DPS923 MAP523 notes – Wed Mar 16 2016

An introduction to accessing the network from an iOS app.

Important things to know before you begin coding

There are some very important things to know before you begin coding network tasks:

Follow

Basic knowledge of HTTP is required. You must know what a 'rec
These are the fundamental building blocks in an app that uses the

Basic knowledge of web services is required. You must know wh
use one. In this course, you do not have to code/create a web servi
You must also be able to understand and use JSON.

Network operations are asynchronous. That means that we do n
will be responded to.

You must learn something about closures. A closure in Swift is a
operations use closures.

Simple tasks are easy to code, but complex tasks require more
one' and 'get all' from a resource. More complex tasks include HTTI
authentication, and so on.

Accessing the network – what is a web service?

A web service is an application that runs on a web server, and is accessed programmatically.

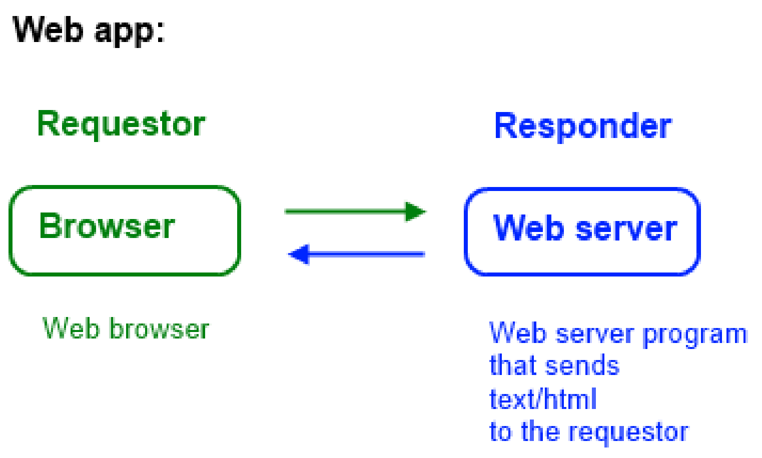
This simple sentence contains a number of very important concepts and techniques, including:

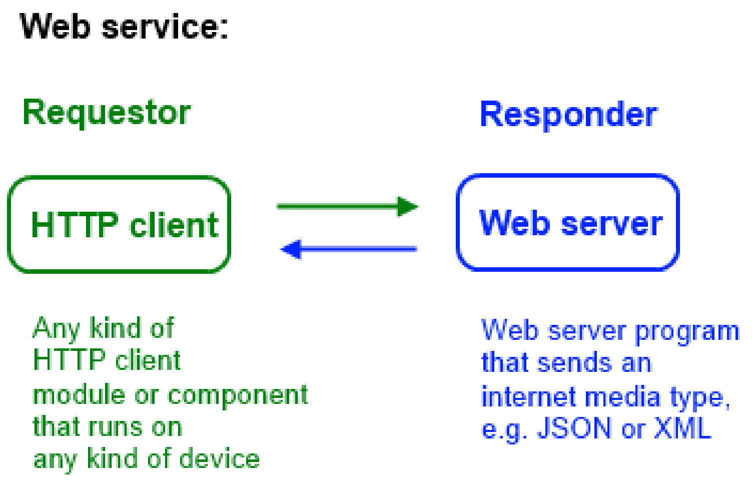
- HTTP is the protocol
- A web service is defined by its application programming interface (API)
- Humans don't use a web service directly – instead, the application they are using creates and sends a request to the web service, and handles the response in a way that's meaningful to the application they're using

.

What's the difference between a web app, and a web service?

These diagrams should help illustrate the difference.




Web service:
Requestor
Responder
HTTP client
Web server
Any kind of HTTP client module or component that runs on any kind of device
Web server program that sends an internet media type, e.g. JSON or XML

.

Give me a brief history lesson on web services

With the rise of the web's use and popularity in the 1990s, efforts were made to define and specify web services.

This led to the *de facto* standardization of SOAP XML web services. Often described as "big web services", SOAP XML web services are the implementation of remote procedure calls on the web. This kind of web service typically has one specific endpoint address, and requestors must create and send a data package (which conforms to SOAP), and then process the response (which also conforms to SOAP).

