Server Side Scripting JavaScript

- ### History

Server-side JavaScript (SSJS) refers to JavaScript that runs on server-side and is therefore not downloaded to the browser. This term is used to differentiate it from regular JavaScript, which is predominantly used on the client-side (also referred to as client-side JavaScript or CSJS for short). The first implementation of SSJS was Netscape's LiveWire, which was included in their Enterprise Server 2.0 back in 1996. Since then, a number of other companies have followed suit in offering an alternative to the usual server-side technologies. One of the biggest players in the field was Microsoft. They supported the use of JavaScript on the server within what is now known as "classic" ASP. Along with the most common VBScript language, it also supported JavaScript and Perl Script. In reality, Microsoft utilized JScript, their own version of JavaScript. To use JScript/JavaScript, all you had to do was set the LANGUAGE attribute in the opening script tag:

Sample code:

```
1       <%@LANGUAGE="JavaScript"%>
2       <%
3       Response.Write("<HTML>\r")
4       Response.Write("<FONT COLOR=\"red\">\"Hello World\"</FONT><BR>\r")
5       Response.Write("</HTML>\r")
6       %>
```

Since the code runs on the server, what is sent to the client is the output of the script rather than the source code. Hence only the tags produced by the Response.Write() functions are found in the page source:

Sample code:

```
1       <HTML>
2       <FONT COLOR="red">"Hello World"</FONT><BR>
3       </HTML>
```

- ### Introduction

JavaScript has traditionally only run in the web browser, but recently there has been considerable interest in bringing it to the server side as well, thanks to the CommonJS project. Other server-side JavaScript environments include Jaxer and Narwhal. However, Node.js is a bit different from these solutions, because it is event-based rather than thread based. Web servers like Apache that are used to serve PHP and other CGI scripts are thread based because they spawn a system thread for every incoming request. While this is fine for many applications, the thread based model does not scale well with many long-lived connections like you would need in order to serve real-time applications like Friendfeed or Google Wave.

Node.js, uses an event loop instead of threads, and is able to scale to millions of concurrent connections. It takes advantage of the fact that servers spend most of their time waiting for I/O operations, like

reading a file from a hard drive, accessing an external web service or waiting for a file to finish being uploaded, because these operations are much slower than in memory operations. Every I/O operation in Node.js is asynchronous, meaning that the server can continue to process incoming requests while the I/O operation is taking place. JavaScript is extremely well suited to event-based programming because it has anonymous functions and closures which make defining inline callbacks a cinch, and JavaScript developers already know how to program in this way. This event-based model makes Node.js very fast, and makes scaling real-time applications very easy.

- **What is Node.js?**

Node.js, often referred to as just Node, is a powerful tool that can run JavaScript applications on both the server side as well as the client side. Node.js can be used to write static file servers, Web application frameworks, messaging middleware, and servers for HTML5 multiplayer games. This article is a very elementary introduction to Node.js.

Node.js is a server side JavaScript built on Googles V8 JavaScript engine. It is an open source programming language that was developed by Ryan Dahl in 2009. It allows us to build scalable network applications, and is very fast when compared with other server side programming languages because it is written in C and the non-blocking I/O model. It is currently sponsored by Joynet, a software company specialising in high performance container native infrastructure. Node.js can run on Linux, Sun OS, Mac OS X and Windows platforms.

The steps below explain the difference between blocking code and non-blocking code models.