NSURLConnection
in Objective-C

jusjos

Protocols that we have to conform to: **<NSURLConnectionDelegate, NSURLConnectionDataDelegate>**, sometimes also **NSURLConnectionDownloadDelegate**

To talk to a server, either for getting something (GET) or posting something (POST) we create a url request called **NSURLRequest**. With this request we create a **NSURLConnection** that actually does the work. The request can be synchronous (single go) or asynchronous (happening in bits and pieces. more efficient.). Note that we cannot do downloads in the background (i.e. when the app is not running) using NSURLConnection.

I) GET requests

2. Synchronously (not recommended)

We create a request in the same way. But don't create a connection with initWithRequest as in async. Instead, we use

```
NSData *responseData = [NSURLConnection
                    sendSynchronousRequest... ];
```

this returns an NSData object and an NSError pointer. if error is nil, we can parse the data here itself. We do not use delegate methods with this one. But note that sync requests will affect the UI. So we usually go for async requests.

II) POST requests

Can be sync or async requests. For example in async,

```
// Create the request. The URL is the place to
which we want to post our data.

   NSMutableURLRequest *request =
        [NSMutableURLRequest requestWithURL:URL];


// Specify that it will be a POST request. By
default it is GET.

   request.HTTPMethod = @"POST";


// Set header fields

   [request setValue:@"application/xml;
         charset=utf-8"
         forHTTPHeaderField:@"Content-Type"];


// Convert your data and set your request's
HTTPBody property. This data is what we want to
upload to the remote URL.

   NSString *stringData = @"some data";
   NSData *requestBodyData = [stringData
   dataUsingEncoding:NSUTF8StringEncoding];
   request.HTTPBody = requestBodyData;


// Create url connection and fire request.

   NSURLConnection *conn = [[NSURLConnection
   alloc] initWithRequest:request
              delegate:self];
```

I) GET requests

1. Asynchronously

We send a request to the server. We get a response. The data that we get back will be in increments. So we create a global mutable variable called responseData to which we keep appending the incoming data until we have all of it. That is..

```
NSMutableData *responseData
   //(globally. i.e. outside all methods.)
```

To perform the request, we do

```
NSURLRequest *request =
    [NSURLRequest requestWithURL:URLToLoad];
NSURLConnection *conn =
    [[NSURLConnection alloc]
    initWithRequest:request delegate:self];
          OR
[NSURLConnection
    sendAsynchronousRequest:request
    queue:someQueue completionHandler: ^{}];
       //this is useful for one-off requests.
```

The connection (conn) is created and in this case, fires off immediately. We can opt to not do this using another init method. Anyway, the request is by default - GET. So what happens after this?
The delegate methods get called. The main ones are:

```
NSURLConnectionDelegate
- connection:didFailWithError:
NSURLConnectionDataDelegate
- connection:didReceiveResponse:
- connection:didReceiveData:
- connectionDidFinishLoading:
- connection:willCacheResponse:
```

1. **connection:didReceiveResponse:**
   `responseData = [[NSMutableData alloc] init];`
   //this is called just once during the course of a server reply. When called, we initialize the data object so that it can be appended to in didReceiveData. If the request has failed, the new response will clear the old one and start again.

2. **connection:didReceiveData:**
   `[responseData appendData: data];`
   //this is called multiple times whenever we get some bit of data. We append this data to the object created earlier.

3. **connection:willCacheResponse:**
   //return nil from here if we are not storing a cached response.

4. **connectionDidFinishLoading:**
   //loading is complete. We have received all the data and can parse it.

5. **connection:didFailWithError:**
   //called if for some reason the request fails. Note that connectionDidFinishLoading and didFailWithError both signal that a request has finished either way.