However, other efforts took advantage of the web and its existing features and benefits. In other words, they simply followed the HTTP specification and its *ex post facto* architecture definition, to create true and pure web services. These kinds of web services, often termed "Web API", exploded in use and popularity from about 2005 onwards, and are now the preferred design approach.

Web services are widely-used by mobile device platforms, iOS and Android. Almost all mobile apps use a web service.

Web services can be developed on any web-connected technology platform, and in any language. Web services are interoperable.

.

Are web services important?

Yes.

Absolutely.

Web services are *vital* for modern software architectures.

For *all* device platforms.

.

How do I start learning web services?

You must learn (more) about HTTP.

In class, we will discuss some HTTP topics, and refer to the overview on Wikipedia, and the official specification series, RFC 7230 to RFC 7235, which recently replaced RFC 2616.

.

Request, response

HTTP is a typical Internet protocol that relies on an exchange of messages between a requestor and a responder. The messages are plain text, and must conform to a specific format.

An example of a request message is shown below. The resource at “/index.html” is being requested:

```
GET /index.html HTTP/1.1
Host: www.google.ca
```

.

An example of the response message is shown below. Some data has been removed to simplify the example. Notice the empty line in between the headers and the message body:

```
HTTP/1.1 200 OK
Date: Sun, 06 Sep 2015 16:29:52 GMT
Content-Type: text/html; charset=ISO-8859-1

<!doctype html>
<html itemscope="" itemtype="http://schema.org/WebPage">
<head>
<meta itemprop="image" content="/images/google_favicon_128.png">
<title>Google</title>
<script>
```

```
(function(){
```

A request to a web service looks similar.

The response looks different - it is not HTML. Instead, it is data. Mostly JSON, in modern web services.

In our Swift programs, we use language features to marshall JSON objects and collections into Swift-native objects (usually dictionaries) and collections (usually arrays).

Getting started, hands-on

If you wish, you can use the ClassesV2 template to rebuild (create) a simple app that uses a web service.

You can get the ClassesV2 template from the GitHub code example repository. Then, perform the project-rename task.

We will use a public web service that your professor team created recently. It is here:

https://ios2016winter.azurewebsites.net

It enables requestors to work with academic program data from the School of ICT. For example, you can obtain a collection of academic "Program" objects, or a collection of academic "Course" objects. The web service is designed to be self-documenting, so you can discover its data and permitted operations.

By itself, a web browser is not a good tool to use to inspect a web service. Instead, use this web app:

http://jsonformatter.curiousconcept.com

Enter a resource URI in its "JSON Data Url" field, and click its "Process" button. The response to each request will be displayed in its own area/box. Try it with these URIs:

http://ios2016winter.azurewebsites.net/api/programs

http://ios2016winter.azurewebsites.net/api/programs/4

https://itunes.apple.com/search?term=big+bang+theory&entity=tvEpisode&limit=10&sort=recent

.

Composing a network request, and handling the response

In iOS (and OS X), we use a 'task' object to compose a network request, and handle the response.

The 'task' object is an instance of NSURLSessionDataTask. It uses a 'session' object, and a 'request' object.

The 'session' object is an instance of NSURLSession, which represents a logical session between your app and a web service. A 'session' object must be configured, using a 'session configuration' object (which is an instance of NSURLSessionConfiguration).

The 'request' object is an instance of NSMutableURLRequest, which needs a 'URL' object, and can be configured with request headers if necessary.

In summary, a 'task' object relies on the presence of a number of other initialized and configured objects.

.

Configuring and executing the 'task' object

When a 'task' object is created, it does not immediately execute. You must execute it, when you are ready, by sending the 'resume' message to the object. The 'task' object executes in the background, so it does not impair the responsiveness of your app's user interface.

More important is the configuration of the 'task' object: You must write a *block of code* that will run when the task completes execution.

This *block of code* is known as a *closure* in Swift. Think of a *closure* as an *inline function*. (You have likely worked with similar constructs in the past, including JavaScript functions (and closures), and C# lambda expressions.)

The *closure* is defined on the 'task' object's *completionHandler* parameter. It exposes these values:

- The data returned in the response body
- Metadata about the response
- If necessary, error information

The data type of the data returned in the response body is NSData. Therefore, we must transform it into the format we expect. In this course, we plan to work with web services that use JSON, so we will transform the NSData object to JSON and then to an array or dictionary, as appropriate for the request.

