ELE548 Research Essay: Topic 3

Java Applets vs. Java Servlets

Zhengqiang Shan

April 12, 1999

Abstract

An applet is a Java program that can be included in an HTML page and executed by a Java technology-enabled browser. It can be used in web-page enhancement and on enterprise intranets for sharing resource and data. Applets allow local validation of data entered by the user, can Using database to perform list of values lookups and data validation, and have Complex GUI widgets. Servlets are protocol- and platform-independent server side components, which runs as part of a network servic and dynamically extend Java-enabled servers. With applets, servlets can provide interactivity/dynamic Web content generation. Servlets can accept form input and generate HTML Web pages dynamically like CGI, support collaborative applications by synchronizing request, partition a single logical service between several servers, act as active agents sharing data. Servles support different protocol, can be part of middle tiers in enterprise networks. Servlet are mor appropriated when involves loading large pieces of code over a slow communication channel; when a large part of the computation for generating the Web page can be done on the server side or when processing involves operations that applets cannot perform due to security restrictions. If a sophisticated user interface is desired, applets are appropriate. Applets and servlets can share data and communicate, so the processing can be split between them. Client-side Java is a glorious vision that does and will not change the way most people use the Internet anytime soon. Sun has taken some dramatic steps to insure that Java remains ubiquitous by creating a new set of APIs that allow developers to use Java as a server-side development tool.

1 Java Applets

1.1 Introduction

An applet is a program written in the Java programming language that can be included in an HTML page, much in the same way an image is included. When you use a Java technology-enabled browser to view a page that contains an applet, the applet's code is transferred to your system and executed by the browser. Web browser severely restricts what an applet can do in terms of file system and network access in order to prevent accidental or deliberate security violations. With applets you have the ability to perform serious high end interactive programming tasks that cannot be performed with Dynamic HTML.

1.2 Capabilities and Limitations of Applets

Java applets obviously have many potential capabilities because of their tight security model, their generally small size, their and network awareness.

Java applets can be used to build full-featured graphical user interfaces, communicate over the Internet to a host server, and even communicate with other applets on a form. All of this can be done in an operating-environment-neutral manner, which is what makes this such a great technology. For Java to be truly successful, however, the client security has to be completely assured. Because of this, security measures place some limitations on Java applets. By default, applets cannot communicate with any server other than the originating server. Applets also cannot read or write files to the local file system.

The growth of technologies such as Web-based client/server application development and electronic commerce has been severely limited by the lack of "industrial-strength" security. Because the underlying Internet was never designed to handle secure transactions (the Department of Defense has a separate net for this purpose), the entire infrastructure of the Internet was somewhat unprepared for the phenomenal growth of the World Wide Web over the last few years. The concept of applets (or related technologies such as software agents) has been discussed in academic circles for years, yet most theoreticians realized the security

shortcomings of the current programming languages such as C and C++.

1.3 Life Cycle of Applet

An applet as an object inherits the properties of its parent object, the applet template. The Applet Package contains a few methods that have some very special functions. These methods, called the lifecycle methods, control how an applet behaves during the course of execution.

When you load a Web page that contains an applet, the applet goes through several stages during the time you see it on-screen. During each of these stages, the applet performs very different tasks, although most of these tasks are invisible to the end-user. These stages are initialization, running, and exiting.

During the initialization stage, the applet is loading the images, sound clips, and other resources that it needs to run. When the applet has all the resources it needs to run, the initialization stage is over, and the applet is ready to run.

When the applet is running, it is performing whatever tasks it has been designed to perform. Conversely, when an applet is not running, it is just sitting idle, waiting for a user to re-enter the Web page. Because applets start and stop when you enter and leave a Web page, running consists of two distinct states, starting and stopping. These states could really be thought of as two separate stages, and in fact, each has a corresponding lifecycle method. You can control what an applet does during both starting and stopping.

Because applets are loaded into your machine's memory and use CPU time, you wouldn't want the applet to remain in memory once you've left the Web browser. During the final exiting stage, the Java Virtual Machine completes some garbage collecting functions, making sure that the resources the applet used are removed from memory and that the applet is completely destroyed when you quit.

Breaking up applets into these stages has some very distinct advantages. For example, if you were writing an animator that used a large number of images, you would want to make sure the images were loaded before the applet started running. Otherwise, your animation

might seem jerky or have frames missing. Manipulating these stages can come in handy, and fortunately the Applet Package contains methods to do just that.

