**Chapter 1**

**What does web server do?**

A web browser lets a user request a resource. The web server gets the request, finds the resource, and returns something to the user. Sometimes that resource is an HTML page. Sometimes it's a picture or a sound file. Does n't matter the client asks for the resource and the server sends it back. Unless the resource is n't there or at least it's not where the server is expecting it to be(of course you are aware about 404 NOT FOUND error!).

**What does a web client do?**

Here web client is the browser app doing what user asked it to do, It shows the user the result of the request.

**But how do the clients and servers talk to each other?**

**1. HTML**

When a server answers a request, the server usually sends some type of content to the browser so that the browser can display it. Servers often send the browser a set of instructions written in HTML.HTML tells the browser how to present the content of the user.

**2. HTTP**

It is a protocol which client and server uses to communicate on the web. The client sends an HTTP request and server answers with an HTTP response.When a web server sends an HTML page to the client, it sends it using HTTP.

**What is the HTTP protocol?**

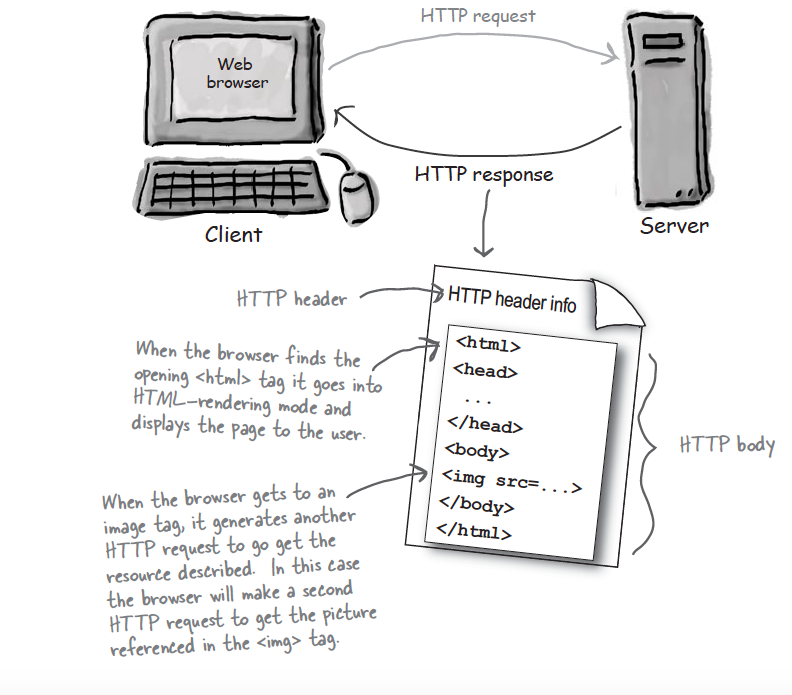
HTTP runs on top of TCP/IP. If you’re not familiar with those networking protocols, here’s the crash course: TCP is responsible for making sure that a file sent from one network node to another ends up as a complete file at the destination, even though the file is split into chunks when it’s sent. IP is the underlying protocol that moves/routes the chunks (packets) from one host to another on

their way to the destination. HTTP, then, is another network protocol that has Web-specific features, but it depends on TCP/IP to get the complete request and response from one place to another. The structure of an HTTP conversation is a simple **Request/ Response** sequence; a browser *requests*, and a server *responds*.

**Html is part of the HTTP response:**

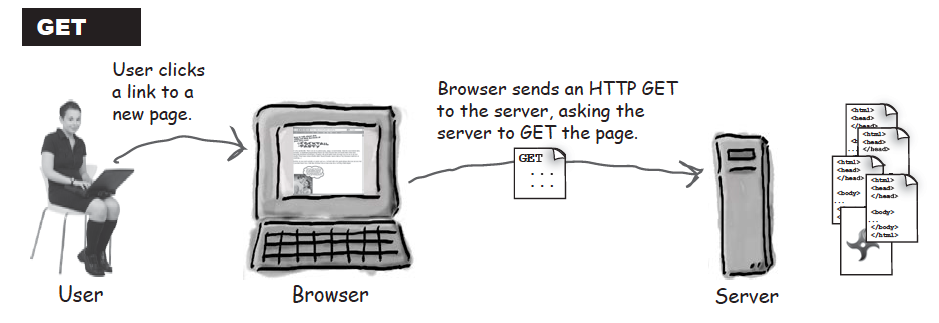
An HTTP response can contain HTML. HTTP adds header information to the top of whatever

