Comp215: Introduction to the World Wide Web

Dan S. Wallach (Rice University)

Copyright © 2015, Dan S. Wallach. All rights reserved.
What, exactly, is the Web?
The Web is a whole stack of things

<table>
<thead>
<tr>
<th>Layer</th>
<th>Data Unit</th>
<th>Function</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host layers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7. Application</td>
<td>Data</td>
<td>High-level APIs</td>
<td>HTTP, FTP, SMTP, SSH</td>
</tr>
<tr>
<td>6. Presentation</td>
<td>Data</td>
<td>Specific file formats, data formats</td>
<td>HTML, JPEG, JSON</td>
</tr>
<tr>
<td>5. Session</td>
<td></td>
<td>Managing communications</td>
<td>SSL/TLS</td>
</tr>
<tr>
<td>4. Transport</td>
<td>Datagram</td>
<td>Reliable transmission between any two nodes on a network</td>
<td>TCP, RSTP (real-time streaming)</td>
</tr>
<tr>
<td>Media Layers</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Network</td>
<td>Packet</td>
<td>Addressing, routing, traffic control</td>
<td>IPv4, IPv6, old stuff like AppleTalk</td>
</tr>
<tr>
<td>2. Data link</td>
<td>Frame</td>
<td>Reliable transmission of data frames between two nodes</td>
<td>PPP, Ethernet MAC</td>
</tr>
<tr>
<td>1. Physical</td>
<td>Bit</td>
<td>Transmit/receive raw bits over a physical medium</td>
<td>Ethernet, DSL, WiFi</td>
</tr>
</tbody>
</table>
The bare minimum you need to know

The Internet *routes* **IP packets**.

Every packet has a source and destination address.
Routers don’t need to do anything else. Just pass on the packet. That’s it.
(Yes, it’s more complicated than that. For now we don’t care.)

The Internet makes no promises. Packets might not arrive.
It’s up to the transport layer (TCP) to notice failures and retransmit.
Subtle details matter, especially dealing with congestion control / fairness.
You’ll want to learn a lot more. For example, whose fault is it if your connection to Netflix doesn’t work?
Netflix blames Verizon / AT&T; they disagree. The details matter.

If you want to understand “net neutrality”...

You’ll want to learn a lot more. For example, whose fault is it if your connection to Netflix doesn’t work?

Netflix blames Verizon / AT&T; they disagree. The details matter.


Take Comp429, learn lots more about this and other related topics!
The web happens on top

Several technologies combined together to make it possible

URLs (uniform resource locators)
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]

HTML (hypertext markup language)

HTTP (hypertext transfer protocol)
HTTPS (HTTP running over TLS - Transport Layer Security)

JavaScript, Cascading Style Sheets (CSS), image/media types
Example: Web advertisements

Some web sites host their own ads


<img src="http://graphics8.nytimes.com/adx/images/ADS/32/05/ad.320528/NYT_DOT_LEFT.jpg" width="184" height="90" border="0"></a>
Example: Web advertisements

Some web sites host their own ads

<img src="http://graphics8.nytimes.com/adx/images/ADS/32/05/ad.320528/NYT_DOT_LEFT.jpg" width="184" height="90" border="0"></a>
If the user clicks the ad
If the user clicks the ad

GET /adx/bin/adx_click.html? ...

User

nytimes.com
If the user clicks the ad

User ➔ GET /adx/bin/adx_click.html? ...

nytimes.com ➔ HTTP/1.1 302 Found
Location: http://www.marcjacobs.com/?utm_source=nyt ...

User
If the user clicks the ad

If a user clicks an ad, the ad server will respond with a 302 HTTP redirect:

```
GET /adx/bin/adx_click.html? ... HTTP/1.1 302 Found
Location: http://www.marcjacobs.com/?utm_source=nyt ...
```

The main website tells the user's browser to redirect to the advertiser, giving credit for the click.

```
GET /?utm_source=nyt ... http://www.marcjacobs.com
```
Typical 3rd party display ads

