Databases and the Cloud: The Notes

Josh Felmeden

2018
December

Contents

1 The Internet **1**
1.1 Protocols . . . 1
1.2 Internet Layers . . . 1
1.3 HTTP: Hyper text transfer protocol . . . 2
1.4 Crud . . . 2
1.5 Structure . . . 2
1.6 Status codes . . . 2
1.7 URLs . . . 3

2 Developing web pages **3**
2.1 Markup . . . 3
2.2 HTML . . . 3
2.2.1 Attributes . . . 4
2.2.2 Links . . . 4
2.3 Forms . . . 5
2.3.1 Validation in forms . . . 5
2.3.2 Buttons . . . 5
2.4 CSS . . . 6
2.5 Data Formats and Operations . . . 7
2.5.1 UNICODE and Encoding . . . 8
2.5.2 Tables and CSV . . . 8
2.5.3 Stream Processing . . . 9
2.6 Representing Data as Trees . . . 9
2.6.1 XML . . . 10

1 The Internet

End systems are connected via the **communication links** that consist of the different types of physical media. Usually, the end systems are not directly attached by a single link, but rather they are attached through a router.

There are two kinds of host: *clients* and *servers*. A program or machine that responds to request and others is called a **server** while a program or machine that sends the requests to the server is called a **client**.

The internet is made possible by the development, testing, and implementation of the *internet standards*. They are developed by the Internet Engineering Task Force (or the IETF). Their documents are known as RFCs (request for comments). There are a number of protocols, such as TCP, IP, HTTP, and SMTP (this one is used for emails). There are more than 2000 RFCs.

1.1 Protocols

A **protocol** is a set of rules that govern the communication to ensure a standard of communication. It also consists of messages sent and actions taken in response to replies or other such events.

A simple protocol could be where one machine sends a message (called a *request*) and another machine replies with a response. This can then be repeated.

1.2 Internet Layers

- HTTP
  - Makes request
  - Reads and handles the response
- TCP
  - Breaks data up into packets
  - Puts the packets back in order and reassembles messages
- IP
  - Attaches to and from addresses to each packet
  - Reads and groups packets based on the address
- Physical internet
  - Send bits to local routers
  - Receives bits and assembles into packets

1.3 HTTP: Hyper text transfer protocol

What's the difference between the web and the internet? Well, the internet is the computer network itself (or the whole infrastructure): while the web (or the world wide web) is an application that runs on that infrastructure.

It's probably the most common application protocol that there is on the web (but there are others like video streaming and FTP and the like). Right now, there's a version 2.0, but we'll be focussing on version 1.1 here.

1.4 Crud

CRUD is an acronym for the basic operations that can be carried out on data.

- Create
  - The create interaction creates a new resource in a server assigned location. The create interaction is performed by a HTTP POST method.
- Read
  - The read interaction accesses the current contents of a resource. The interaction is performed by a HTTP GET method
- Update
  - The update interaction makes a whole new version for an existing resource (or makes a new one if there isn't one)
- Delete
  - The delete interaction deletes an existing resource

1.5 Structure

HTTP is *line-based* and each line ends with a **carriage return line feed** (CR LF). In it, there is a header and a method.

1.6 Status codes

There are some cases where an interaction does not go well. The response from a server can be a number, and the first digit informs you of the nature of the error.

1.7 URLs

The internet needs to have addresses. It needs to know the addresses of both the client and the server. The URL (**uniform resource locator**) tells you where some resource is. A resource is an *address*.

2 Developing web pages

2.1 Markup

Historically, marking up a paper manuscript was done by editors to show authors how to revise their manuscripts. The markup was done in *blue pen* to make it distinguishable from the manuscript text.

In electronic documents, **tags** are used to make the markup distinguishable from the content. A markup language is used to annotate a document.

2.2 HTML

HTML (or hypertext markup language) consists of a fixed set of *tags* that describe how information should be displayed. For example:

```
<p> This is some text </p>
<h1> This is some header </h1>
```

The Browsers do not display the HTML tags, but they use them to render the content of the page.