The Applet Package has four lifecycle methods, each of which corresponds directly to the stages of an applet. Each of these methods is automatically called as the applet loads, runs, and exits, so you might not always use each of these methods in your own applets. Also, you only need to use these methods if you need something specific to occur during a particular stage, like stopping a sound when you leave the page. Often, you will use one or two lifecycle methods, but not all of them. The decision to use a lifecycle method depends largely on what you are trying to do with your applet. You will find that init(), start(), and stop() are all used fairly commonly because these stages each have practical implications for applets. You want to make sure images and sounds load first, you want to make sure sounds stop playing, and so on. Evaluating the need to use one of these methods is a part of the planning process when writing your applets.

The first method called by an applet once it has been loaded by the browser is init(). Because the applet is not running when init() is called, this method is an excellent place to take care of any groundwork that must be laid for the applet to carry out its goals. Some good tasks to take care of during init() include loading images or establishing the user interface. Take, for example, an applet that plays an audio clip at the click of a button. In such an applet, you would need to load the audio clip and set up the button before the applet began to run.

After the applet has been loaded into the browser and is ready to begin, the start() method is called automatically. The start() method generally contains the meat of your applets. After all, this method is what you want the applet to do. In most applets, you will define the init() and start() methods. You can put any kind of code in the start() method; you could draw images, play sounds, and accept user input, essentially any of the functions you might expect a program to perform.

The stop() method is the counterpart to the start() method. It is called automatically when an applet should stop execution, when you leave an applet's Web page. If you use the

start() method to start some functions that need to be stopped before the user moves on, you stop them with the stop() method. You don't necessarily have to redefine the stop() method in all your applets. If an applet is simple enough, you could let the stop() method automatically terminate any methods that might be running. But if you have any sounds playing, or especially if you have any threads running, it is a good idea to invoke the stop() method on your own to keep your applet under control.

The destroy() method is essentially the death of an applet. When you leave your browser, this method is called automatically to do any cleanup that might be necessary. Just as the name would imply, the destroy() method eliminates any trace of your applet. It purges any memory that was used by your applet, and it stops any running threads or methods. Generally speaking, you do not have to do anything to use the destroy() method, a base destroy() method is predefined and automatically called, so all you have to do is sit back and let it do the dirty work.

1.4 Applet vs. HTML

Applets allow local validation of data entered by the user. Local validation of data is possible using HTML combined with JavaScript but variances in JavaScript implementations make JavaScript difficult to generally use.

An applet can use the database to perform list of values lookups and data validation. HTML (even if combined with JavaScript) can not do that without invoking a CGI or servlet program and drawing a new HTML page.

Once an applet is downloaded, the amount of data transferred between the Web browser and the server is reduced. HTML requires that the server transfer the presentation of the data (the HTML tags) along with the data itself. The HTML tags can easily be 1/4 to 1/2 of the data transferred from the server to the client.

Applets allow the designer to use complex GUI widgets such as grids, spin controls, and scrollbars. These widgets are not available to HTML.

2 Java Servlets

2.1 Introduction

Java code that runs as part of a network service, typically an HTTP server and responds to requests from clients is called Java Servlet. Servlets are protocol- and platform-independent server side components, which dynamically extend Java-enabled servers. They provide a general framework for services built using the request-response paradigm. For example, a client may need information from a database; a servlet can be written that receives the request, gets and processes the data as needed by the client and then returns the result to the client. Their initial use is to provide secure web-based access to data which is presented using HTML web pages, interactively viewing or modifying that data using dynamic web page generation techniques. Since servlets run inside servers, they do not need a graphical user interface. Otherwise, they are the server side counterpart to applets: they are Java application components which are downloaded, on demand, to the part of the system which needs them.

Servlets are most often provided by organizations which provide customized multi-user services to their customer bases. However, servlets are also flexible enough to support standardized services such as serving static web pages through the HTTP (or HTTPS) protocols, and proxying services. And since they are used for dynamic extensibility, they may be used in a plug-in style, supporting facilities such as search engines and semi-custom applications

2.2 Java Servlet Usage

- Protocol support is one of the most viable uses for servlets. For example, a file service can start with NFS and move on to as many protocols as desired; the transfer between the protocols would be made transparent by servlets. Servlets could be used for tunneling over HTTP to provide chat, newsgroup or other file server functions.

- Servlets could play a major role as part of middle tiers in enterprise networks by

connecting to SQL databases via JDBC. Corporate developers could use this for several applications over the Intranet, extranet, and Internet.