Similar operation to 1st party ads

<a target="_blank" href="http://ad.doubleclick.net/click;h=v8/3d41/4/a2/%2a/g;255075316;0-0;78667076;3454-728/90;47210945/47227088/4;/okv=;pc=aol02555489;/;sscs=%3fhttp%3A//at.atwola.com/adlink%2F5113%2F786827%2F0%2F225%2FAdId%3D2555489%3BBnId%3D1%3Bitime%3D595358030%3Bnodecode%3Dyes%3Bimpref%3D13545953581141452207%3Blink%3Dhttp://www.choosenissan.com/zip.aspx?dcp=zmm.78667076.&amp;dcc=47210945.255075316"> <img src="http://s0.2mdn.net/viewad/1361550/All-New+Altima_MSRP_728x90.jpg" alt="Advertisement" border="0"> </a>
If the user clicks the ad

User

GET /click;h=v8/3d41/4/a2/ ...

ad.doubleclick.net
If the user clicks the ad

**User**

GET /click;h=v8/3d41/4/a2/ ...

HTTP/1.1 302 Moved Temporarily
Location: http://at.atwola.com/adlink...

**User**

GET /adlink...

**ad.doubleclick.net**

**at.atwola.com**
If the user clicks the ad

HTTP/1.1 302 Moved Temporarily
Location: http://at.atwola.com/adlink...

HTTP/1.1 302 Redirect
Location: http://www.choosenissan.com/...

HTTP/1.1 302 Moved Temporarily
Location: http://at.atwola.com/adlink...

GET /click;h=v8/3d41/4/a2/ ...
Tracking Cookies & Mobile Geolocation
Here's a paper example
Here's a paper example
What happens if you scan this QR code with your phone?
HTTP Headers

Body (HTML + JavaScript)
Cookies

Key/value pairs, stored in the browser
Transmitted back to the site that set them

Set-Cookie: _unbg=50bbbe60cb7419.09871084.50bbbe60cb7512.99318963;
expires=Fri, 02-Dec-2022 20:47:28 GMT
JavaScript behaviors

Arbitrary code, running inside your browser

tracking('2957203_db7b71b43a3ad916d08ac54b0b2c86a8');
function tracking(vid) {
    if (vid == -1)
    {
        hide('user_action');
        tagAction();
        return;
    }
    if (navigator.geolocation)
    {
        g_vid = vid;
        navigator.geolocation.getCurrentPosition(function(pos){
            if (send) return;
            send = true;
            var a = (pos.address) ? pos.address : {};
            var c = (pos.coords) ? pos.coords : {};
            var params = {
                vid: vid,
                country: ((a.country) ? a.country : ''),
                region: ((a.region) ? a.region : ''),
                county: ((a.county) ? a.county : ''),
                city: ((a.city) ? a.city : ''),
                street: ((a.street) ? a.street : ''),
                streetNumber: ((a.streetNumber) ? a.streetNumber : ''),
                latitude: ((c.latitude) ? c.latitude : ''),
                longitude: ((c.longitude) ? c.longitude : ''),
                altitude: ((c.altitude) ? c.altitude : ''),
                accuracy: ((c.accuracy) ? c.accuracy : ''),
                altitudeAccuracy: ((c.altitudeAccuracy) ? c.altitudeAccuracy : ''),
                heading: ((c.heading) ? c.heading : ''),
                speed: ((c.speed) ? c.speed : '')
            };
            if (send)
            {
                send = true;
                // Code to send the parameters
            }
        });
    } else {
        errorCallback({code: 4}); // 15sec for user decision
    }
}
Web server programming
Two main kinds of requests

HTTP “GET” requests
Used to request static resources
Ideally, should be “stateless”

HTTP “PUT” requests
Used to respond to form requests
Ideally, should be the only thing that changes state

So far, you’ve only written code for “GET” requests
And they return a different value every time!
“Please don’t hit the back button”

“Because we’re not doing functional programming!”
More on this next week.
If the user types “2 5 +”, the URL looks like this.
http://localhost:4567/rpnserver/?input=2%205%20%2B

get("/rpnserver/", (request, response) -> {
    // dump the full request into the log
    Log.i(TAG, "input url: " + request.url());
    Log.i(TAG, "input body: " + request.body());
    Log.i(TAG, "input query string: " + request.queryString());

    // the library gives us "parameters" and "query parameters". The former seems to be elements from the URL
    // before the ? as specified with colons in the get() command above. The latter seems to be the set of
    // ?foo=x,bar=y things that go at the end of the URL, which is what we're looking for here.

