Introduction to HTTP

Contents

Overview of HTTP

Hypertext Transfer Protocol

HTTP is a stateless protocol, meaning every request is treated as a completely independent transaction.

There is no relationship between one request and any previous requests; the server does not hold any information to connect several transactions with the same client.

This allows the underlying system to easily scale, since adding more capacity to a load balanced environment means the requests are evenly distributed across the entire system.

Overview of HTTP

Request/Response Structure

HTTP consists of a Request-Response client-server communication pattern; the client (usually a web browser) submits a request to a server (usually several load balanced web servers) which respond with the requested resource(s).

An HTTP request consists of a URI and method (or HTTP verb) followed by various HTTP Header Fields

```
GET /sport HTTP/1.1
Host: www.bbc.co.uk
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Connection: keep-alive
```

Overview of HTTP

Request/Response Structure

An HTTP response consists of an HTTP Status Code, various HTTP Header Fields, and the resource body - where applicable

```
HTTP/1.1 200 OK
Server: Apache
Expires: Fri, 18 Sep 2015 23:17:19 GMT
Content-Language: en-GB
Content-Type: text/html
Content-Length: 192376
Date: Fri, 18 Sep 2015 23:17:28 GMT
Cache-Control: private, max-age=30
X-Cache-Action: HIT
X-Cache-Hits: 35
X-Cache-Age: 9
X-LB-NoCache: true
Vary: X-CDN
Connection: keep-alive
```

TCP

Transmission Control Protocol

- TCP is the core protocol of the Internet Protocol. It is optimized for reliable, accurate, delivery of data packets.
- As such, it is not specifically optimized for speed.
- Making a large number of HTTP requests results in a large number of TCP connections and handshakes, which may result in an unoptimized user experience.

TCP

Three-way Handshake

The TCP three-way handshake is made up of the operations: SYN, SYN-ACK, and ACK

- **SYN** - Sequence Number
- **ACK** - Acknowledgement

TCP

Three-way Handshake

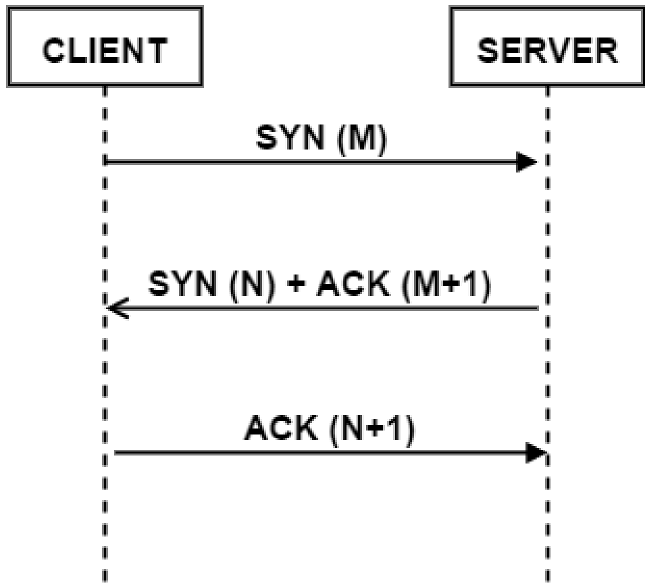
1. The client sends a `SYN` to the server. The client sets the segment's sequence number to a random value `M`.

2. The server replies with a `SYN-ACK`. The acknowledgment number is set to one more than the received sequence number i.e. `M+1`, and the sequence number that the server chooses for the packet is another random number, `N`.

3. The client sends an `ACK` to the server. The acknowledgement number is set to one more than the received sequence number i.e. `N+1`.

TCP

Three-way Handshake



Verbs

HTTP "verbs" are Methods by which a client can request a resource from a server.

