

Aalay Tech Note

February 2005

All about the Web Programming Dr. Rama Kanneganti CTO, Aalayance Inc.

A semi-technical introduction to web programming for managers and Technical people.

Contents

1	History	1
1.1	HTTP Protocol	1
1.2	CGI Programming	2
1.3	Embedded languages programming	3
1.3.1	php	3
1.3.2	Perl – Embperl, Mason, Mod_perl	4
1.3.3	ASP	5
1.4	Servlet Programming	6
1.4.1	Tomcat	8
1.4.2	App Servers	8
1.5	Related technologies	9
1.5.1	ORMs	9
1.5.2	Page Customizations	10
1.5.3	Controlling applications	10
1.5.4	Future directions	11
1.6	How do you start with Java web development?	11
1.7	Colophon	11

1 History

Long time back, even before the web existed, people were finding files and getting services. FTP was a popular choice to exchange files. How do people know what is on any FTP? People used place a ls-lR (recursive listing of the files) in every folder. Of course, you had to get that file, examine it and determine what you need and then get back into FTP. It was cumbersome.

For interactive programs, there were email gateways. You specify what you need and send to an email server and the email server fulfills your request. I remember a recipe server from those days operated this way.

Emails and newsgroups let people know more about which FTP site has what. That is how people knew where to get what.

In the corporate intranets, people used to use dedicated Client/Server applications. That is, everybody has the application on their desktop and the application connected to the server. The protocols are proprietary. Normally, it is merely the database in the back end, but sometimes, it can even be a server program.

In the main frames, there is a terminal called TN3270. The central application would push out the text which is displayed on TN3270. (Cf: the modern web browser). There was no support for mouse or images or advanced fields. It was mostly function keys, and archaic commands like A/N*RT. Next time you are at a Travel agent, try looking at the computer screen: Most likely it is TN3270 at least an emulator) over a telephone line.

Into this landscape HTTP(WWW) appeared.

1.1 HTTP Protocol

Even before HTTP was popular, there were couple of services called gopher and WAIS that came into being. They were attempting solve the problem: what is where? For example, people connect a gopher server using gopher protocol and submit what they want like say: search for chocolate chip cookie. It will come back with all the file names from various servers that has those words. *Gopher digs for information*. WAIS also did something similar.

In the beginning HTTP browser was little more than a front end to ftp. Instead of getting file (ls-lR) to your desktop and reading it, you would read it on the fly. Instead of maintaining a dedicated connection and timing out, it would connect as needed. Of course, the text is little easier to read with bolds and italics. The markup was reminiscent of TeX and easy enough. The big plus point is the invention of URL. You could embed the link with the human readable name.

People started with writing overview material and placing the links. The browser was initially available for various Unixes (I used it on Solaris in 1993) only.

1.2 CGI Programming

Naturally, people wanted to interact with the web system via forms. Enter CGI – common gateway interface. The idea is simple. What ever information you submit is collected as variables and provided to an external program. This external program is expected to send back a page.

That is, each page is a program that takes the form variables as env vars, and returns a page as output. The web server waits till it gets all the output and pushes it out to the client, aka browser.

There are variations in this theme. The page need not be just HTML. It can be pdf, ps, xls, doc, txt and so on. By setting the mime-type in the header, you can intimate to the browser what to expect. Also, using some standard convention, you can tell the browser not to wait until all the page is received, but push out the data as it gets. For example, if you want to implement Unix "tail" command, you could use this "nph" (non-parsed-headers) programs.

The advantages of CGI ? There are plenty:

1. Every web server supports it. So easy to distribute and easy to install.
2. Can be written in any language. In C (cgic), Perl (CGI.pm), shell and so on.
3. Easy to understand model. Anybody can write a program in a few hours.

For all its advantages, there are severe drawbacks to CGI :

1. It is slow. Imagine every request forks a new web server which loads a Perl interpreter, which executes the program.
2. It is tough to maintain. For large programs, we need to evolve shared data so that each page can reuse other computations. Without that, each page has to repeat the computations.

Here is a sample HelloWorld program:

```
#!/usr/bin/perl
use strict;
# instantiate a new CGI object
my $cgi = new CGI ;

my $name = $cgi->param('name');
# perform a single print statement, with liberal use of the perl
# string concatenator "." and some CGI methods
print
```

```
$cgi->header .
$cgi->start_html('Hello ' . $name) .
$cgi->h1('Hello ' . $name) .
$cgi->end_html;

# Tell the web server everything is fine
exit (0);
```

Since HTTP does not maintain persistent connections, you cannot recognize if a user came before. That is, if a user entered his information earlier, how do we recognize him next time?

