



Using Java servlets to generate dynamic WAP content

24.1	Generating dynamic WAP content	380	24.4	Invoking a Java servlet	385
24.2	The role of the servlet	381	24.5	Processing client requests	388
24.3	Generating output to WAP clients	382	24.6	Summary	393

24.1 *GENERATING DYNAMIC WAP CONTENT*

You have seen in part VI how Microsoft ASP technology is used to generate dynamic WML content in a WAP-based application. The concept of performing the same task using Java servlets is much the same.

The main issues undertaken by a typical snippet, be it an ASP document or a Java servlet, that is capable of performing backend processing as well as generating results for a WML-enabled client dynamically are:

- Retrieving the values of WML variables from the calling or referring context
- Specifying the correct content type of the WML deck to be generated

- Overriding specific header information, such as the caching mechanism, as necessary
- Generating valid WML content to form a well-defined WML deck

This chapter will explain how to invoke a servlet and how to use a servlet to process client requests and generate dynamic WML content.

24.2 THE ROLE OF THE SERVLET

Before we plunge into writing servlets, we should first look at where the servlet fits in the whole picture of a typical WAP-based application (figure 24.1):

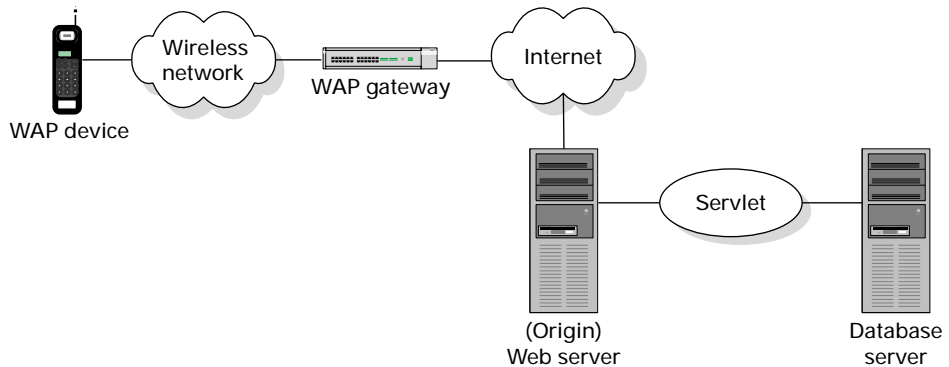


Figure 24.1 Components in a WAP-based application system

As discussed in chapter 22, a servlet processes a client request that is passed from the web server. In a WAP-based application environment, the web server receives the client (i.e., WAP device) request from a remote WAP gateway via the Internet. Alternatively, the WAP gateway and the web server can reside on a single machine. The setup and infrastructure regarding the connectivity of the various components is covered in greater detail in part VIII. Recall that the application-level protocol used in the communication between the WAP gateway and the web server is HTTP. The servlet must understand an HTTP request message that is redirected to it from the web server, as well as generate an HTTP response message to be returned to the client via the web server.

This is exactly what a servlet is doing in a web-based application environment described in chapters 22 and 23. However, one major difference lies in the content it generates. In servicing a web-based client, the servlet generates HTML content that the client can interpret. In servicing a WAP-based client, the servlet generates WML content so that the client can understand.

Hence, in addition to any server-side processing an application may require, a servlet must generate an appropriate HTTP response message, which consists of the response header information such as the MIME content type and caching information,

as well as the body of the response. The response body that is intended for a WAP client is a WML deck. Both the header information and the body content are generated as an output stream by the servlet.

24.3 GENERATING OUTPUT TO WAP CLIENTS

To generate an HTTP output stream, we can use a servlet that is derived from `HttpServlet`, which defines interfaces and classes that we can use to handle HTTP requests and create HTTP responses. Refer to appendix F for a listing of the interfaces and classes in the `HttpServlet` API specification.

This section will show the specific methods an `HttpServlet` object uses to set the header information and to output the response body to a WAP client.

24.3.1 Writing the response header information

As mentioned in chapter 22, the methods for setting or overriding an HTTP header are included in the `HttpServletResponse` object of an `HttpServlet`. We will revisit two relevant methods in this section. For a list of HTTP response headers that can be used in an HTTP response message, see appendix E.

setContentTypes()

For the user agent to interpret the received content correctly, we need to set the content-type header field of the response message to the user agent. The servlet uses the `setContentTypes()` method to accomplish this. A valid content type of a WML message for a WAP browser is `text/vnd.wap.wml`.