The most common ones are:

- `GET`
- `POST`
- `PUT`
- `DELETE`

GET

- The GET verb represents a request for a specific resource; no extra data is submitted.
- It is considered "Safe"; multiple requests for the same resource should not change that resource.
- GET is considered "Idempotent" as it achieves the same result no matter how many times it is executed.

```
$.ajax({
    url: 'products/123',
    type: 'GET',
    success: function(response) { /* use the returned data */ }
});
```

or the shorthand form

```
$.get('products/123', function(data){ /* use the returned data */ };
```

POST

- The POST verb is used to send data to the server, such as a form submission or file upload.
- POST is *not* considered "Safe"; multiple requests to the same resource are actually requesting the server create multiple new instances of the submitted data.
- POST is *not* considered "Idempotent" as it achieves a different result each time it is executed. What is a good way to handle this?

```
$.ajax({
    url: 'products/',
    type: 'POST',
    data: { name: "truck", price: "10GBP" }
    success: function(response) { /* use the returned data */ }
});
```

or the shorthand form

```
 $.post( "products/", { name: "truck", price: "10GBP" })

  .done(function( data ) {  /* use the returned data */  });
```

PUT

- The PUT verb is usually used to update an existing resource with the data being submitted, or request creation of a resource with that data if it doesn't already exist
- PUT is *not* "Safe" since it alters the resource at the URL
- PUT is considered "Idempotent" - why?

```
$.ajax({
    url: 'products/123',
    type: 'PUT',
    data: { name: "truck", price: "10GBP" }
    success: function(response) { /* use the returned data */ }
});
```

DELETE

- The DELETE verb requests the server to delete the resource identified by the URL.
- DELETE is *not* "Safe" since it alters the resource at the URL (obviously!)
- DELETE is considered "Idempotent" - why?

```
$.ajax({
    url: 'products/123',
    type: 'DELETE',
    success: function(result) { /* just status code returned */ }
});
```

Verbs

- How do `GET`, `POST`, `PUT`, and `DELETE` map to CRUD (Create Read Update Delete), if at all?
- There are others such as `HEAD` and `OPTIONS`

Exercise

---

1. Create a single html page with a reference to jQuery
2. Configure it to make a `GET` request to a website using jQuery
3. Output the response - or part of the response - into the html page

---

Hints

- Try using http://www.bbc.co.uk/news/technology
- Remember the syntax:

```
$.ajax({
    url: 'products/123',
    type: 'GET',
    success: function(response) { /* use the returned data */ }
});
```

or

```
$.get('products/123', function(data){ /* use the returned data */ };
```

Essential HTTP

Contents

Headers

Header fields are components of the header section of request and response messages.

HTTP Request Message

```
<request-line>
<general-headers>
<request-headers>
<entity-headers>
<empty-line>
[<message-body>]
```

- **Request Line**
    - [METHOD] [URI] [HTTP VERSION] e.g. `GET /sport HTTP/1.1`
- **General Headers**
    - Used in both request and response messages, usually just the date
- **Request Headers**
    - E.g. User-Agent, If-Modified-Since, and Accept headers
- **Entity Headers**
    - If sending data (e.g. via POST or PUT), these would have the content type and content length
- **Message Body**
    - If sending data, this is the encoded data

HTTP Request Message

```
<request-line>
<general-headers>
<request-headers>
<entity-headers>
<empty-line>
[<message-body>]
```

```
GET /sport HTTP/1.1
Host: www.bbc.co.uk
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-GB,en-US;q=0.8,en;q=0.6
Connection: keep-alive
```

HTTP Response Message

```
<status-line>
<general-headers>
<response-headers>
<entity-headers>
<empty-line>
[<message-body>]
```

- **Status Line**
  - [HTTP VERSION] [STATUS CODE] [STATUS] e.g. `HTTP/1.1 200 OK`
- **General Headers**
  - Same as for a Request
- **Response Headers**
  - e.g. "Server"
- **Entity Headers**
  - Assuming a resource is returned (e.g. an html page), these would include content type andcontent length
- **Message Body**
  - If returning data, this is the appropriately encoded data (e.g., an html page)

HTTP Response Message

```
<status-line>
<general-headers>
<response-headers>
<entity-headers>
<empty-line>
[<message-body>]
```

```
HTTP/1.1 200 OK
Server: Apache
Expires: Fri, 18 Sep 2015 23:17:19 GMT
Content-Language: en-GB
Content-Type: text/html
Content-Length: 192376
Date: Fri, 18 Sep 2015 23:17:28 GMT

<!DOCTYPE html>
<html  lang="en-GB"  id="front-page-no-brand" class="front-page">
    <!--Rendered by NOLAPPS709-6006@2015-09-19T16:36:10+00:00 -->

    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta http-equiv="X-UA-Compatible" content="IE=8" />
        <title>BBC Sport - Sport</title>
        <meta name="description" content="BBC Sport - live sports coverage, breaki

        <meta name="keywords" content="BBC, Sport, BBC Sport, bbc.co.uk, world, uk
```

Accept Headers & Content Negotiation

Accept

By sending different request headers for the exact same resource, it is possible to retrieve that resource in a different format.