.

Code example

See the "**ICT WS v1**" code example.

.

Learning resources

URL Loading System Programming Guide (excerpted below…)

.

A deeper look at the networking classes

The following is a summary of the important parts of the URL Loading System Programming Guide, as well as the class reference documents for NSURLSession and others.

> Content copied directly/verbatim from the Apple documentation is blue-coloured.

The URL loading system is a set of classes and protocols that allow your app to access content referenced by a URL.

The most commonly used classes in the URL loading system allow your app to retrieve the content of a URL from the source. In iOS 7 and later or OS X v10.9 and later, NSURLSession is the preferred API for new code that performs URL requests. Old code examples and search engine results will use other classes (NSURLConnection, AFNetworking). While useful and instructive, you should not rely on old code examples.

The URL loading classes use two helper classes that provide additional metadata—one for the request itself (NSURLRequest) and one for the server's response (NSURLResponse).

The NSURLSession class and related classes provide an API for downloading content via HTTP. Like most networking APIs, the NSURLSession API is highly asynchronous. If you use the default, system-provided delegate, you must provide a completion handler block that returns data to your app when a transfer finishes successfully or with an error.

The NSURLSession API supports three types of sessions. We will use the "default session" type.

Within a session, the NSURLSession class supports three types of tasks: data tasks, download tasks, and upload tasks. We will mostly use "data tasks".

The NSURLSession API provides a wide range of configuration options. Most settings are contained in a separate configuration object (which is an instance of NSURLSessionConfiguration).

For many of our getting-started examples, when you instantiate an NSURLSession object, you specify the following:

1. A configuration object that governs the behavior of that session and the tasks within it
2. Optionally, a delegate object; however, we will use *nil*. If you do not provide a delegate, the NSURLSession object uses a system-provided delegate.
3. The name of an *operation queue* that performs the task specified in the *completion handler block*. We will use *[NSOperationQueue mainQueue]*.

Your app can provide the request body content for an HTTP POST request in three ways. We will use an NSData object.

To upload body content with an NSData object, your app calls … the uploadTaskWithRequest:fromData:completionHandler: method to create an upload task, and provides request body data through the fromData parameter.

The session object computes the Content-Length header based on the size of the data object.

Your app must provide any additional header information that the server might require—content type, for example—as part of the URL request object.

.

Life Cycle of a URL Session with System-Provided Delegates

Here is the basic sequence of method calls that your app must make and completion handler calls that your app receives when using NSURLSession with the system-provided delegate:

1. Create a session configuration.

2. Create a session, specifying a configuration object, a nil delegate, and an operation queue.

3. Create task objects within a session that each represent a resource request. Write a completion handler block.

Each task starts out in a suspended state. After your app calls resume on the task, it begins downloading the specified resource.

When a task completes, the NSURLSession object calls the task's completion handler.

When your app no longer needs a session, invalidate it by calling *finishTasksAndInvalidate* (to allow outstanding tasks to finish before invalidating the object).

> Note: NSURLSession does not report server errors through the error parameter. The only errors your app receives through the error parameter are client-side errors, such as being unable to resolve the hostname or connect to the host.
>
> Server-side errors are reported through the HTTP status code in the NSHTTPURLResponse object.

.

.

.

.

.

.

.

.

.

.

.

.

Action:

Print

Like

Be the first to like this.

RSS feed

DPS907 Web services dev links

- DPS907 Course Notes
- Web service code examples

BTI420 Web apps dev links

- BTI420 Course Notes
- Web app code examples

DPS923 iOS dev links

- DPS923 Course Notes
- iOS code examples

My recent posts

- Responsive top-of-page logo image with Bootstrap
- Configure Fiddler export to save the message body
- Designing an ASP.NET MVC web app (version 1)
- Adding HTML Forms to ASP.NET MVC web apps
- Data persistence choices for ASP.NET MVC web apps

Past courses etc.

- 2015 Winter BTI420 (web apps)
- 2014 Fall DPS907 (web services)
- 2014 Winter BTI420 (web apps)
- 2014 Winter DPS923 (iOS dev)
- 2013 Fall DPS907 (web services)

- 2013 Summer BTI220 (web client)

Top
Blog at WordPress.com. The INove Theme.