The following statement sets the appropriate content type for a response containing a WML deck. The `HttpServletResponse` instance, *response*, outputs the generated content:

```
response.setContentType("text/vnd.wap.wml");
```

setHeader() and setDateHeader()

To set a header field with a string value, use the `setHeader()` method. Use `setDateHeader()` to add a header with a date value.

The following statement sets the value of the `Cache-Control` header field to `no-cache`, signaling to the receiving proxy server not to cache the received message content.

```
response.setHeader("Cache-Control", "no-cache");
```

For more details about the basic issues on caching, see chapter 19.

24.3.2 Writing the response body

After setting the header information, the servlet is responsible for generating all the relevant content in the response body that constitutes a valid WML deck. Recall from

chapter 22 that the servlet uses a writer stream (`PrintWriter`) object or an output stream (`ServletOutputStream`) object to generate output data to a client.

To obtain a stream object, the servlet uses one of the following declarations depending on the type of stream you choose:

```
PrintWriter out = response.getWriter();
ServletOutputStream out = response.getOutputStream();
```

Subsequently, use the `print()` method or the `println()` method to write the output messages.

The following servlet code shows how a writer stream generates the XML prologue that is found in a typical WML deck:

```
"PrintWriter out = response.getWriter();"
out.println("<?XML version=\"1.0\"?>");
out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
out.println("<\"http://www.wapforum.org/DTD/wml_1.1.xml\">");
```

24.3.3 Creating a WAP application using a servlet

We will now look at a servlet example that writes a response header and generates a simple WML deck. `HelloServlet` generates a message that appears in a WML browser:

Listing 24.1 Source code for `HelloServlet`

```
/* Generate a greeting: HelloServlet.java */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet
{
    public void init(ServletConfig config) throws ServletException {
        super.init (config);
    }

    public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException
    {
        doPost (request, response); ❶
    }

    public void doPost (HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/vnd.wap.wml");
        PrintWriter out = response.getWriter(); ❷
        out.println("<?XML version=\"1.0\"?>"); ❸
        out.println("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
        out.println("<\"http://www.wapforum.org/DTD/wml_1.1.xml\">");
    }
}
```

Initializes the servlet when it is first loaded into memory

❶

❷ Sets the value of the MIME content type of the generated message to the client

❸ Creates a `PrintWriter` instance

```

out.println("<wml>");
out.println("<card id=\"start\">");
out.println("<p>");
out.println("Hello, here's a dynamically generated greeting!");
out.println("</p>");
out.println("</card>");
out.println("</wml>");
out.close();
}
}

```

← Releases the `PrintWriter` object by using the `close()` method

Generates an output stream that contains the `<wml>` element, the `<card>` element, and the card content

Code comments

- ❶ Handles incoming HTTP GET requests. It consists of only one statement: a call to `doPost()`. The effect is to redirect HTTP GET requests to `doPost()`. Hence, we need to write the client request servicing code only in `doPost()` since in our application, we intend to handle both types of requests in a similar manner.
- ❷ Originally handles only incoming HTTP POST requests. In our application, it is also called from within `doGet()` to handle incoming HTTP GET requests.
- ❸ Prints an output stream that consists of the XML prologue required in a WML deck.

24.3.4 Viewing the result

Using the UP browser emulator (UPBrowser), you can view the WML deck that the `HelloServlet` servlet generates. To run the servlet:

- Compile the Java servlet and save it under the appropriate directory where the servlet loader you use can locate it.
- Invoke the browser emulator and enter the URL of the compiled servlet, `HelloServlet.class`, in the Go input box of the browser interface.
- In figure 24.2, for testing purposes, we use `localhost` as the host of the URL, and the path, `/servlet`.



Figure 24.2
URL of a servlet residing on the local host

A text message should appear in the microbrowser (figure 24.3).

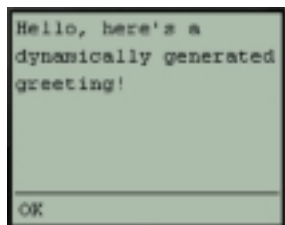


Figure 24.3
Display of WML content generated by `HelloServlet`

From the emulator, press F5 to display the source content in the Phone Information window. Figure 24.4 shows the dynamic content generated by the servlet and delivered to the client via the WAP gateway. The content is what one would expect to see in a .wml document.