For example, changing a request header from:

```
Accept: text/xml
```

to

```
Accept: text/javascript
```

could result in that same piece of data being returned in JSON instead of XML.

Accept Headers & Content Negotiation

Accept

If you're implementing your own content negotiation, then you could support something less obvious, such as returning a visual representation of the same data if the request header is changed to:

```
Accept: image/jpg
```

Accept Headers & Content Negotiation

Accept

The client is usually sending a series of preferences and their rank:

```
Accept: text/html; q=1.0, text/*; q=0.8
```

In this example we're giving `text/html` the highest preference (`1.0`) and then any other `text` a slightly lower rank (`0.8`).

This allows the server to respond with the most appropriate type for this resource.

Accept Headers & Content Negotiation

Accept-Language

A useful form of content negotiation is to make the same information available at the same URI, but in different languages.

For example, changing a request header from:

```
Accept-Language: en-GB
```

to

```
Accept-Language: fr-FR
```

could deliver the same resource but in French instead of English.

This particular header is set automatically by your browser based on the chosen system language.

Accept Headers & Content Negotiation

Accept-Language

Similar to the `Accept` header setting a preference for various options, so too can we do this for `Accept-Language`:

```
Accept-Language: de; q=1.0, en; q=0.5
```

"I'd really prefer German, otherwise any English will do"

Status codes

The server always responds with an HTTP status code to identify what happened when processing the request.

The page cannot be found

100s - FYI

Informational. You'll never really see these in practise as they're just informational statuses between client and server

- 100 - Continue
  - Used to inform the client that the initial part of the request has been received and has not yet been rejected by the server
- 101 - Switching Protocols
  - For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time, synchronous protocol might be advantageous when delivering resources that use such features

200s - All good!

Great success! The request went well, or is going well so far. There are a few of these, and the most common ones are below.

- 200 - OK
  - The most common HTTP status code you'll see; the request succeeded.
- 201 - Created
  - The request has been fulfilled and resulted in a new resource being created.
- 204 - No Content
  - The server has fulfilled the request but does not need to return an entity-body.
  - A common response from a `DELETE`.

300s - Over there

Redirection; the client needs to go somewhere else in order to complete the request. Again, there are a few of these status codes so the more common ones are below.

- 301 - Permanent Redirect
    - Returns an alternative location for the requested resource, informing the client to remember not to go to the original location any more.
- 302 - Found (aka Temporary Redirect)
    - Returns a different location to the one originally requested; client is still allowed to go to the original location in subsequent requests.
- 304 - Not Modified
    - Indicates the resource has not been modified since last requested. Typically, the client provides a header with a date and time against which to compare.
    - The response has no body, and merely informs the client that there is no newer version available so it should use the version it already has.

400s - You messed up

Client error; something about the request was invalid or missing. There are many of these statuses, the most common ones being:

- 400 - Bad Request
    - Really unhelpful. Generally related to bad syntax in the request.
- 401 - Unauthorized
    - The request needs to have some form of authentication before the server will accept it.
- 403 - Forbidden
    - The server understood the request, but is refusing to accept it. You may be authenticated (avoiding the 401) but you still may not have permission to view that resource.
    - Essentially, the computer said "no".

400s - You messed up

- 404 - Not Found
    - The resource being requested does not match anything the server understands.
    - Unfortunately this error code has become a term understood by the general public due to early web developers (like me) being lazy, and letting the server deliver a default "404" page which early users became used to.
- 405 - Method Not Allowed
    - For example, the client made a `PUT` request, but that particular resource is only available via `GET`
- 418 - I'm A Teapot
    - The result of an April Fools' joke in 1998. However, there have been some implementations of it!

500s - I messed up

Server error; the request was accepted, and something went horribly wrong when processing it. Here's a summary:

- 500 - Internal Server Error
  - Another helpful one. Right up there with "Computer says 'no'". An unexpected condition occurred whilst processing the request.
- 501 - Not Implemented
  - Similar to `405` (Method Not Allowed), except that the server itself has no idea how to handle that method at all, as opposed to a single resource doesn't support it.
- 503 - Service Unavailable
  - The server is there, but it's unable to respond. It may be overloaded or undergoing maintenance. Usually a temporary condition.

