0471431036_01.qxd 7/30/03 3:41 PM Page 1

CHAPTER 1

# An Overview of Resin

During the early years of the Web, developers used a fairly small set of HTML document tags to manipulate the style and layout of each Web page. These static Web pages were usually stored on the same computer that was executing a Web server. When a user launched a Web browser on his or her local machine and pointed that browser to a Web server, a protocol-based exchange of information began. The page sent to the user was really just a text file with information and HTML tags embedded in it to present the information in a clear manner.

As time went on, it became clear that providing up-to-date and personalized information was going to be a developer's nightmare if using static pages was the only option. Keeping the static pages up-to-date was feasible, but personalizing that information manually was next to impossible. This meant that the pages had to be created on the fly in response to an action or request by the user. One early solution to this problem was called *Common Gateway Interface* (CGI). Figure 1.1 shows an example of how requests are made using CGI and a Web server.

Unfortunately, there were some problems with using CGI for dynamic Web page generation. First, the CGI protocol was designed to allow a Web server to communicate with an external program and not specifically for the generation of dynamic Web pages. The creation of the Web pages is actually a side effect of the CGI protocol. As shown in Figure 1.1, the protocol is written so that the Web server will create new process spaces in which the CGI application will execute. All information passed from the user's Web browser to the Web server

0471431036_01.qxd 7/30/03 3:41 PM Page 2



is passed as environment variables to the waiting CGI application. All of these operations are expensive in terms of processor time as well as memory utilization.

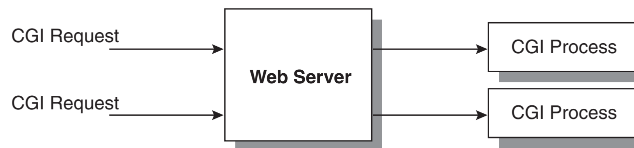



**Figure 1.1** CGI request flow.

The second problem is the fact that most CGI applications were written in Perl, with C and C++ coming in second. Depending on your programming capability, Perl could be a stumbling block to Web development.

After the success of CGI became apparent, an alternative implementation of CGI was created called FastCGI. Figure 1.2 shows the request flow in FastCGI.

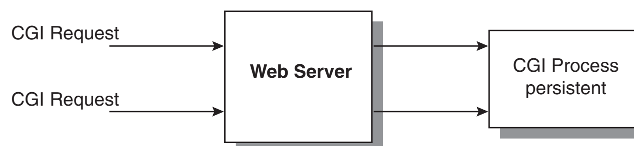



**Figure 1.2** FastCGI request flow.

The primary difference between CGI and FastCGI is the creation of persistent processes in which the CGI application executes. By using persistent processes, the Web server doesn't have to expend processor time and memory constantly creating and destroying processes. The success of FastCGI led to the development of other products, like mod_perl and PerlEx, to address performance and interface issues.

The overall success of CGI was dependent on the fact that most Web servers on the market supported its use. This included open-source servers, such as Apache; Internet Information Services (IIS) from Microsoft; and iPlanet from Netscape. However, at the same time these Web servers started experimenting with newer technologies like Active Server Pages (ASP) and Server-Side JavaScript (SSJS).

Both ASP and SSJS solutions to the problem of dynamic Web pages employ the concept of embedding code into the HTML page where dynamic content is needed. ASP, for example, uses JScript or VBScript. Figure 1.3 shows the process flow for ASP and SSJS.

As Figure 1.3 shows, a user requests an ASP page from a Web server that supports ASP, usually IIS. The page is pulled into the Web server and processed.

0471431036_01.qxd 7/30/03 3:41 PM Page 3



The processing involves executing the ASP or SSJS code within the server and replacing the code with HTML tags and content. The resulting static page is sent to the browser.

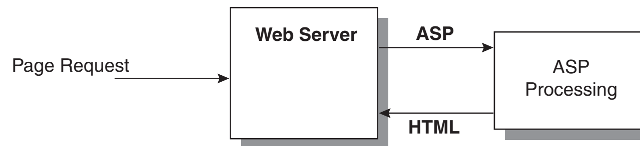



**Figure 1.3** ASP/SSJS process flow.

One of my core beliefs is that all programming languages—both conventional and Internet based—should be compared with each other only when we're discussing a specific solution. Is Java better than Cobol? Well, for some applications clearly it is, but in some business situations, fancy GUIs and RMI just aren't needed. The same is true for things like ASP and servlets. Therefore, at this point, let's move to a discussion of JSP, servlets, and beans, which entered the scene after ASP and SSJS.