HTML5 is different from HTML because it's simpler, but also **semantic** (which means that some of the tags describe what the data means as well has how it should be displayed). It also has some more features.

Example HTML5:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
        <title>My title</title>
    </head>
    <body>
        content
    </body>
</html>
```

Tags have to be nested too.

A block-level element always starts on a new line and takes up the full width available. Conversely, an inline element doesn't start on a new line and it only takes up as much width as is necessary.

The <div> tag is a block level tag that has no specific meaning. This is OK to use for layout purposes, but you should not use it as a replacement for something that should be a semantic tag. Because the semantic tags are mostly a new addition to HTML5, older frameworks used <div> all over the place to structure the pages.

2.2.1 Attributes

In this example:

```
<p id="today">
    28 September
</p>
<p class="info">
    lecture 2
</p>
<p class="info">
    QB 0.18
</p>
```

2.2.2 Links

Almost anything can go inside a <a> tag: text, images, other HTML elements. The href could be a full URL, or it can be relative to the current page.

The main issue with HTML is that you need to structure your web pages really carefully because it's going to be viewed on all kinds of devices and browsers.

Here are some basic rules:

1. Use lower case element names
2. Close all your elements (you don't need to close them in HTML5 but do it anyway)
3. In HTML5, it's optional to close the empty statements, but do it anyway.
4. HTML5 allows the mixing of uppercase and lowercase names, but just use lowercase because it looks nicer and it's easier to write.
5. HTML5 allows attribute values without quotes but again, it's bad because it looks ugly
6. ALWAYS add the alt attribute to images, because if, for some reason, the image can't be displayed, you need some alternate text to display. It's also used for people using screen readers.
7. In HTML5, the html and body tag can be omitted, but, again, it's **bad**. It can crash some XML software.
8. To ensure that everything is interpreted and has correct search engine indexing, the language AND the character encoding should be defined as early as possible.
9. Don't use absolute pixel width measurements

2.3 Forms

The form tag is used for things like buttons, text boxes, etc. It has input types of things like:

- Button
- Month
- Number
- Text
- Password
- Color
- Date
- ...

The **action** attribute defines the action to be performed when the form is submitted. Normally, the data from the form is sent to a web page on the server when the user clicks on the submit button. For example:

```
<form method="post"
action="/action_page.php">
</form>
```

In this example, the data is sent to a page on the server called "/action_page.php". This page contains a script that will handle the form data such as storing it in a database.

There are two (2) methods to send form data, **GET** and **POST**. In HTML5, browser forms support them both. GET places form data in the URL parameters by default (GET/search?query=pancakes), while POST sends the data in the HTTP request body. There are fewer limitations and it's more secure because the data is not visible in the URL.

2.3.1 Validation in forms

If you use type="number", then it won't let you type in letters. It you use required, then the browser won't let you submit if the field is empty.

Place holder is text that can be displayed while the field is empty. It's NOT a label.

2.3.2 Buttons

Buttons can have these types:

- Submit (default)

- Reset: reset all form fields
- Button: do nothing by default (use this if you're using JS).

2.4 CSS

Use HTML for the structure, and then CSS for the styling. This includes layout, appearance, and some behaviours. You can customise ANYTHING. If you want emphasised words to be underlined, then by golly you can do that.

CSS stands for Cascading Style Sheets. CSS describes how HTML elements are to be displayed on screen. CSS saves a lot of work. It can control the layout of a number of multiple web pages all at once. It can be added to HTML elements in 3 ways

- **Inline** which means that it uses the style attribute in HTML elements
- **Internal** which means tat it uses a <style> element in the <head> section.
- **External** which means that it links to a CSS file. This is the recommended method btw.

Linking to a stylesheet looks something like this:

```
<link rel="stylesheet" href="styles.css">
```

This goes in the head tag of any web page. External style sheets have a few advantages, namely that a single style sheet can control a lot of pages. In general, you have a number of different pages that share a common style. You can define the style sheet in a single document and then have all of the HTML files refer to the same CSS file. It also facilitates *global changes* because if you're using external styles, you make a change in one place and then it's automatically propagated to all of the pages in the system. Finally, it allows for the *separation* of content and design. With the external CSS, all of the design is housed in the CSS and the data is in HTML.

Oh, also note that there are two kinds of reference: absolute (for example the absolute reference of a page like href="http://www.example.com/theme.css") while the relative looks like href="/themes/theme.css".

Here's an example of a style sheet:

```
h1 {
    font-family: sans-serif;
}

.lecture {
    font-weight: bold;
}

em { font-style: normal; }
em.room { font-style: italic; }
```

2.5 Data Formats and Operations

In 2015, Bristol elected a councillor for each of the 24 wards. Here are the results for this:

Candidate	Party	Votes
Chris Davies	Liberal Democrat	2435
Christopher Louis Orlik	Labour	1499
Glenn Royston Vowles	Green	722
Claire Lisa Louise Hayes	Uk Independence	625
Anthony Paul Lee	Conservative	590
Domenico Hill	Trade Unionists and Socialist Coalition	37

Here's what the structure would look like if it was written in C:

```
struct Candidate {
    char  name[100];
    char party[100];
};

struct Ward {
    char name[100];
    int electorate;
};

struct Result {
    struct Candidate candidate;
    struct Ward ward;
    int votes;
};
```

Here's how it would look in Java:

```
class Candidate {
    String  name;
    String party;
};

class Ward {
    String name;
    int electorate;
};

class Result {
    Candidate candidate;
    Ward ward;
    int votes;
};
```

And here's how we use the data structures

```
winner.candidate.name = "Chris Davies";
winner.candidate.party = "Liberal Democrat";
winner.ward.name = "Knowle";
winner.ward.electorate = 8820;
winner.votes = 2435;

System.out.println("In " + winner.ward.name + ", the winner was " +
winner.candidate.name + " with " + winner.votes + " votes.");

>> "In Knowle, the winner was Chris Davies with 8820 votes."
```

The question now is: How do we read data from storage? Well, what we shouldn't do is create our own data format (unless you're a big boy like Google/Microsoft). What we should do is to use the existing standards for encoding and storing data.

2.5.1 UNICODE and Encoding

Every file is a sequence of bytes that can further be broken down into bits. What the bytes mean are entirely dependent on how the are encoded and what sort of file type it is. 1 byte means that there are 256 possibilities. This is more than enough for most Western languages, but the stupid Chinese have something like 50k characters depending on how you count them. We also might want to encode more than the alphanumeric characters like icons, emojis, line drawing chars, mathematical symbols and so on.

ASCII was invented in the early 60's as a standard character set for computers and electronic devices. It's a 7-bit set containing 128 chars. It contains all of the English alphanumeric characters and some special characters. Unfortunately, once a lot more countries got involved in the internet, we needed a few more characters to allow us to represent their languages and alphabets. This is where Unicode came in.

The Unicode Consortium develops their Unicode standard. It replaces the existing character sets (ASCII) with a new one that is then implemented in many language such as HTML, XML, Java, and even the Latex I'm writing this in. Unicode defines a crazy 136k characters. **Encoding** is how these numbers are translated into binary numbers to be stored and processed in a computer. There are different ways that Unicode characters or code points can be encoded. There are some fixed width and some variable length encodings.

Line endings are kind of complicated, because in UNIX based devices, the ending is written as 'LF', or, in binary, '0000 1010'. However, in early mac, it's written as 'CR', or, in binary, '0000 1101'. Some systems even use a combination of the two.

Some network protocols have fixed conventions. For example, HTTP uses ASCII and defines a line ending as CRLF.

2.5.2 Tables and CSV

The simplest 'structured' file format is a list file with one entry per line. We can read or write to this in a loop. *Tables* are for 2-dimensional data. Another option we have is a CSV, or **comma separated values**. This might look like:

```
Candidate, Party, Votes
Chris Davies, Liberal Democrat, 2435
...
```

2.5.3 Stream Processing

In a stream, data items arrive one at a time and you only get to see them once each. We can use these streams if processing can be done with a single pass over the data, or if we only need to access recent data. We cannot do stream processing if we nee to do multiple passes through a data set or we need random access to items in the data set.

There are a few stream operations: filter, map, and reduce.

Filter means the we only read certain values that matches some condition.

Maps apply a function to each of the data items that are received through a stream.

Reduce applies a function to the whole stream and then output a single output.

We can also chain these operations.

Streams are really important because when we develop data driven web apps, we often need to process these streams of data, and if we bear these in mind when we write the code, we can structure it accordingly. *Map* and *filter* are stateless, per-element tasks. They are easy to parallelise. Some *reduce* operations can be done in parallel too.

2.6 Representing Data as Trees

If a list is 1-dimensional, and a table is 2-dimensional, what on earth is a 3-dimensional data? Actually, it's pretty easy, we just pick a third separator character.

Tree structure (or tree diagram) is a way of representing the hierarchical structure of data. It's called tree structure because it resembles a tree (duh doi) even though the diagram is generally upside down compared to a tree. The top most level is called the **root** while the bottom most ones are called the **leaves**.

Interestingly, most data can be represented as a tree. If we look at the candidate class from above:

```
<candidate>
    <name>Catherine Slade</name>
    <party>
        <name>Green</name>
    </party>
    <ward>
        <name>Bedminster></name>
        <electorate>9951</electorate>
    </ward>
</candidate>
```

Another way to represent this might be:

```
{
    "name:"Catherine Slade",
    "party": {"name:": "Green"},
    "ward": {"name": "Bedminster", "electorate": 9951},
}
```

Escape characters are used to signify that a character sequence needs to get special treatment from the same characters. Here are some now:

- \n is a line feed
- \r is a carriage return
- \\ is a back slash
- \” is a double quote.

2.6.1 XML

XML is pretty straightforward to use all over the internet. It’s also easy to write programs that process the XML documents.

HTML is all about displaying the information, while XML **describes** information. XML is the most common tool for data manipulation and data transmission. It can also be used for data storage. XML is both human AND machine readable, while also being flexible enough to support platform and architecture independent data interchange. XML allows a software engineer to create a vocabulary and use it to describe data (also sometime called being an **extensible** language).

The properties of XML include:

- Information identification
- Information storage
- Portable and non-proprietary
- Data transfer

The components of XML are:

- The declaration
- The root element
- Attributes
- Child elements
- Text data

In XML there are different steps for validation and processing. There are a total of 2 validation methods. The first is called DTD, or **Document Type Definition**. The other is called **schema**.

Here is an example of how this works;

Example XML:

```
<candidate>
    <name>Catherine Slade</name>
    <party>
        <name>Green</name>
    </party>
    <ward>
        <name>Bedminster></name>
        <electorate>9951</electorate>
    </ward>
</candidate>
```

DTD validation:

```
<?xml version="1.0"?>
<!DOCTYPE candidate [
<!ELEMENT candidate (name, party, ward)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT party (name)>
<!ELEMENT ward (name, electorate)>
<!ELEMENT electorate (#PCDATA)>
]>
<candidate> ... </candidate>
```

XML schema are another way of describing and XML document structure in XML itself. Nuts, right? Schemas are *more powerful* than DTDs. Also, an **XSD** is an **XML schema definition**.

Example schema:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="candidate">
        <xs:complexType> <xs:sequence>
    <xs:element name="name"type="xs:string"/>
    <xs:element name="party"> <xs:complexType> <xs:sequence>
        <xs:element name="name"type="xs:string"/>
    </xs:sequence> </xs:complexType> </xs:element>
        <xs:element name="ward"> <xs:complexType> <xs:sequence>
            <xs:element name="name "type="xs:string"/>
            <xs:element name="electorate"
                        type="xs:nonNegativeInteger"/>
        </xs:sequence> </xs:complexType> </xs:element>
    </xs:sequence> </xs:complexType> </xs:element>
</xs:schema>
```

XML entities are kinda like escape sequences, and they're also used in HTML. For example, there are:

- < is written by & lt;
- > is written by & gt;
- " is written by & quot;