Cookies provide the answer. They are sent by the server for the browser to keep. They are like a ticket stub they hand you out at the movie halls. A server side program is free to set a cookie (with a name, a value, and expiration date – within limitations of course) and expect to get it back the next time the user visits. Obviously, you can't cram everything in one cookie, so you put a key in the cookie and the info in the database. To make sure cookie is not tampered, you pad it and encrypt it.

Cookies are a part of HTTP protocol and hence available as the only means to achieve persistence in the connection within HTTP.

1.3 Embedded languages programming

The first step forward from CGI is the "Page oriented languages". These languages do all that CGI can, but faster, and easier.

The trick is easy. Since the delay is mostly due to loading an external program each time, why not make the external program as a part of web server itself? That means, to execute a CGI programs, you do not have to start another process.

1.3.1 php

Among the page oriented languages, php is the leading one. It runs on Windows and Unix; it can be embedded in Apache and IIS; it can deal with multiple databases like Oracle, mysql, sqlite and multiple file types like pdf, flash. You can more information from [php homepage](#).

You write a php program as a mix of HTML and php commands. php commands can deal take a piece of data and parse it, put it in the database, get records from the database, generate flash or pdf and so on. It has rudimentary support for OO as well.

However, some problems remain: each page is a program unto itself. Suppose you want to deal with zip codes, you cannot create a zip codes object once for all and

reuse it in the rest of the application.¹ So, in each page, you have to load it up and initialize it.

Despite such problems php is hugely popular. It is one of the easiest to program with. The gentle slope of learning encouraged several people to try out php and consequently there is a large body of public domain code for us to borrow ideas from. In fact, several CMS solutions are written in php (See: [Opensource CMS](#)).

Here is some sample code written in php.

```
<html>
<body>
<h1>Hello User</h1>
This is a normal HTML document with php interspersed in it.
<?php
if ($HTTP_POST_VARS["name"]) {
    echo "<p>Hello, ".$HTTP_POST_VARS["name"]."!";
    echo "\n";
}
?>
</body>
</html>
```

Since it structurally looks like HTML, several HTML editors such as Dreamweaver work well with it. Web designers can work well with php, since it does not alter the page layout.

One of the early reasons that php succeeded was that it had built in db connection pooling. Turns out opening the database (in particular, Oracle) is very expensive. By caching the connections, php improved the performance of database backed web programs considerably. Since CGI and even sophisticated programming systems did not do that at that time, php became very popular.

1.3.2 Perl – Embperl, Mason, Mod_perl

Obviously, php is not everybody's cup of tea. For one thing, it is yet another language to learn. If you already learnt Perl and its quirks why not use it? Apache supports Perl as a first class solution in several different ways.

An Apache extension(module) called [mod_perl](#) provides a solution. It embeds a Perl interpreter in Apache. It supports initial loading of various Perl modules, a way to write apache extensions in Perl and execute Perl CGI programs directly in Apache without forking.

¹ There are advanced techniques using shared memory. However, they are too cumbersome and not usually found in applications.

Naturally, it speeds up the execution. It provides an easy upgrade path from CGI . Still, it is not as easy as php to code in.

If you look at php, the main advantage is that you can add bits and pieces of php to a web page to make it interactive. Mod_perl by itself does not do it.

Enter **Embperl**. It lets Perl statements embedded just like in php. It even supports db connection pooling. All in all, it was a good enough solution for many sites.

An advancement over embperl is **Mason**. It lets you build the system like bricks. Our intranet is built in Mason. Amazon is built with Mason. It lets us do a lot of magic like:

1. a standard script to display default page (the one rendering the list of our files is one such example).
2. a standard script when a file is not found
3. a component structure that lets us structure the site as components.
4. A way of accessing the arguments and templating the pages.

Here is a sample code:

```
<%args>
$name =>'Anonymous'
</%args>
<html>
<body>
<h1> Hello User </h1>
Hello <% $name %>!
</body>
</html>
```

If you are interested in a full-fledged example, take a look at /intranet/dhandler on linux1.

1.3.3 ASP

Microsoft's answer to php is ASP. It starts out with an embedded language that looks like VB. This framework along with the language handles cookies and sessions for you. If you are programming in MS Technologies, it lets you access the whole range of services via COM objects. I am unfamiliar with new ASP.net to comment intelligently about it.

Here is a small snippet of asp code. It is remarkably similar to php.

```
<html>
<%
dim name
```

```
name=Request.QueryString("name")
If name<>"" Then
    Response.Write("Hello " & name & "!<br/>")
End If
%>
</body>
</html>
```

1.4 Servlet Programming

All these page oriented languages, while easy to learn and program did not suit Corporate America. The reasons are as follows:

1. Lot more people are trained in procedural and OO languages where a persistent server serves out requests. Think Client/Server.
2. A page oriented system does not deal well with creating persistent servers side objects.
3. A page oriented system does not let you imagine the whole system as a program with each page as a function invocation. If you can do that, you can use several standard programming tools for writing and debugging programs.
4. Most of the web-only languages, by the need to co-exist with HTML, are not full fledged programming systems suitable to interoperate with other enterprise applications.

