URL Loading System

URL Loading System

- URL과 상호작용하고 표준 인터넷 프로토콜을 통해 서버와 통신하기 위한 시스템

  (URL로 참조되는 콘텐츠에 접근할 수 있게 해주는 클래스 및 프로토콜 집합)

- Foundation Framework
- 프로토콜 지원: FTP, HTTP, HTTPS, FILE, DATA, and CUSTOM
- 주요 클래스: URLSession, URL, URLSessionTasks, URLRequest, URLResponse 등

URLSession

- 네트워크 데이터 전송 작업에 관련된 클래스 그룹을 조정하는 객체 (네트워크용 API)
- 대부분의 네트워킹 API 와 마찬가지로 URLSession API 역시 비동기 동작
- URL data 또는 파일을 가져오는 2가지 기본적인 접근 방법

  > 간단한 요청: URLSession에 URL만 전달하여 데이터를 가져오거나, 파일을 다운로드

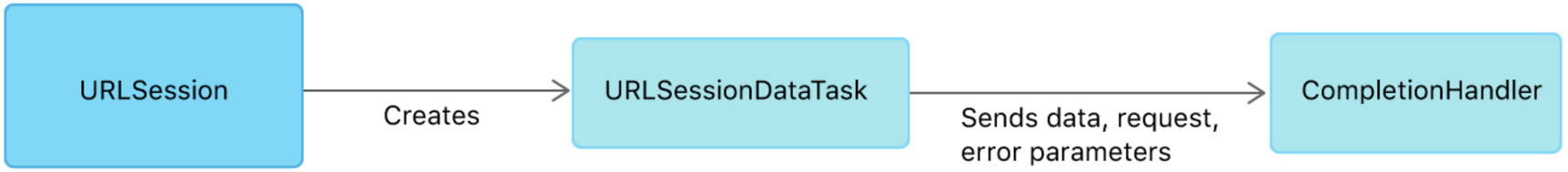
  > 복잡한 요청: 데이터 업로드 처럼 추가 정보가 필요한 요청은 URLRequest 객체를 URLSession에 제공
- URLSessionConfiguration에 의해 세션 동작 결정
- Data task, Download task, Upload tasks, Stream task 4가지 유형의 작업 지원
- HTTP/1.1, SPDY, and HTTP/2 지원

Network API Lifecycle

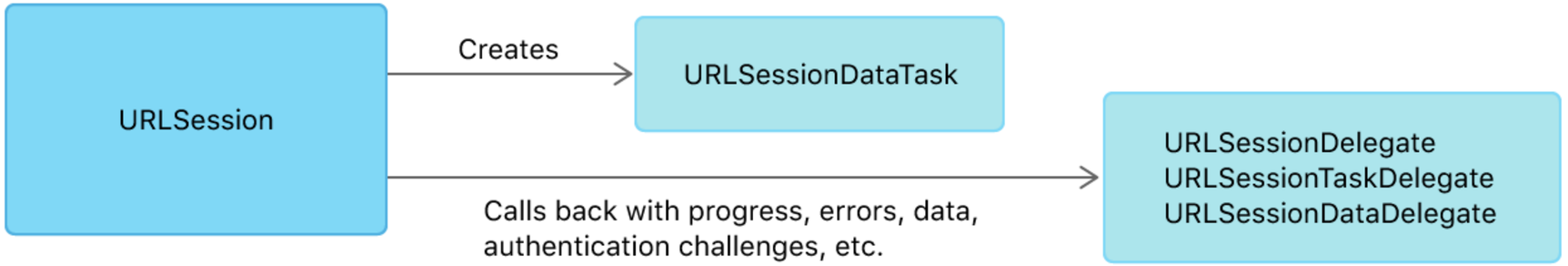
- SessionConfiguration을 설정하고 Session 생성
- URL 또는 URLRequest 객체 생성
- 사용 목적에 맞는 Task를 결정하고, 이에 해당하는 Completion Handler 파라미터 전달 또는 Delegate 메서드 구현
- 생성한 Session을 통해 Task 수행
- 작업 완료 후 미리 설정한 Completion Handler 또는 Delegate 메서드가 호출되고 실행 종료

Receive results from a task

- 요청에 대한 응답을 처리하기 위해 completion handler와 delegate 중 선택
- Complection handler 구현 시 delegate 메서드는 호출되지 않음
- 비교적 간단한 응답 처리는 completion handler를 이용하고 세세한 처리가 필요할 때는 delegate 구현



