements `URLSessionDelegate` or one of its subprotocols. Sessions can be reused to create multiple tasks, so for each unique configuration you need, create a session and store it as a property.

> **Note**
>
> Be careful to not create more sessions than you need. For example, if you have several parts of your app that need a similarly configured session, create one session and share it among them.

Once you have a session, you create a data task with one of the `dataTask()` methods. Tasks are created in a suspended state, and can be started by calling `resume()`.

Receive results with a completion handler

The simplest way to fetch data is to create a data task that uses a completion handler. With this arrangement, the task delivers the server's response, data, and possibly errors to a completion handler block that you provide. shows the relationship between a session and a task, and how results are delivered to the completion handler.



To create a data task that uses a completion handler, call the `dataTask(with:)` method of `URLSession`. Your completion handler needs to do three things:

1. Verify that the `error` parameter is `nil`. If not, a transport error has occurred; handle the error and exit.
2. Check the `response` parameter to verify that the status code indicates success and that the MIME type is an expected value. If not, handle the server error and exit.
3. Use the `data` instance as needed.

The following example shows a `startLoad()` method for fetching a URL's contents. It starts by using the `URLSession` class's shared instance to create a data task that delivers its results to a completion handler. After checking for local and server errors, this handler converts the data to a string, and uses it to populate a `WKWebView` outlet. Of course, your app might have other uses for fetched data, like parsing it into a data model.

Creating a completion handler to receive data-loading results

```
func startLoad() {
    let url = URL(string: "https://www.example.com/")!
    let task = URLSession.shared.dataTask(with: url) { data, response, error in
        if let error = error {
            self.handleClientError(error)
            return
        }
        guard let httpResponse = response as? HTTPURLResponse,
            (200...299).contains(httpResponse.statusCode) else {
            self.handleServerError(response)
            return
        }
        if let mimeType = httpResponse.mimeType, mimeType == "text/html",
            let data = data,
            let string = String(data: data, encoding: .utf8) {
            DispatchQueue.main.async {
                self.webView.loadHTMLString(string, baseURL: url)
```

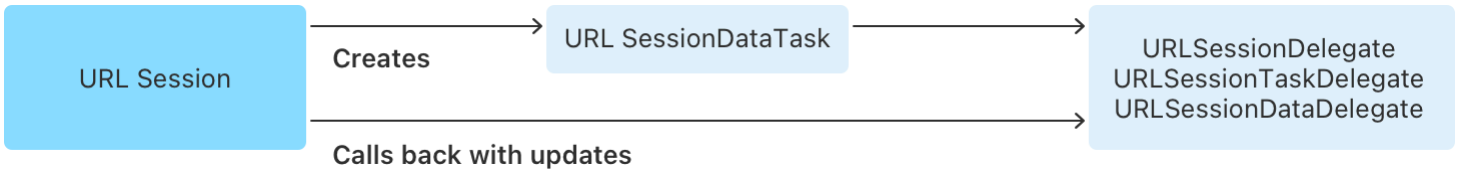
```
            }
        }
    }
    task.resume()
}
```

Important

The completion handler is called on a different Grand Central Dispatch queue than the one that created the task. Therefore, any work that uses `data` or `error` to update the UI — like updating `webView` — should be explicitly placed on the main queue, as shown here.

Receive transfer details and results with a delegate

For a greater level of access to the task's activity as it proceeds, when creating the data task, you can set a delegate on the session, rather than providing a completion handler. shows this arrangement.



With this approach, portions of the data are provided to the `urlSession(_:dataTask:didReceive:)` method of `URLSessionDataDelegate` as they arrive, until the transfer finishes or fails with an error. The delegate also receives other kinds of events as the transfer proceeds.

You need to create your own `URLSession` instance when using the delegate approach, rather than using the `URLSession` class's simple `shared` instance. Creating a new session allows you to set your own class as the session's delegate, as shown in the following example.

Declare that your class implements one or more of the delegate protocols (`URLSessionDelegate`, `URLSessionTaskDelegate`, `URLSessionDataDelegate`, and `URLSessionDownloadDelegate`). Then create the URL session instance with the initializer `init(configuration:delegate:delegateQueue:)`. You can customize the configuration instance used with this initializer. For example, it's a good idea to set `waitsForConnectivity` to `true`. That way, the session waits for suitable connectivity, rather than failing immediately if the required connectivity is unavailable.

Creating a URLSession that uses a delegate

```
private lazy var session: URLSession = {
    let configuration = URLSessionConfiguration.default
    configuration.waitsForConnectivity = true
    return URLSession(configuration: configuration,
                      delegate: self, delegateQueue: nil)
}()
```

The following example shows a `startLoad()` method that uses this session to start a data task, and uses delegate callbacks to handle received data and errors. This listing implements three delegate callbacks:

- `urlSession(_:dataTask:didReceive:completionHandler:)` verifies that the response has a succesful HTTP status code, and that the MIME type is `text/html` or `text/plain`. If either of these is not the case, the task is canceled; otherwise, it's allowed to proceed.
- `urlSession(_:dataTask:didReceive:)` takes each `Data` instance received by the task and appends it to a buffer called `receivedData`.
- `urlSession(_:task:didCompleteWithError:)` first looks to see if a transport-level error has occurred. If there is no error, it attempts to convert the `receivedData` buffer to a string and set it as the contents of `webView`.

Using a delegate with a URL session data task

```
var receivedData: Data?

func startLoad() {
    loadButton.isEnabled = false
    let url = URL(string: "https://www.example.com/")!
    receivedData = Data()
    let task = session.dataTask(with: url)
    task.resume()
}

// delegate methods

func urlSession(_ session: URLSession, dataTask: URLSessionDataTask, didReceive resp
                completionHandler: @escaping (URLSession.ResponseDisposition) -> Voi
    guard let response = response as? HTTPURLResponse,
        (200...299).contains(response.statusCode),
        let mimeType = response.mimeType,
        mimeType == "text/html" else {
        completionHandler(.cancel)
```

```
            return
        }
        completionHandler(.allow)
    }

    func urlSession(_ session: URLSession, dataTask: URLSessionDataTask, didReceive data
        self.receivedData?.append(data)
    }

    func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError er
        DispatchQueue.main.async {
            self.loadButton.isEnabled = true
            if let error = error {
                handleClientError(error)
            } else if let receivedData = self.receivedData,
                let string = String(data: receivedData, encoding: .utf8) {
                self.webView.loadHTMLString(string, baseURL: task.currentRequest?.url)
            }
        }
    }
```

The various delegate protocols offer methods beyond those shown in the above code, for handling authentication challenges, following redirects, and other special cases. `Using a URL Session`, in the `URLSession` discussion, describes the various callbacks that may occur during a transfer.

See Also

Essentials

Analyzing HTTP traffic with Instruments

Measure HTTP-based network performance and usage of your apps.

`class` `URLSession`

An object that coordinates a group of related, network data transfer tasks.

`class` `URLSessionTask`

A task, like downloading a specific resource, performed in a URL session.