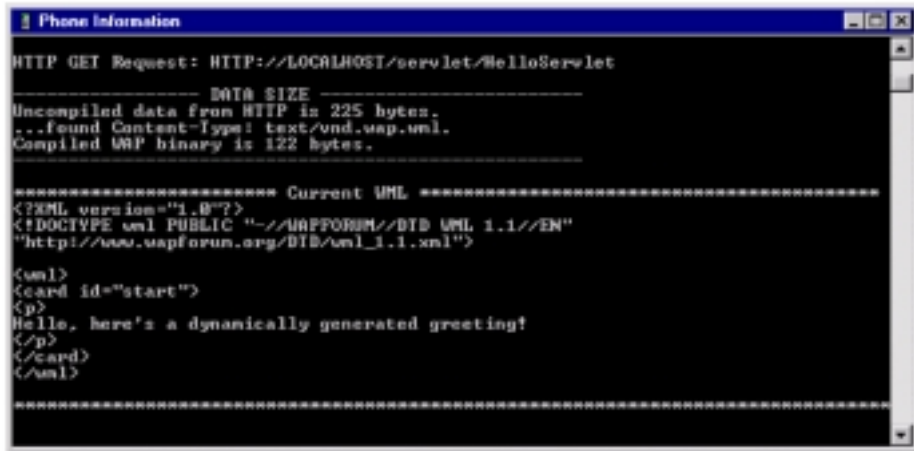


Figure 24.4 Phone Information window displaying the WML code generated by HelloServlet

24.4 INVOKING A JAVA SERVLET

Although you can invoke a servlet by entering its URL directly in the browser interface, it is more often the case that a servlet is invoked from a WML card. You can also pass parameter values from the calling WML card to the servlet. In this section, we will present two other common ways to kick off a servlet.

24.4.1 Calling a servlet from a card

To call a servlet from a card, use the WML `<go>` or `<a>` element.

Using the WML `<go>` element

To see how the WML `<go>` element works, we will use our HelloServlet example (listing 24.1).

The following code uses the `<go>` element to invoke HelloServlet when Accept is clicked:

```

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml> <!-- hello1.wml -->
<card id="start" title="hello">
  <do type="accept" label="servlet">

```

← Gives the Accept button of the phone interface the label, `servlet`. Clicking this key will kick off the task specified in the `<go>` element

```

        <go href="http://myHost:port/servlet/HelloServlet" method="get" />
    </do>
    <p>
        Click "servlet" to invoke HelloServlet!
    </p>
</card>
</wml>

```

Causes the current card to pass control to HelloServlet via an HTTP GET method

The screens in figure 24.5 show what the WML code looks like using UP.Browser.

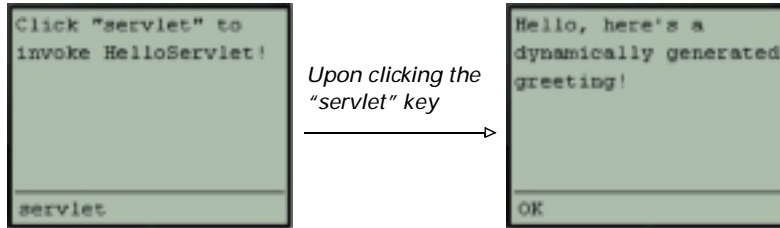


Figure 24.5 Invoking HelloServlet from hello1.wml using the `<go>` element

In our example, if a client invokes the servlet using the HTTP GET method, the request is first passed to the `doGet()` method, which in turn calls the `doPost()` method to service the request. In this way, we need only to furnish the code for `doPost()`, which handles POST requests directly and the GET requests via redirection.

If no appropriate service method can be found in the servlet, the typical error code issued by the origin server is HTTP Error 405.

Using the WML `<a>` element

The WML deck shows how a servlet can be invoked using the WML `<a>` element:

```

<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml> <!-- hello2.wml -->
<card id="start" title="hello">
    <p>
        Navigate to:
        <a href="/servlet/HelloServlet">HelloServlet</a>
        <a href="#end">Ending card</a>
    </p>
</card>
<card id="end">
    <p>
        Thank you for visiting!
    </p>
</card>
</wml>

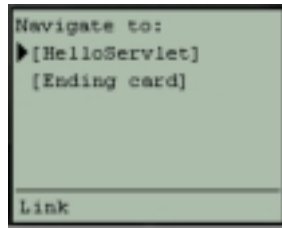
```

A link to the servlet, HelloServlet

A link to the last card in the same WML document

Figures 24.6 and 24.7 show what the WML code looks like using UP.Browser.