[Completion Handler]



[Delegate]

URLSessionConfiguration

- URLSession이 데이터 업로딩/다운로딩을 수행할 때 사용되는 정책과 동작을 정의하기 위한 설정 객체
- URLSession 사용을 위한 첫 번째 단계
- 시간제한, 캐싱 정책, Cellular 제한 여부, 연결 요구사항 등 URLSession에서 사용할 여러 유형의 정보를 설정
- URLSession 생성 시 Configuration 객체의 설정을 복사하는 형태이므로, 그 이후의 설정 변경은 반영되지 않음

  새로운 설정을 적용해야 한다면 새로운 세션 객체를 만들어서 사용
- URLRequest의 속성에서 개별 설정을 덮어쓸 수가 있으나 SessionConfiguration의 정책이 더 제한적일 경우 불가

  e.g. Cellular 허용 여부 - Configuration: 제한(X) / URLRequest: 허용(O) 일 때, 해당 세션 Cellular 네트워킹 불가

Types of Session Configurations

- URLSession의 동작과 기능은 대부분 세션 설정 유형에 따라 결정됨.

- Shared session (URLSession.shared)

  > 별도의 Configuration 객체나 Delegate를 사용하지 않는 싱글톤 객체

  > 일반적으로 이 세션에서는 설정을 변경하지 않고 사용하며, 커스터마이징이 필요한 경우 Default session 이용

  > 기본 설정만으로 충분한 간단한 요청에 적합

- Default session (URLSessionConfiguration.default)

  > 기본 세션. 설정을 특별히 변경하지 않는 이상 공유 세션과 매우 유사하게 동작하나 Delegate 메서드 이용 가능

  > 디스크에 캐시(파일 형태로 다운받은 것 제외)와 쿠키를 저장하며, 자격 증명(credential)은 유저 키체인에 저장

Types of Session Configurations

- Ephemeral session (URLSessionConfiguration.ephemeral)

  > 임시 세션은 기본 세션과 비슷하지만, caches, cookies, credential 등 어떤 데이터도 디스크에 기록하지 않음

  > 메모리(RAM)에만 저장하여 사용하다가 앱이 세션을 무효화하면 모든 세션 데이터가 자동으로 제거

  > 메모리 캐시는 Suspended 상태일 때도 제거되지 않지만, 앱이 종료되거나 메모리가 부족해지면 제거 될 수 있음

  > 별도로 기록해야 할 내용은 파일로 저장

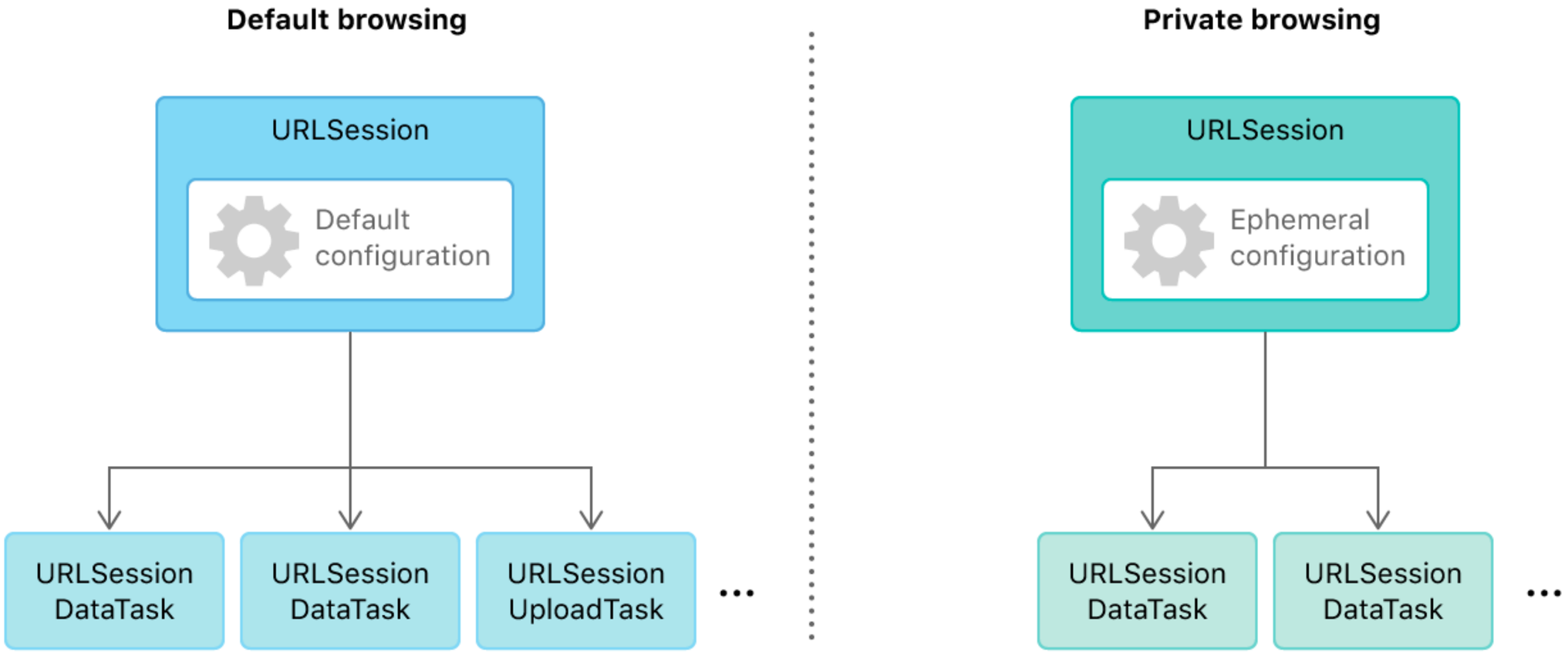
- Background session (URLSessionConfiguration.background(withIdentifier: “MyIdentifier”))