Java seems to provide satisfactory solution to this problem in Corporate America. It is a full-fledged language with introspection and garbage collection, and it works well within enterprise.

The first step in Java web application is servlets. Each servlet produces a HTML page. The servlet is a part of the servlet container. That is, there is a long running program called *servlet container* that has loads these servlets and executes them on demand from the users and sends out html.

What is the big deal? Unlike CGI , servlet has a clear life cycle. They are loaded when they are first needed and initialized. They are used whenever there is a need. Eventually they are release back to be garbage-collected. It is eerily similar to mod_perl modules.

For example, the Hello user program might look like this:

```
import Javax.servlet.*;
import Javax.servlet.http.*;
import Java.io.*;
import Java.util.*;
import Java.text.*;
```

```

public class FormDisplayServlet extends HttpServlet{
    public void init(ServletConfig config)
        throws ServletException {
        super.init(config);
        connections = 0;
    }
    //Process the HTTP Post request
    public void doPost(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
        String name = "";
        name = request.getParameter("name");
        response.setContentType("text/html");
        PrintWriter pw = new PrintWriter (response.getOutputStream());
        pw.println("<html>");
        pw.println("<head><title>Hello User</title></head>");
        pw.println("<body>");
        pw.println("<h1>Hello User</h1>");
        pw.println("Hello " + name);
        pw.println("</body></html>");
        pw.close();
    }
}

```

Verbose, isn't it? Most of it is mechanical. So, why would anybody want to write servlets?

Indeed. Just as in embperl or Mason makes it easy to embed Perl into HTML, jsp makes it easy to embed Java into HTML. The same page looks like the following in jsp.

```

<html>
  <body>
    <h1>Hello User</h1>
    Hello <%= request.getParameter("name") %>!

  </body>
</html>

```

Of course, this is a whole lot simpler. In fact, that is not where the story ends. Since the jsp server converts this page into a servlet, it can be made to do lot of magic. In particular, you can use new tags.

These tags can do wonderful things. Without you having to write Java code, they can take care of generating the HTML for you. They can do loops; they can do conditionals; they can do expressions. For example, to generate a table from a list of

rows such that it is paginated after every 10 records? You do not have to write code. Use an existing tag from [display tags](#). You can write your own tags too.

Tags let you make HTML look almost semantic, instead of purely syntactic documents.

1.4.1 Tomcat

Now, how do you execute a servlet? Using a servlet container. These days, the most popular choice is Tomcat, a servlet container from Apache.

The latest version 5.5.x of Tomcat supports the servlet version 2.4 and JSP version 2.0. That means, Tomcat closely tracks the standards. The license is business-friendly too. So, tomcat is a popular choice to run web applications using jsps.

Tomcat not only runs jsps, it even supports CGI , mod_proxy, and regular page serving. What it means is that you can serve out your site with static content, jsp pages, and CGI programs straight out of Tomcat, without resorting to additional web servers.

1.4.2 App Servers

Since Tomcat runs any web application, why do we care about so called "App Servers"? What do they do really?

App servers are lot more than servlet containers. Servlet containers is only a small part of the services they offer. They offer management of EJBs and various ways to interact with the external systems.

For example, these app servers host the components called Beans (EJB). They create them as per need, they invoke functions on them as per need (sometimes to rollback transactions) and destroy the beans. That is, they manage the life cycle of the EJBs. These beans are exposed to other applications through EJB protocols and CORBA, JMS. A web application can be put together through invocation of these beans. A GUI can be put together by Java GUI toolkit like Swing. These beans can be used in CORBA applications and communicate with other systems via JMS. They even can be used to provide Web Services.

All these services come at a price. If all you are doing is web application, EJBs are an overkill. They can take a perfectly running application and make it slow enough to need two machines. They solve this self-induced problem through built-in replication!

There are several choices for App Servers. On the high end, Weblogic not only satisfies the J2EE requirements, but comes with several management facilities, and ability to generate code to talk to other enterprise applications. It also supports facilities for fail-over and replication.

The other choices include Websphere (Solid, but difficult to manage), Oracle App server (Flaky, and difficult to manage), and JBOSS (Free, has some interesting ideas, but not entirely main stream).

1.5 Related technologies

For most purposes, if you are using Java for your web applications, Tomcat with JSP will suffice. There are lot of other options that help you address different issues. The foremost being ORM (Object Relational Bridge).