In the hierarchy of power, the JavaServer Pages (JSP) technology probably comes next. JSP pages are similar to both ASP and SSJS in that code is embedded in the HTML of a Web page. However, JSP isn't designed to be specific to one Web server. As long as a Web server can interpret the Java code embedded in the Web page, it will be able to handle JSP pages. Some of the real power in JSP comes from its ability to interact with JavaBeans. Beans are modular pieces of Java code designed to perform a specific job. Beans can be referenced from JSP pages and automatically instantiated on the server machine.

Finally, we have servlets. A servlet is an extension to a Web server written as a class that will be instantiated to provide additional functionality to the server. Figure 1.4 shows the process flow for a Web page that utilizes servlets.

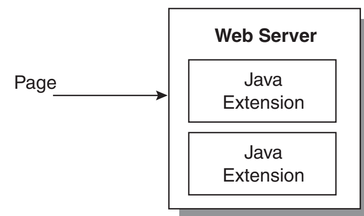



**Figure 1.4** Servlet processing.

When a Web page that includes calls to a servlet is requested by a browser, the HTML is processed by the Web server and the servlet calls are farmed to a servlet engine. The servlet engine will instantiate the servlet code, process the request, and return any results to the Web server. Servlets are powerful because they:

0471431036_01.qxd 7/30/03 3:41 PM Page 4



- Remain in memory after the first instantiation.
- Are portable to any platform that can execute Java.
- Have access to the full power of the Java language and its supporting classes.

You can use servlets for database access, distributed access using Remote Management Interface (RMI) or Common Object Request Broker Architecture (CORBA), and a host of other operations.

In the remainder of this chapter, we cover the Resin server product family and take a more in-depth look at the features provided by the servers.

## What Is Resin?

Resin is a high-performance XML application server for use with JSPs, servlets, JavaBeans, XML, and a host of other technologies. The 2.1.x line of servers is fully Servlet 2.3 and JSP 1.2 compliant and beats all of the competition in number of operations per second, as shown in the data obtained from www.caucho.com/articles/benchmark.xtp. Released in August 2003, Resin 3.x is JSP 2.0 and Servlet 2.4 draft compliant. As the drafts become standard, Resin 3.x will be updated to reflect any changes.

Over the past few years, the Resin family of servers has evolved with the needs of its users. As of this writing, two Resin application servers are available. Resin is the core product, and it supports all of the basic features defined in this chapter. Resin Enterprise enhances the core product with Enterprise JavaBeans (EJB) container-managed persistence (CMP) and database caching. Distributed objects are included using the Burlap protocol as well as full support for the EJB 2.0 specification. In the remainder of this chapter, we outline the major features of the Resin application server.

### Resin/Resin Enterprise Features

The following features are available in both Resin and Resin Enterprise:

- JSP/servlet support
- Serif/XTP
- XML/XSL
- Internal and external Web servers
- Load balancing
- Distributed sessions
- Security
- Virtual hosting

0471431036_01.qxd 7/30/03 3:41 PM Page 5



- Web services
- JSP-EL
- JDK 1.4 Logging

Let's take a look at each of these features.

### JSP

As mentioned earlier, the Resin application server fully supports the Java JSP 2.0 specification. JSP pages can use Java as well as JavaScript to embed code within the HTML tags and text on the page. However, beginning with Resin 3.x, JavaScript is not supported; therefore you will only want to use Java in your JSP pages. The server can handle XML-based JSP notation like the following:

```
<jsp:directive.page import="hello.Name" />
<jsp:useBean id="mybean" scope="page" class="hello.Name" />
<jsp:setProperty name="mybean" property="*" />

<html>
<head><title>Hello</title></head>
<body>

<h1>What's your name?</h1>

<form method="get">
<input type="text" name="username" size="25">
<br>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</form>
</body>
</html>
```

The Resin server allows you to create custom tag libraries to add better internal documentation to Web pages. Full support is provided for Web applications pulling together JSP pages, scripts, and beans.

### XSL

JSP, servlets, and XML Template Pages can take advantage of Resin's ability to process Extensible Stylesheet Language (XSL) or Extensible Markup Language (XML) style sheets. By using XSL, basic XML can be transformed to HTML or another output, such as Wireless Application Protocol (WAP). Depending on the needs of the user, different style sheets can be created and used against XML to produce the desired output. In the following example, a JSP page instructs the Resin server to use the new.xsl style sheet against the provided XML:

0471431036_01.qxd 7/30/03 3:41 PM Page 6



```
<%@ page session=false contentType='x-application/xslt' %>
<?xml-stylesheet href='new.xsl'?>
<datarow>
 <title>Output Just For You</title>
 <data>data</data>
</datarow>
```

### Web Server Support

Out of the box, Resin is a fully self-contained application server as well as a Web server. There is no need for any additional server software. The internal Web server is very fast and, in most cases, outperforms the competition. Resin also supports these external Web servers: Apache, Netscape, IIS, and O'Reilly WebSite.