- Servlets often work in conjunction with applets to provide a high degree of interactivity and dynamic Web content generation.

- The most common use for servlets is to accept form input and generate HTML Web pages dynamically, similar to traditional CGI programs written in other languages. A simple servlet can process data which was POSTed over HTTPS using an HTML FORM, passing data such as a purchase order (with credit card data). This would be part of an order entry and processing system, working with product and inventory databases and perhaps an on-line payment system.

- A community of servlets could act as active agents which share data with each other.

- Since servlets handle multiple requests concurrently, the requests can be synchronized with each other to support collaborative applications such as on-line conferencing. One could define a community of active agents, which share work among each other. The code for each agent would be loaded as a servlet, and the agents would pass data to each other.

- One servlet could forward requests other servers. This technique can balance load among several servers which mirror the same content. Or, it could be used to partition a single logical service between several servers, routing requests according to task type or organizational boundaries.

2.3 Life Cycle of Servlet

Servlets support the familiar programming model of accepting requests and generating responses. This model is used with a variety of distributed system programming toolsets, ranging from remote procedure calls to the HTTP requests made to web servers. Servlets implement the Servlet interface, usually by extending either the generic or an HTTP-specific

implementation. The simplest possible servlet defines a single method, service. The service method is provided with Request and Response parameters. These encapsulate the data sent by the client, providing access to parameters and allowing servlets to report status including errors. Servlets normally retrieve most of their parameters through an input stream, and send their responses using an output stream.

Servlets are always dynamically loaded, although servers will usually provide an administrative option to force loading and initializing particular servlets when the server starts up. Servlets are loaded using normal Java class loading facilities, which means that they may be loaded from remote directories as easily as from the local filesystem. This allows for increased flexibility in system architecture and easier distribution of services in a network. The life cycle of a servlet is:

- Server loads and initializes the servlet: When a server loads a servlet, the server runs the servlet's init method. Initialization completes before client requests are handled and before the servlet is destroyed.
- The servlet handles client requests: An HTTP Servlet handles client requests through its service method. The service method supports standard HTTP client requests by dispatching each request to a method designed to handle that request. HTTP servlets are typically capable of serving multiple clients concurrently.
- The server removes the servlet: Servlets run until the server are destroy them. When a server destroys a servlet, the server runs the servlet's destroy method. The method is run once; the server will not run that servlet again until after the server reloads and reinitializes the servlet.

2.4 CGI and Servlets

Servlets provide an alternative mechanism to CGI programs for generating dynamic data. CGI programs have existed for a while; they are stable and universally accepted. They are

language-independent (although they are not platform-independent). The main advantages of servlets over CGI scripts are:

- Performance Servlets offer a substantial improvement in performance over CGI. Each CGI request on the same server results in the creation of a new process. On the other hand, a servlet can continue to run in the background after servicing a request. Also, CGI programs are not threaded. Servlets can use threading to process multiple requests efficiently, provided that the JVM embedded in the Web server offers thread support.

- Platform Independence CGI programs are platform-dependent. Servlets are Java classes and follow the "write once, run everywhere" doctrine. Therefore, they are truly portable across platforms.

- State Information CGI programs are stateless because they result in the creation of a new process each time a request is serviced. A servlet has memory of its state once it is loaded by the server. The JVM running on the Web server loads the servlet when it is called. The servlet does not have to be reloaded until it changes, and a modified servlet may be dynamically reloaded without restarting the server. Maintaining state information allows multiple servlets to share information.

- Network Programming Your Java servlets have full access to Java's networking features. The servlets can connect with other networked computers using sockets or Remote Method Invocation (RMI). Also, the servlet can easily connect to a relational database using the Java Database Connection (JDBC). By using the networking features of Java, servlets can be used to easily develop middleware.

- Reuse and Modularity One shortfall of server-side programming in scripting languages such as Perl and VBScript is that of reuse. If you have to create another server-side module based on existing code then the only reuse you have with scripting languages is "cut-and-paste" reuse.

Since servlets are written in Java, you gain all the object-oriented features of Java such as reuse. You can create an object framework of common servlets and reuse them in future applications. For example, you can create a simple servlet for processing of HTML form data. Later, another developer can use this servlet as is or extend it to add custom functionality.

Supporting the idea of modularity, servlets can communicate with other servlets on the Web server. This mechanism, known as servlet chaining, allows the output of one servlet to be passed as input to another servlet. As an example, a database query servlet can retrieve sales data and pass this data to a charting servlet. The charting servlet simply prepares a graphical representation of the data and returns it to the client.