  > 백그라운드 세션은 앱이 실행 중이지 않은 상태에서도 데이터 전송이 가능하도록 설정 가능

  > 별도의 프로세스에서 전송을 처리하도록 시스템에 전송 제어권을 전달

  > 앱 종료나 재실행 시, 이전 종료 시점의 전송 상태와 세션 정보를 identifier를 통해 구분하고 재생성. 단, 시스템에 의한 일반적인 앱 종료시에만 이와 같이 동작. 멀티태스킹 화면에서 유저에 의한 강제 종료시에는 모든 백그라운드 작업이 취소되며, 유저가 다시 앱을 실행하기 전까지는 시스템이 자동으로 앱을 재실행하지 않음

Creating tasks from URL sessions



Types of Tasks

- URLSessionTask

  > URLSession의 Task 수행을 위한 베이스 클래스로서 URLSession에서 메서드를 호출하여 생성

  > Task 생성 후 resume() 메서드를 통해 작업 수행

  > URLSession이 요청 작업이 종료되거나 실패할 때까지 Task에 대한 강한 참조 유지

[URLSessionTask의 서브 클래스]

- URLSessionDataTask - Data를 주고 받기 위한 Task, 백그라운드 작업 미지원
- URLSessionUploadTask - 주로 파일 형태로 업로드하기 위한 Task, 백그라운드 업로드 지원
- URLSessionDownloadTask - 파일을 바로 디스크로 다운받기 위한 Task, 백그라운드 작업 지원
- URLSessionStreamTask - TCP/IP 연결을 통해 지속적인 데이터 교환을 하기 위한 Task

URLSessionDownloadTask

- 다운 받은 파일은 temporary 디렉토리에 위치하며 캐시로 저장해두려면 별도의 경로에 복사 또는 이동

```
let downloadTask = URLSession.shared.downloadTask(with: url) {
    urlOrNil, responseOrNil, errorOrNil in
    // check for and handle errors:
    // * errorOrNil should be nil
    // * responseOrNil should be an HTTPURLResponse with statusCode in 200..<299

    guard let fileURL = urlOrNil else { return }
    do {
        let documentsURL = try
            FileManager.default.url(for: .documentDirectory,
                                    in: .userDomainMask,
                                    appropriateFor: nil,
                                    create: false)
        let savedURL = documentsURL.appendingPathComponent(
            fileUrl.lastPathComponent)
        try FileManager.default.moveItem(at: fileUrl, to: savedURL)
    } catch {
        print ("file error: \(error)")
    }
}
downloadTask.resume()
```

Resuming Downloads

- 다운로드 취소 또는 실패 시 중간부터 재개 가능

```
downloadTask.cancel { resumeDataOrNil in
    guard let resumeData = resumeDataOrNil else {
      // download can't be resumed; remove from UI if necessary
      return
    }
    self.resumeData = resumeData
}
```