hello2.wml#start



Generated by HelloServlet

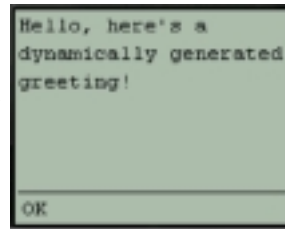
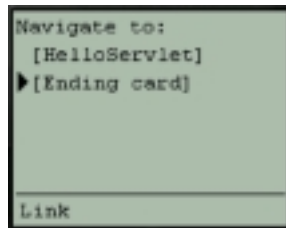


Figure 24.6 Using `<a>` in `hello2.wml` to invoke `HelloServlet`

hello2.wml#start



Generated by HelloServlet

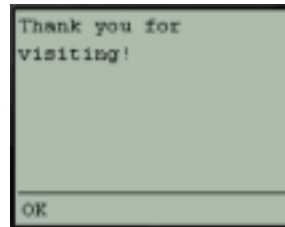


Figure 24.7 Using `<a>` in `hello2.wml` to invoke another card in the same WML deck

In general, there is no difference between the use of `<go>` and `<a>` unless you need to pass parameters to the referenced servlet as will be discussed next.

24.4.2 Passing parameter values

The way to pass parameters from the calling document to the servlet depends on how the servlet is invoked.

Using explicit query string

One straightforward means to pass parameter values to the referred servlet is to append the variable's name and value pairs as a query string after the URL of the servlet.

Here are four ways of using a query string to pass the values of two parameters, `myname` and `mypwd`:

- Supplying parameter values to the `Logon` servlet via the browser interface:

```
http://localhost/servlet/Logon?myname=susan&mypwd=wap
```

- Supplying parameter values to the `Logon` servlet via the `<a>` element:

```
<a href="http://localhost/servlet/Logon?myname=susan&mypwd=wap">
  My login
</a>
```

- Supplying parameter values to the Logon servlet via the `<go>` element:

```
<go href="http://localhost/servlet/Logon?myname=susan&mypwd=wap" />
```

- Supplying parameter values to the Logon servlet via the `onpick` event of the `<option>` element:

```
<option
  onpick="http://localhost/servlet/Logon?myname=susan&mypwd=wap" />
  My login
</option>
```

Using the `<postfield>` element

Another way of passing parameter values from a WML deck to a servlet is through the WML `<postfield>` element, which is used with the `<go>` element.

This code invokes a servlet with the user-input values of two variables, `Name` and `Passwd`, using the `<postfield>` element for each value to be passed to the servlet:

```
<do type="accept" label="servlet"> ❶
  <go href="http://myHost:port/servlet/HelloServlet" method="post"> ❷
    <postfield name="myname" value="$Name" />
    <postfield name="mypwd" value="$Passwd" /> ❸
  </go>
</do>
```

Code comments

- ❶ Assigns the label `servlet` to the Accept button of the phone interface. Pressing this key will kick off the task specified in the `<go>` element.
- ❷ Causes the current card to pass control to `HelloServlet` via an HTTP `POST` method.
- ❸ Specifies the name that the servlet must use to retrieve the corresponding value. Here, `Name` and `Passwd` are variables in the card's context, but `myname` and `mypwd` are not visible within the card. The latter two variables can be retrieved from the referenced URL (i.e., `HelloServlet`) specified in the `<go>` element.

24.5 PROCESSING CLIENT REQUESTS

The way a servlet processes a request from a WML client is identical to the way it handles a request from an HTML client. Typically, a servlet is initialized when it is invoked. It then processes individual requests using the appropriate service method such as `doGet()` or `doPost()`.

In this section, we will use a simple example to illustrate the way a servlet processes a request from a WML client.

24.5.1 Retrieving header information

Just as there are `HttpServletResponse` methods that a servlet can use to set HTTP header information, there are methods that a servlet can use to retrieve HTTP

header information from an HTTP request message. For a list of HTTP request headers, see appendix E.

There are four `HttpServletRequest` methods capable of retrieving information about the headers in an incoming request:

- `getHeader("header_name")` returns the value of the specified header as a string.
- `getHeaders("header_name")` returns all the possible values of the specified header as an array of strings.
- `getDateHeader("header_name")` returns the value of the specified header as a *long* number, indicating the number of milliseconds elapsed since midnight, Jan. 1, 1970, UTC. It can be converted into a Java `Date` object.
- `getIntHeader("header_name")` returns the value of the specified header as an integer.

Next is a simple example to retrieve information about the HTTP headers, `Accept` and `User-Agent`. If the user agent is a WML client, the `HTTP_ACCEPT` header should contain the string `text/vnd.wap.wml`, while the `HTTP_USER_AGENT` header provides clues to the type of browser used.