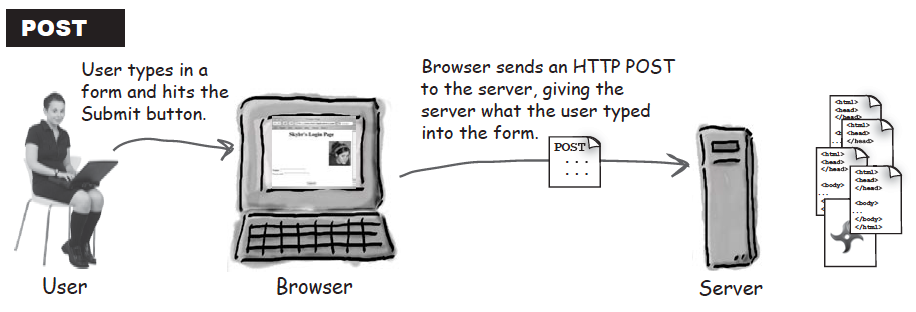
content is in the response (in other words, the thing coming back from the server). An HTML browser uses that header info to help process the HTML page. Think of the HTML content as data pasted inside an HTTP response.



**So what is in the request?**

The first thing you’ll find is an HTTP *method* name. These aren’t *Java* methods, but the idea is similar. The method name tells the server the kind of request that’s being made, and how the rest of the message will be formatted. The HTTP protocol has several methods, but the ones you’ll use most often are *GET* and *POST*.





**GET** is the simplest HTTP method, and its main job in life is to ask the server to *get* a resource and send it back. That resource might be an HTML page, a JPEG, a PDF, etc. Doesn’t matter. The point of GET is to *get* something back from the server.

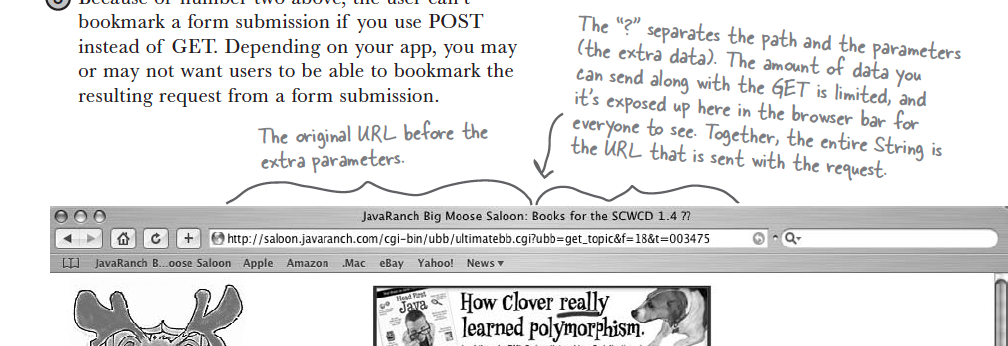
**POST** is a more powerful request. It’s like a GET plus plus. With POST, you can *request* something and at the same time *send* form data to the server .

You can send a little data with HTTP GET also, but you might not want to

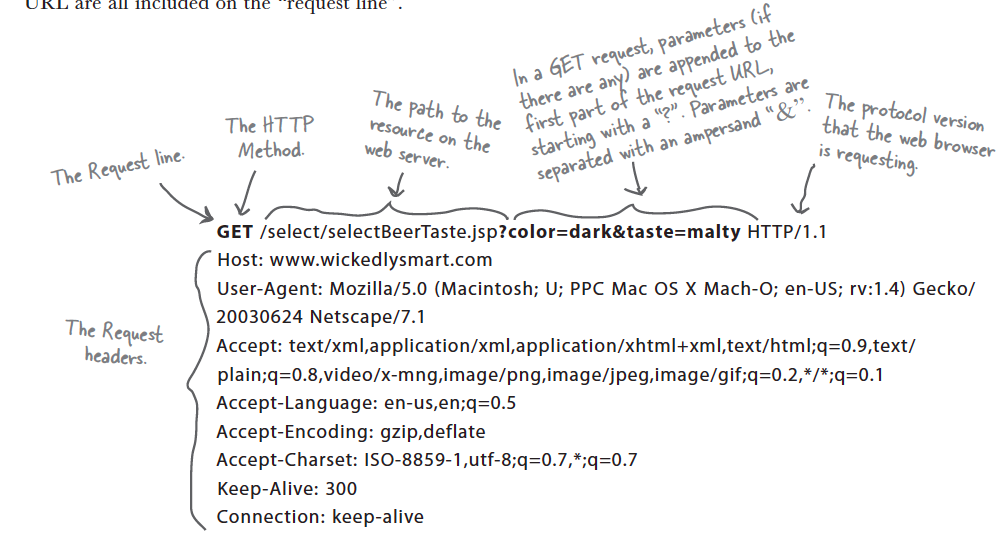
1. The total amount of characters in a GET is really limited (depending on the server). If the user types, say, a long passage into a “search” input box, the GET might not work.