Exercise

---

1. Open up dev tools on your browser
2. Make a request for a URL
3. Investigate the Request and Response headers and the status codes

---

Hints

- Try using http://bbc.co.uk/sport
- What are the first few responses doing?

Going Further with HTTP

1. Caching

Caching

Web application caching is the process of storing dynamically generated data for reuse.

- It can reduce the load on the server, give a faster response, and reduce network load

Caching

There are many different possible locations between the server code and the browser window where the resources can be cached; where can you think of?

Caching

There are many different possible locations between the server code and the browser window where the resources can be cached; where can you think of?

These are the main categories of cache:

1. Browser Cache
2. Proxy Cache
3. Reverse Proxy (aka Surrogate)
4. Server Cache

Caching

1. Browser Cache

A browser cache stores the recently viewed web content (html/text pages, images, javascript files, stylesheets, etc), in order to speed up the browsing experience for the end user.

Returning to a page already in the browser cache will result in an HTTP `304` status for any cached resource that had an appropriate caching header defined, allowing the browser to cache it.

Such files are commonly the static content, such as images, javascript, and css stylesheets.

Executing a "Hard Refresh" or clearing the browser's cache forces it to request the resources from the server again.

Caching

2. Proxy Caching

Where several users may be sharing the same internet connection, the device that provides external access may also act as a cache.

This can significantly reduce bandwidth when many users are accessing the same or similar resources.

The idea being that only one user needs to request a resource from the orgin server and any other users requesting the same resource can be served it from the proxy cache instead.

Caching

3. Reverse Proxy

A reverse proxy protects the origin server by filtering all traffic thorugh itself as if it were the web server - hence the other term of "Surrogate".

The client has no knowledge that it's interacting with a proxy instea dof a server unless the proxy is configured to add extra HTTP Headers to the Response message (e.g. a `Via` header).

Reverse proxies can be the difference between a server meltdown in unpredictably busy periods and your site staying up.

Caching

4. Server Cache

The server itself can cache dynamically created content in memory, such that subsequent requests for the same content are not regenerated.

ASP.Net MVC has some very convenient attributes avilable to allow the server to cache individual actions or entire controllers.

It can be configured to serve the cached version for a set period of time, based on a combination of headers and parameters:

```
[OutputCache(Duration = 30, VaryByParam = "id", VaryByHeader = "Accept-Language")]
```

Caching Rules

The rules around whether a resource can be cached or not, and if so for how long and where, are defined in the Response HTTP Headers.

- Dynamic Content
- Cachable Private Content
- Cachable Public Content

Caching Rules

Dynamic Content

The headers that say "Nothing is allowed to cache this resource":

```
Pragma: no-cache
Cache-control: no-cache, must-revalidate
```

For when the content of the resource is expected to change for each request, or to change frequently enough so as to make caching detrimental to the expeerience for the end user.

Caching Rules

Cachable Private Content

This resource can be cached, but *only* in the end user's browser cache:

```
Cache-control: private, max-age=36000
```

The content is specific to the end user, and as such should not be cached in a shared access cache, such as proxy or reverse proxy.

Caching Rules

Cachable Public Content

This resource can be cached anywhere:

```
Cache-control: public, max-age=1209600
```

Cache it anywhere, for anyone.

Exercise 1

1. Create an **Empty MVC** Web Project within Visual Studio
2. Add a controller `TimeController.cs` and create a class `TimeViewModel.cs`

```
public class TimeViewModel
{
    public string CurrentTime { get; set; }
}
```

1. Create a view `~/Views/Time/Index.cshtml` to display the view model
2. Create an action `Index` on `TimeController` to return this `Index.cshtml` view, passing in a new instance of `TimeViewModel.cs` with the current time.
3. Implement caching on the action to cache for 10 seconds:

```
[OutputCache(Duration = 10)]
```

1. Run the project and refresh your page. What happens?

Exercise 2

1. Alter your attribute to allow the caching to vary by a querystring parameter:

```
[OutputCache(Duration = 10, VaryByParam = "x")]
```

1. Run the project, refresh your page and note the results.

2. Add the querystring param "x" with different values and refresh. What happens?

Robin Osborne

Contact

- @rposbo
- robin@frameworktraining.co.uk
- robinosborne.co.uk

Slides

- https://github.com/otomotech/http-course