1.5.1 ORMs

Naturally, any interesting application on the web has a database at the back. To talk to the database, we need to use JDBC. There are several choices the way we can structure this database access:

1. You can keep the code in JSP. [Bad choice] The code becomes difficult to understand. It does not take advantage of caching. Every programmer needs to know Java as well as SQL. A symptom of this problem is that the system will have incorrect or suboptimal. If we have to extend/change the datamodel, we have to search all the jsps to correct any code.
2. You can keep the code in Java code and invoke it from JSP. [OK choice]. Even then, the same issues of Java developer needing to know SQL remains. Only the front end code is little less cluttered. Database modification will require extensive modifications to the Java code.
3. We can separate out SQL statements into a different layer and make calls to that library. [Seems OK, but cumbersome in practice]. Here, the SQL code is centralized, but each time the Java coder needs some new kind of data out of the database, new SQL code needs to be written. It is what is typically called *Leaky abstraction*.
4. We can generate Java objects that mirror the database. Any function calls onto these objects make a call to the database or use the cached information. This is called an Object-Relational mapping. There are several ways you can do these ORMs
 - a. We can hand code these classes. It is painful and error prone.
 - b. We can generate these classes out of SQL (or some enhanced xml from which SQL and these classes can be generated). The code can be bloated, but this is a good enough solution. For example, Torque from Apache can generate these classes. This technique has an added advantage of compile time checks for any database changes. That is, schema changes cascade to generated code

changes. If our code has different assumptions, then the Java compiler shows the errors.

- c. At run time, the system can provide the functionality via introspection. It produces the code that is the easiest to understand and cleanest to debug.

The most popular choice in ORMs these days is hibernate. There is a Java spec out there for OJB. We will have to wait and see if the landscape of ORMs changes because of that spec.

1.5.2 Page Customizations

If we look at jsp pages, large amounts of page is pure HTML. JSP provides some critical information that forms the complete page. If we want to change the layout of the page, we have to change HTML in each and every page. Obviously, it is error prone.

What are the choices?

1. We can encapsulate the HTML via includes
2. We can generate the HTML through Java procedures

Both these options work. If we elaborate on these options, we can come up with some frameworks.

1. **Velocity**: This is a framework that is based on template expansion. We can define macros which can be used in a page. Unlike tags, these macros provide the full power of Java. Though it is used in the context of web development, it has found uses in several other code generators as well.
2. **Tiles**: Tiles framework builds upon the simple “include” concept. It lets the developer arrange the page in tiles where each tile can be reused. They remind me of components in Mason where you can pass parameters as well.
3. **Java Server Faces**: This one is a mix of “uber-tags” and some functionality from struts – where you can validate the pages, render the UI based on some state kept on the server. If JSF was not there, you would have to write Java code to validate at the backend. Here, JSF code takes care of such validation and rendering as a part of the framework. It helps that it is an official standard from JCP.
4. **Tapestry**: This is the latest technology in page layout. In fact, it is touted as a replacement for JSP. For the people who know Zope, it is a bit like Zope templates. The salient features are these: You would describe the complete page in HTML, inserting dummy variables wherever needed. You mark the dummy variables as such and indicate to Tapestry what components/variables to replace them by. You specify using XML and Java in Tapestry framework what those components are.

1.5.3 Controlling applications

One issue with pure jsp pages is that the page flow can get complex. In a typical situation, you would have multiple pages, at each page you do error checking.

Eventually, you store in the database. If you hook up these pages in the JSP itself, it can become maintenance problem. Frameworks such as *struts* help you with that.

Struts gets the standard MVC (Model, View, Control) paradigm to the webworld. In pure jsp world, there is no control – a user requests and JSP does all the work. Here, a user request is handled by the control, which can setup/initialize a model or a make a call to the model and then go to the view of a page.

1.5.4 Future directions

One thing is certain. The needless complexity of J2EE spec is drawing lot of criticism. People are attacking that problem from multiple directions – by simplifying the spec, and coming up with alternates.

One approach called *Spring framework* is worth while checking out. It is makes web development easy – it is supposed to provide similar benefits as a J2EE server as far as web development is concerned, but without the overhead.

1.6 How do you start with Java web development?

Here is a step-by-step guide:

- Download and install J2SE SDK 1.4.2
- Download J2EE code as well as documentation for future reference.
- Install Eclipse and become proficient in it.
- Make sure you know basic Java: Objects, interfaces, static members, Collection classes, Logging, and basics of XML.
- Make sure that you know basic HTML: If you can use Nvu and understand what you are producing, you will do fine.
- [Appfuse](#) is a beginner toolkit for teaching Java web development. They provide a complete toolkit which lets you put several different combinations of solutions. They provide simple applications for you to tinker and learn.

I omitted several hyperlinks – you can google for them.

1.7 Colophon

This document is produced in ConT_EXt, using XEmacs as the editor. The result is proofread in Adobe Acrobat reader.