2. The data you send with the GET is appended to the URL up in the browser bar, so whatever you send is exposed. Better not put a password or some other sensitive data as part of a GET!

3. Because of number two above, the user can’t bookmark a form submission if you use POST instead of GET. Depending on your app, you may or may not want users to be able to bookmark the resulting request from a form submission.

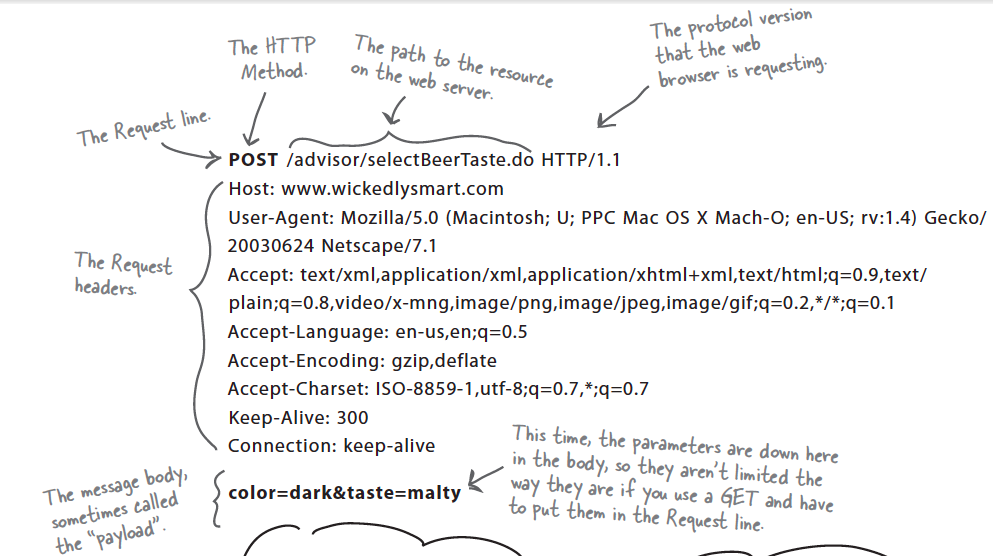


**Anatomy of an HTTP GET request:**



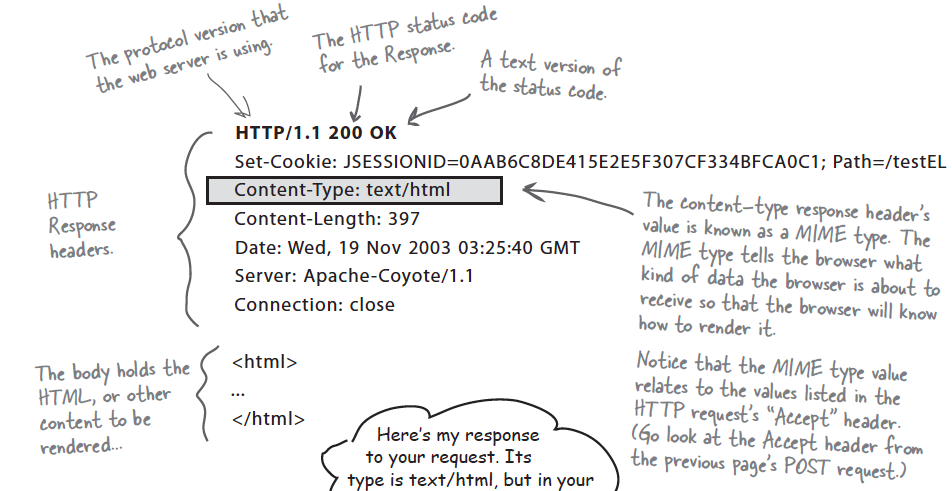
**Anatomy of an HTTP POST request:**

HTTP POST requests are designed to be used by the browser to make complex requests on the server. For instance, if a user has just completed a long form, the application might want all of the form’s data to be added to a database. The data to be sent back to the server is known as the “message body” or “payload” and can be quite large.