The code illustrates how a servlet determines whether to respond with WML content specifically for the UP browser or WML content for other WML-enabled browsers.

```
String acceptHeader = request.getHeader("Accept");
String userAgentHeader = request.getHeader("User-Agent");

if (acceptHeader.indexOf("wml") != -1 )
    if (userAgentHeader.indexOf("UP") != -1)
        generateDeckforUP();
else
    response.sendRedirect ("http://host/WML/nonUP.wml");
}
```

Code comments

- ① Uses the `getHeader()` method to retrieve both the headers, `HTTP_ACCEPT` and `HTTP_USER_AGENT`.
- ② Uses the `indexOf()` method to check if the retrieved headers contain the key substrings that indicate support for WML content and use of a UP browser.
- ③ If both of these conditions are satisfied, invokes an arbitrary method, `generateDeckforUP()`.
- ④ If the user agent is not a UP browser but does support WML, it redirects request to a static WML document, `nonUP.wml`.

For more information on the use of headers such as `Accept` and `User-Agent` to help determine the client browser using ASP, see chapter 19.

24.5.2 Retrieving parameter values

To retrieve the values passed from a WML deck to a servlet, use the `getParameter()` or `getParameterValues()` methods discussed in chapter 22.

The following WML deck invokes the servlet, `WelcomeServlet`, on the local host for retrieving a name value entered by the user. When `WelcomeServlet` is called, a parameter is passed to the servlet:

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml> <!-- welcome.wml -->
<card id="start" title="welcome">
  <do type="accept" label="login">
    <go href="http://localhost/servlet/WelcomeServlet" method="post">
      <postfield name="myname" value="$Name" />
    </go>
  </do>
  <p>
    What is your name?
    <input name="Name" type="text" maxlength="30" />
  </p>
</card>
</wml>
```

Invokes the servlet, `WelcomeServlet`, via the HTTP POST request method

Parameter name is `myname` and the value is that entered by the user under the name, `Name`

Prompts the user to input a value for the variable, `Name`, as a text field

The result is displayed in figure 24.8.

Listing 24.2 is the source code for the `WelcomeServlet`.

Listing 24.2 Source code for `WelcomeServlet`

```
/* Library Welcome Message: WelcomeServlet.java */
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class WelcomeServlet extends HttpServlet
{
  /* Initialization of servlet */
  public void init(ServletConfig config) throws ServletException {
    super.init (config);
  }

  /* Redirecting client request that uses HTTP GET method */
  public void doGet (HttpServletRequest request,
                    HttpServletResponse response)
    throws ServletException, IOException
  {
    doPost (request, response);
  }

  /* Servicing client request using HTTP POST method */
  public void doPost (HttpServletRequest request,
```

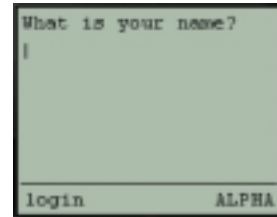


Figure 24.8 Message from the only card in `welcome.wml`

```

        HttpServletResponse response)
        throws ServletException, IOException
    {
        String username = request.getParameter ("myname"); ❶
        response.setContentType("text/vnd.wap.wml");
        PrintWriter out = response.getWriter();           ❷

        out.println ("<?xml version=\"1.0\"?>");        ← Outputs the XML prologue
        out.println
            ("<!DOCTYPE wml PUBLIC \"-//WAPFORUM//DTD WML 1.1//EN\"");
        out.println ("\"http://www.wapforum.org/DTD/wml_1.1.xml\">");