[Cancel]

```
func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError
    guard let error = error else {
        // Handle success case.
        return
    }
    let userInfo = (error as NSError).userInfo
    if let resumeData = userInfo[NSURLSessionDownloadTaskResumeData] as? Data {
        self.resumeData = resumeData
    }
    // Perform any other error handling.
}
```

[Fail]

[Resume 가능 조건]

- 자원이 처음 요청했을 때 이후로 미변경
- HTTP / HTTPS의 GET 메서드 요청
- 서버가 ETag 또는 Last-Modified 헤더를 응답에 포함해서 제공해야 함
- 서버가 바이트 범위(byte-range) 요청을 제공해야 함
- 저장해 둔 임시 파일이 지워지지 않은 상태

URLCache

- URL 요청을 CachedURLResponse 객체에 매핑하여 응답에 대한 캐싱을 구현하기 위한 객체
- 네트워크 자원에 대한 응답에 대해 캐싱을 수행하며 기본값으로 shared 싱글톤 객체를 사용하거나 직접 생성 가능
- 4 종류의 캐시 정책에 따라 각각 다르게 작업을 수행하며, SessionConfiguration 또는 URLRequest 에서 설정
- 캐시를 위한 최대 디스크 용량과 메모리 용량을 지정할 수 있으며, 기본 디렉토리 이외에 별도 저장 공간 설정 가능
- iOS에서 디스크 캐시는 디스크 용량이 부족해지면 시스템에 의해 삭제 가능 (앱이 실행중이지 않을 때 한정)
- 데이터의 크기가 디스크 캐시 크기의 5%보다 작아야 함

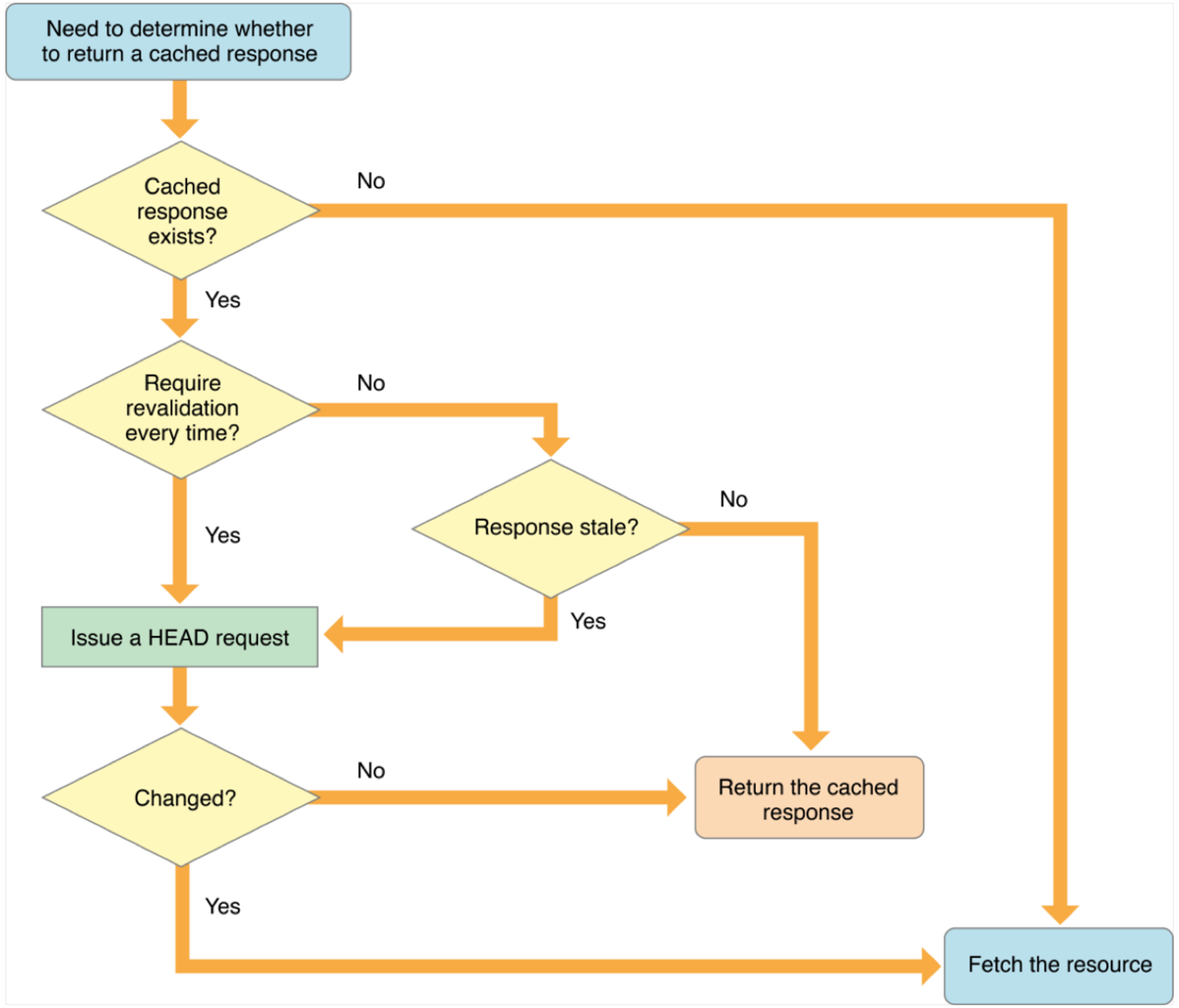
In iOS, the on-disk cache may be purged when the system runs low on disk space, but only when your app is not running.