3 Java Applets vs. Java Servelets

Servlets are named after applets which are also written in Java but which run inside the JVM of a HTML browser on the client. Servlets and applets allow the server and client to be extended in a modular way by dynamically loading code which communicates with the main program via a standard programming interface.

Basically, a servlet is the opposite end of an applet. A servlet can almost be thought of as a server-side applet. Servlets run inside the Web server in the way that applets run inside the Web browser. The browser can submit a request to execute a servlet directly; it can be stand-alone in terms of its actions – as a browser can request an applet directly.

Since servlets run inside servers, they do not need a graphical user interface, often referred to as faceless applets. Servlets are free of the security restrictions that apply to applets. This is because they run within a Web server on the server-side. Thus, they are trusted programs, Java application components which are downloaded, on demand, to the part of the system which needs them.

Like applets, servlets may be called from HTML files dynamically and there are several

cases in which the two could be used interchangeably. So when should we design servlets and when should we design applets? The answer to this question goes back to the basic issue of load distribution between the client and the server. The distributed client/server paradigm has shifted over the past few years from fat clients to thin clients and subsequently from thin servers to fat servers. Applets are representative of the client side of the architecture and servlets represent the server side. Some scenarios in which servlets are more appropriate are given below:

1. Applet classes are downloaded over the Internet to the client and then executed in a JVM running on the client. If this involves loading large pieces of code over slow modem lines, applets are not the appropriate choice.

2. If a large part of the computation for generating the Web page can be done on the server side, it is pointless to load the part of the code that does the computation to the client. The computation should be done on the server and the results passed back to the client.

3. If processing involves operations that applets cannot perform due to security restrictions, then a local servlet may be used.

Applets are more appropriate in the following scenarios:

1. Applets basically have a well-defined user interface (remember that they derive from Panel). In servlets, on the other hand, a user interface would have to be built from scratch. Therefore, when a sophisticated user interface is desired applets are appropriate.

2. If the speed of the communication channel is adequate, then the overhead involved in downloading applets may not be an issue.

Applets and servlets can also share data and communicate. Therefore, the processing can be split between them.

4 Discussion

It's a peculiar moment in the industry's history. The Java buzz is intense. And yet when you look at the Web applications that people actually use every day to do their work, you invariably find that there are no Java applets in the mix. The universal client today is still the HTML browser. The universal client of tomorrow will be the HTML/JavaScript browser. Client-side Java is a glorious vision that will not change the way most people use the Internet anytime soon. It's just more than what the majority of today's computers and networks can readily push. So what are millions of people running every day? Server-based applications that feed the universal HTML client.

Server-side scripting is still very popular for several reasons:

- totally independent of the browser since everything takes place on the server;
- complex requests may execute faster on the server;
- can be made safer since the programs run under direct control of the server administrator.

Java servlets give you the capability to develop complex server-side applications. Servlets leverage Java's object-oriented features to build reusable and modular components. You can easily create servlets to replace CGI applications, access databases and communicate with remote computers.

As promising as applet development is, developers' frequent lack of control over the final appearance and performance of their products is often frustrating. The limitations on applet capabilities and the changes between Java versions make it difficult to apply the full strength of Java's incredibly powerful programming structures, restricting the real-world use of Java fairly severely. As applet development becomes a specialized field for multimedia toys and carefully-built applications, Sun has taken some dramatic steps to insure that Java remains ubiquitous – and useful – by creating a new set of APIs that allow developers to use Java as a server-side development tool.

5 Reference

1. http://jserv.javasoft.com:80/products/java-server/documentation/ webserver1.0.2/servlets/api.html
2. http://www.sys-con.com/java/feature/3-1/servletsfriends/
3. http://www.sys-con.com/java/feature/3-1/cgi-scripts/index.html
4. http://www.servletcentral.com/common/articlelist.dchtml
5. http://webreview.com/wr/pub/97/10/10/feature/colton.html
6. http://www.developer.com/news/techfocus/021698_servlet.html
7. http://www.sys-con.com/java/feature/3-2/3-tier/index.html
8. http://www.servletcentral.com/
9. http://javaboutique.internet.com/
10. http://www.javasoft.com/applets/index.html
11. http://www.javasoft.com/features/1997/oct/applets.html
12. http://www.javasoft.com/features/1997/dec/applets2.html
13. http://www.javasoft.com/features/1998/07/applet.power.iii.html