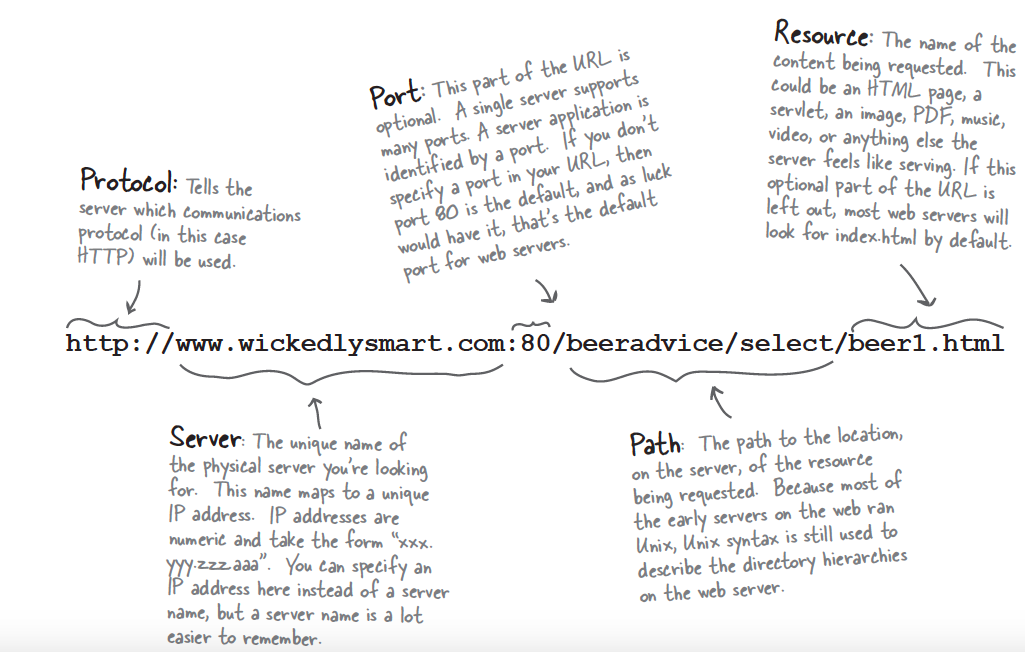


**Anatomy of an HTTP response**

Now that we’ve seen the requests from the browser to the server, let’s look at what the server sends back in response. An HTTP response has both a header and a body. The header info tells the browser about the protocol being used, whether the request was successful, and what kind of content is included in the body. The body contains the contents (for example, HTML) for the browser to display.



**Anatomy of URL :**



A

**Introduction To Web Applications**

Static pages just sits there in a directory. The server finds it and hands it back to the client as it is. Every client sees the same thing.

But what if I want, say, the current time to show up on my page?

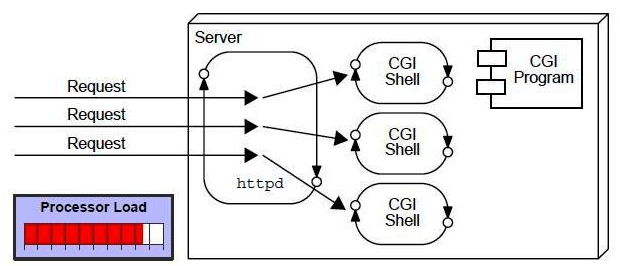
What if I want a page that has something dynamic (ok now we are talking about dynamic pages) .

Think of dynamic pages as Just in time pages don't exist before the request comes in. It's like making an HTML page out in the air. The request comes in, the application writes the HTML and gets it back to the client.

Web applications which produces dynamic contents also is known as Dynamic Web Application.

We have an old way to write dynamic web applications which is known as CGI(Common Gateway Interface).

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



There are many problems in CGI technology

1. If number of client increases, it takes more time for sending response.

2. For each request, it takes a process and Web server is limited to start processes.

3. It uses platform dependent languages like C, C++, Perl.

Sounds not effective!

Let's introduce Servlet Technology!

Servlet Technology is used to create Dynamic web application(more efficiently as compare to CGI ). Before digging into Servlets let's first discuss it's advantages over CGI:

1. Better performance: because it creates a thread for each request not process.

2. Portability: because it . JAVA …. Yipeeee !

3. Threads have a lot of benefits over the Processes such as they share a common memory area,

lightweight, cost of communication between the threads are low.