        out.println("<wml>");
        out.println("<card id=\"start\" title=\"welcome\">");
        out.println("<p>");
        out.println("Hello, " + username +
            ", welcome to the XYZ Library!");
        out.println("</p>");
        out.println("</card>");
        out.println("</wml>");
        out.close();
    }
}

```

Sets the content type for WML content
 Outputs the retrieved user's name and a welcome message

Code comments

- ❶ Retrieves the user's name. Since the parameter name used to pass in the value was myname the servlet must use the same parameter name to retrieve the user's name.
- ❷ Creates a writer stream instance for use in generating dynamic content.

24.5.3 Understanding the code

Figure 24.9 shows what the WML code looks like using UP.Browser.



Figure 24.9 Passing name parameter to WelcomeServlet from welcome.wml using <go> and <postfield>

The Phone Information window displays the variable and value sent to the servlet via the HTTP POST request (figure 24.10).

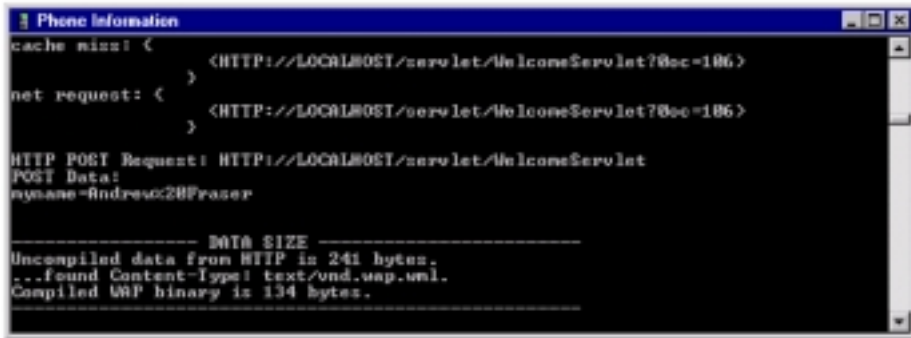


Figure 24.10 Viewing parameter value posted to `WelcomeServlet` in Phone Information window

The variable, `myname`, has the value `Andrew%20Fraser` where `%20` denotes the hexadecimal value of the ASCII code for the space character.

Next, we shall show another WML deck to illustrate an alternative way to invoke `WelcomeServlet`, namely via the WML `<a>` element. The way to pass in a variable value is through a query string appended to the URL, i.e., the HTTP GET request method.

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml> <!-- welcome1.wml -->
<card id="start" title="welcome">
  <p>
    What is your name?
    <input name="Name" type="text" maxlength="30" />
    Navigate to:
    <a href="/servlet/WelcomeServlet?myname=$Name">
      Welcome
    </a>
    <a href="#end">
      Ending card
    </a>
  </p>
</card>
<card id="end">
  <p>
    Thank you for visiting!
  </p>
</card>
</wml>
```

Prompts user to input value for the variable, `Name`

Creates a hotlink labeled `welcome` that links to `WelcomeServlet`

Creates a hotlink labeled `Ending card` that links to the last card in the same deck

← Card referred to in the second hotlink

The use of the second welcome WML document results in similar displays (figure 24.11):



Figure 24.11 Passing name parameter to `WelcomeServlet` from `welcome1.wml`

The Phone Information window displays the variable and value sent to `WelcomeServlet` via a query string attached to the URL of the servlet, using the HTTP GET request method (figure 24.12).

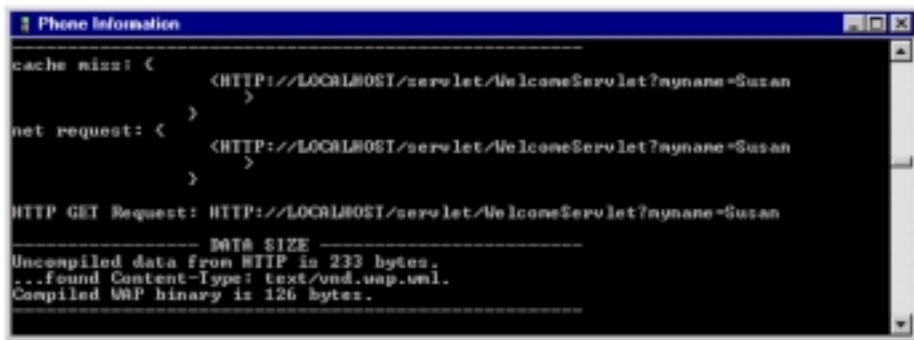


Figure 24.12 `WelcomeServlet` is invoked via an HTTP GET request

24.6 SUMMARY

In this chapter, we recapture the main tasks involved in generating dynamic content using server-side technology. Specifically, we see how a Java servlet can be invoked from a WML card as well as the ways of passing parameter values to the servlet.

Once the servlet is loaded, depending on the application on hand, the servlet may need to retrieve the HTTP header information and the incoming parameter values before performing further processing.

To generate output, it requires an output stream or writer object as described in chapter 22. To generate WML output to a WAP client, the servlet needs to set the correct content type, and in order that the client interprets the incoming WML response

as intended, the developer is responsible for ensuring in his code the generation of a valid WML deck.

In the next chapter, you will see how we apply what you have learned in this chapter in applications that involve cookie and session, which are commonly used techniques to retain information about the state of the client.