    TreeMap fromMap(request.params())
        .toList()
        .foreach((entry) -> Log.i(TAG, () -> String.format("Parameter: %s->%s", entry.getKey(), entry.getValue())));

    TreapSet fromSet(request.queryParams())
        .toList()
        .foreach((entry) -> Log.i(TAG, () -> String.format("QueryParams: %s -> %s", entry, request.queryParams(entry))));

    String commandLine = request.queryParams("input");
    Log.i(TAG, "commandLine: " + commandLine);
Web requests & SparkJava

If the user types “2 5 +”, the URL looks like this.

http://localhost:4567/rpnserver/?input=2%205%20%2B

get("/rpnserver/", (request, response) -> {
    // dump the full request into the log
    Log.i(TAG, "input url: " + request.url());
    Log.i(TAG, "input body: " + request.body());
    Log.i(TAG, "input query string: " + request.queryString());

    // the library gives us "parameters" and "query parameters". The former seems to be elements from the URL
    // before the ? as specified with colons in the get() command above. The latter seems to be the set of
    // ?foo=x,bar=y things that go at the end of the URL, which is what we're looking for here.

    TreeMap.fromMap(request.params()).
        .toList().
        .forEach((entry) -> Log.i(TAG, () -> String.format("Parameter: %s->%s", entry.getKey(), entry.getValue())));

    TreapSet.fromSet(request.queryParams()).
        .toList().
        .forEach((entry) -> Log.i(TAG, () -> String.format("QueryParams: %s -> %s", entry, request.queryParams(entry))));

    String commandLine = request.queryParams("input");
    Log.i(TAG, "commandLine: " + commandLine);

Key/value pairs, escaped (%20 = space, %2B = +)
get("/rpnserver/", (request, response) -> {
    // dump the full request into the log
    Log.i(TAG, "input url: " + request.url());
    Log.i(TAG, "input body: " + request.body());
    Log.i(TAG, "input query string: " + request.queryString());
    // the library gives us "parameters" and "query parameters". The former seems to be elements from the URL
    // before the ? as specified with colons in the get() command above. The latter seems to be the set of
    // ?foo=x,bar=y things that go at the end of the URL, which is what we're looking for here.
    TreeMap.fromMap(request.params()).
        .toList().
        .forEach((entry) -> Log.i(TAG, () -> String.format("Parameter: %s->%s", entry.getKey(), entry.getValue())));
    TreapSet.fromSet(request queryParams()).
        .toList().
        .forEach((entry) -> Log.i(TAG, () -> String.format("QueryParams: %s -> %s", entry, request.queryParams(entry))));
    String commandLine = request.queryParams("input");
    Log.i(TAG, "commandLine: " + commandLine);
“Parameters” vs. “query parameters”

Spark lets you encode requests in the URL path, not just the “query”

Let’s say you’re returning an image and you know how to (optionally) resize it.

```java
get("/image/:name/", (request, response) -> {
    // no request for a particular image size
    String imgName = request.params("name");

    ...
});

get("/image/:name/size/:width/:height", (request, response) -> {
    String imgName = request.params("name");
    int width = Integer.parseInt(request.params("width"));
    int height = Integer.parseInt(request.params("height"));

    ...
});
```
“Parameters” vs. “query parameters”

Spark lets you encode requests in the URL path, not just the “query”
Let’s say you’re returning an image and you know how to (optionally) resize it.

This is structural pattern matching (see week 9). You’re guaranteed that there will be a “width” and “height” or else Spark wouldn’t have called your lambda.

get("/image/:name/size/:width/:height", (request, response) -> {
    String imgName = request.params("name");
    int width = Integer.parseInt(request.params("width"));
    int height = Integer.parseInt(request.params("height"));

    ...

});
Web responses

The lambda takes two inputs `(request, response)` and returns a String

So what’s the purpose of the response object?

- Set cookies
- Issue “redirect” commands
- Label the response “type” (image/jpeg, text/html, etc. – “MIME types”)
  
  [http://www.iana.org/assignments/media-types/media-types.xhtml](http://www.iana.org/assignments/media-types/media-types.xhtml)

Ultimately the lambda returns the “body” of the request

- We’re returning Strings (via our JSON code), but you could also return byte arrays, etc.
- There’s actually a huge “plugin” architecture in Spark to “transform” things.
- You mostly don’t need to care about this.
Returning vanilla HTML