### Load Balancing

There's no need to worry about losing customers or visitors to your site when it's powered with a Resin server. Resin supports the use of a hardware load balancer for higher cost situations and a low-cost solution using Resin itself. The LoadBalanceServlet servlet is executed on one front-end machine, and it distributes user requests to numerous back-end servers. The Resin server also includes an HTTP proxy cache to make the entire system more efficient.

### Distributed Sessions

For sites that must use sessions, Resin supports sticky and distributed sessions. Through file-based and in-memory hashtables, customer won't have problems with shopping carts when requesting transfers to different servers in a server farm or when a server crashes. There is support for database persistent sessions as well as TCP-based distribution.

### Security

The Resin server includes four security mechanisms for use in applications:

**Authentication through XmlAuthenticator—**This allows username/password combinations to be placed in configuration files.

**Authorization for protecting areas of a Web site—**The areas of the site can be protected using various constraints: pattern-based for specific pages, roles-based for specific users, IP-based for keeping pages specific to specific machines, and transport-based for limiting pages to SSL. Further authorization can be constrained using custom controls.

**Encryption of data—**Resin supports OpenSSL for encrypting data. Full support is provided for certificates.

0471431036_01.qxd 7/30/03 3:41 PM Page 7



**Security Manager for separating virtual uses in an ISP situation—** The manager will put each Web application in its own separate security environment.

## *Virtual Hosting*

If you are in a situation where numerous users or departments will be using Resin, it's a good idea to configure virtual hosting. In this mode, each host has its own directory structure. Advanced configuration allows each host to have its own Java Virtual Machine (JVM) for complete separation.

## *Web Services*

The developers of Resin have incorporated two different Web service protocols to support this new Internet technology. The protocols are called Burlap Web Service Protocol and Hessian Binary Web Service Protocol.

### Burlap

Burlap Web Service Protocol is designed to be a lightweight protocol for allowing EJB services to communicate with non-Java servers and clients. The underlying protocol is all XML-based for maximum interoperability. The protocol is simple, and using a Burlap service results in behavior similar to instantiating an object and calling a method. A basic client would look like this:

```
public class Client {
  public static void main(String[] args)
    throws Exception {
    BurlapProxyFactory factory = new BurlapProxyFactory();
    TestService test = (TestService) factory.create(
      TestService.class, "http://www.gradecki.com/burlap/testSer-
vice");
    System.out.println("Call doIt(): " + TestService.doIt());
  }
}
```

Building a Burlap service is just like creating a standard Java class, with the public methods exposed or made available to remote clients. Here's the code for the service called by our client:

```
public class TestService extends BurlapServlet implements Basic {
 public String doIt()
 {
 return "called doIt()";
 }
}
```

0471431036_01.qxd 7/30/03 3:41 PM Page 8



You can use the protocol in Java 2 Micro Edition (J2ME) devices, and it also supports object serialization.

#### Hessian

The Hessian Binary Web Service Protocol is designed to be a simple protocol just like Burlap but is optimized for sending binary data between servers and clients. The API for Hessian is just as simple as Burlap, and it supports serialization as well.

## Resin Enterprise

The remaining features are specific to Resin Enterprise:

- CMP
- XDoclet
- EJB-QL

### *CMP*

For those of you who have written servlets—or just about any type of Web application that uses a database—the acronyms ODBC and JDBC may make you tremble. When writing servlets that connect with a database, JDBC is the common interface used. The developer is responsible for creating the connection to the database, building the SQL, and executing the calls. With EJB container-managed persistence (see http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBConcepts4.html#62950), most of the work is done for you. Resin helps even further by intelligently caching the database data. Full relations can be created between CMP beans like 1-n, 1-1, and others using a series of XML-based configurations.

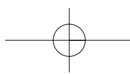
### *XDoclet*

Resin Enterprise supports XDoclet, which is an extension to JavaDoc available at http://xdoclet.sourceforge.net/. The custom Javadoc @tags created by Resin are: resin-ejb:entit-bean, resin-ejb:cmp-field, resin-ejb:entity-method, and resin-ejb:relation.

### *EJB-QL*

The Enterprise server supports Enterprise JavaBeans Query Language (EJB-QL) in its use of container-managed persistence. EJB-QL (which is defined by

0471431036_01.qxd 7/30/03 3:41 PM Page 9

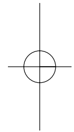


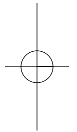


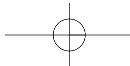
Sun at http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html) allows for support of portable data stores and maintains SQL '92 compliance.

## What's Next

As we've seen, the Resin family of products is quite comprehensive and full of features. This chapter has only touched on the high points. In the next chapter, we begin the process of installing and learn how to bring out the best in the application servers.







0471431036_01.qxd 7/30/03 3:41 PM Page 10