Cache policies and behaviors

- reloadIngoringLocalCacheData - 캐시 파일은 무시하고 항상 원본 소스에 접근
- returnCacheDataDontLoad - 오프라인 모드와 유사. 캐시 파일이 존재할 경우에만 데이터 반환
- returnCacheDataElseLoad - 캐시 파일 우선, 없을 경우 원본 소스에 접근
- useProtocolCachePolicy - 각 프로토콜별 정책에 따름

Cache policy	Local cache	Originating source
`NSURLRequest.CachePolicy.reloadIgnoringLocalCacheData`	Ignored	Accessed exclusively
`NSURLRequest.CachePolicy.returnCacheDataDontLoad`	Accessed exclusively	Ignored
`NSURLRequest.CachePolicy.returnCacheDataElseLoad`	Tried first	Accessed only if needed
`NSURLRequest.CachePolicy.useProtocolCachePolicy`	Depends on protocol	Depends on protocol

HTTP Cacheing Behavior



[UseProtocolCachePolicy Decision Tree]

- URL 요청에 대한 캐시 응답이 없으면 원본 소스에 요청
- 캐시 유효성 검증을 매번 할 필요가 없고, 아직 캐시 만료일이 지나지 않았다면 캐시 데이터를 반환
- 캐시된 응답이 오래 되었거나 유효성을 재검증해야 하면 URL Loading System이 원본 소스에 HEAD 요청을 보내 자원 변경 여부를 확인 변경이 있으면 새로 가져오고 그렇지 않으면 캐시 데이터 반환

Helper Classes

- 요청과 응답에 대한 추가 메타데이터 제공을 위해 두 개의 헬퍼 클래스(URLRequest, URLResponse) 제공
- URLRequest

  > 프로토콜에 독립적인 방식으로 URL 및 모든 프로토콜 관련 속성을 캡슐화

  > 일부는 프로토콜별 속성 지원. 예를 들어 HTTP는 request body, headers 를 설정하거나 반환하는 메서드 추가

- URLResponse

  > 서버의 응답은 컨텐츠를 설명하는 메타데이터(헤더) + 컨텐츠(바디)로 나눌 수 있음

  > 이 중 메타데이터만 URLResponse 객체에 저장되고 컨텐츠는 별도로 처리

  > 메타데이터는 공통으로 MIME, 예상 컨텐츠 길이, 텍스트 인코딩, 응답 URL로 구성되며 URLResponse에 의해 캡슐화

  > 프로토콜별 하위 클래스는 추가 메타 데이터 제공 가능 e.g. HTTPURLResponse - header, statusCode

Completion handler example

```
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()
}
```

[처리 순서]

1. error가 nil인지 확인

   nil이 아닌 경우 전송(transport) 에러
2. statusCode와 mimeType 값 확인
3. data 처리

Delegate example (1)

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

```
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()
}
```

[처리 순서]

1. URLSession의 delegate 객체 설정
2. CompletionHandler 없이 Task 작업 수행
3. Delegate 메서드 구현

Delegate example (2)

```
func urlSession(_ session: URLSession, dataTask: URLSessionDataTask, didReceive r
                completionHandler: @escaping (URLSession.ResponseDisposition) ->
    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 d
    self.receivedData?.append(data)
}

func urlSession(_ session: URLSession, task: URLSessionTask, didCompleteWithError
    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
        }
    }
}
```