Option 1: just stick a .html file in the “WebPublic” folder

Option 2: compute it yourself

```javascript
get("/rpncalc/", (request, response) =>
  html().with(
    Utils.mui/cssHeader("Comp215 RPN Calculator!", "/rpncalc.js"),
    body().with(
      header().withId("header").with(nav().withId("appbar")
        .withClass("mui-container-fluid")
        .with(h1("Comp215 RPN Calculator!"))),
      div().withId("textOutput"),
      div().withId("goButton").with(
        button(">").withClass("mui-btn mui-btn-floating mui-z3")),
      div().withId("footer")
        .with(Utils.textEntryForm("commandLine", "type your commands here")))));
```
Returning vanilla HTML

Option 1: just stick a .html file in the “WebPublic” folder

Option 2: compute it yourself

```javascript
get("/rpncalc/", (request, response) ->
  html().with(
    Utils.muicssHeader("Comp215 RPN Calculator!", "/rpncalc.js"),
    body().with(
      header().withId("header").with(nav().withId("appbar")
        .withClass("mui-container-fluid")
        .with(h1("Comp215 RPN Calculator!"))),
      div().withId("textOutput"),
      div().withId("goButton").with(
        button(">").withClass("mui-btn mui-btn-floating mui-z3"))),
      div().withId("footer")
        .with(Utilities.textEntryForm("commandLine", "type your commands here")))));
```
Returning vanilla HTML

Option 1: just stick a .html file in the “WebPublic” folder
Option 2: compute it yourself

```javascript
get("/rpncalc/", (request, response) =>

html().with(

Utils.muicssHeader("Comp215 RPN Calculator!", "/rpncalc.js"), body().with(

header().withId("header").with(nav().withId("appbar")
 .withClass("mui-container-fluid")
 .with(h1("Comp215 RPN Calculator!")) ),

div().withId("textOutput"),

div().withId("goButton").with(

button(">").withClass("mui-btn mui-btn-floating mui-z3"),

div().withId("footer")

 .with(Utils.textEntryForm("commandLine", "type your commands here")))))
```

“helper” functions that we wrote

j2html “fluent” HTML builder
Returning vanilla HTML

Option 1: just stick a .html file in the “WebPublic” folder

Option 2: compute it yourself

get("/rpncalc/", (request, response) =>
  html().with(
    Utils.muicssHeader("Comp215 RPN Calculator!", "/rpncalc.js"),
    body().with(
      header().withId("header").with(nav().withId("appbar")
        .withClass("mui-container-fluid")
        .with(h1("Comp215 RPN Calculator!"))),
      div().withId("textOutput"),
      div().withId("goButton").with(
        button(">").withClass("mui-btn mui-btn-floating mui-z3"),
      div().withId("footer")
        .with(Utils.textEntryForm("commandLine", "type your commands here"))))));

“styles” for our web client (more on this Friday)

“helper” functions that we wrote

j2html “fluent” HTML builder
Returning JSON data

get("/rpnserver/", (request, response) -> {
    String commandLine = request.queryParams("input");

    if (commandLine != null) {
        response.status(200); // okay!
        response.header("cache-control", "no-cache");
        Parser.JObject jresponse =
            jobject(jpair("response", rpnCalculator.calc(commandLine)));
        return jresponse.toString();
    }

    response.status(400); // bad request
    return "{"; // empty JSON
});
get("/rpnservice/", (request, response) -> {
    String commandLine = request.queryParams("input");

    if (commandLine != null) {
        response.status(200); // okay!
        response.header("cache-control", "no-cache");
        Parser JObject jresponse = jobject(jpair("response", rpnCalculator.calc(commandLine)));
        return jresponse.toString();
    }

    response.status(400); // bad request
    return "{ }"; // empty JSON
});
Returning JSON data

get("/rpnserver/", (request, response) -> {
    String commandLine = request.queryParams("input");

    if (commandLine != null) {
        response.status(200); // okay!
        response.header("cache-control", "no-cache");
        Parser.JObject jresponse =
            jobject(jpair("response", rpnCalculator.calc(commandLine)));
        return jresponse.toString();
    }

    response.status(400); // bad request
    return "\{"\}"; // empty JSON
});

hey browser, don’t cache this ("because mutation!")

HTTP response status codes
404: Not found
404: Not found

You’ve probably seen this one beforehand.
404: Not found

You’ve probably seen this one beforehand.

Yahoo! once bought this advertisement in AT&T Park (San Francisco Giants)
Live demo: Chrome’s dev console