4.6 Filtering Strings for HTML-Specific Characters

Normally, when a servlet wants to generate HTML that will contain characters like `<` or `>`, it simply uses `&lt;` or `&gt;`, the standard HTML character entities. Similarly, if a servlet wants a double quote or an ampersand to appear inside an HTML attribute value, it uses `&quot;` or `&amp;`. Failing to make these substitutions results in malformed HTML code, since `<` or `>` will often be interpreted as part of an HTML markup tag, a double quote in an attribute value may be interpreted as the end of the value, and ampersands are just plain illegal in attribute values. In most cases, it is easy to note the special characters and use the standard HTML replacements. However, there are two cases in which it is not so easy to make this substitution manually.

The first case in which manual conversion is difficult occurs when the string is derived from a program excerpt or another source in which it is already in some standard format. Going through manually and changing all the special characters can be tedious in such a case, but forgetting to convert even one special character can result in your Web page having missing or improperly formatted sections.

The second case in which manual conversion fails is when the string is derived from HTML form data. Here, the conversion absolutely must be performed at runtime, since of course the query data is not known at compile time. If the user accidentally or deliberately enters HTML tags, the generated Web page will contain spurious HTML tags and can have completely unpredictable results (the HTML specification tells browsers what to do with legal HTML; it says nothing about what they should do with HTML containing illegal syntax).

Core Approach

If you read request parameters and display their values in the resultant page, you should filter out the special HTML characters. Failing to do so can result in output that has missing or oddly formatted sections.

Failing to do this filtering for externally accessible Web pages also lets your page become a vehicle for the *cross-site scripting attack*. Here, a malicious programmer embeds `GET` parameters in a URL that refers to one of your servlets (or any other server-side program). These `GET` parameters expand to HTML tags (usually `<SCRIPT>` elements) that exploit known browser bugs. So, an attacker could embed the code in a URL that refers to your site and distribute only the URL, not the malicious Web page itself. That way, the attacker can remain undiscovered more easily and can also exploit trusted relationships to make users think the scripts are coming from a trusted source (your organization). For more details on this issue, see http://www.cert.org/advisories/CA-2000-02.html and http://www.microsoft.com/technet/security/topics/ExSumCS.asp.

Code for Filtering

Replacing `<`, `>`, `"`, and `&` in strings is a simple matter, and a number of different approaches can accomplish the task. However, it is important to remember that Java strings are immutable (i.e., can't be modified), so repeated string concatenation involves copying and then discarding many string segments. For example, consider the following two lines:

```
String s1 = "Hello";
String s2 = s1 + " World";
```

Since `s1` cannot be modified, the second line makes a copy of `s1` and appends `"World"` to the copy, then the copy is discarded. To avoid the expense of generating and copying these temporary objects, whenever you perform repeated concatenation within a loop, you should use a mutable data structure; and `StringBuffer` is the natural choice.

Core Approach

If you do string concatenation from within a loop, use `StringBuffer`, not `String`.

Listing 4.7 shows a static `filter` method that uses a `StringBuffer` to efficiently copy characters from an input string to a filtered version, replacing the four special characters along the way.

Listing 4.7 ServletUtilities.java (Excerpt)

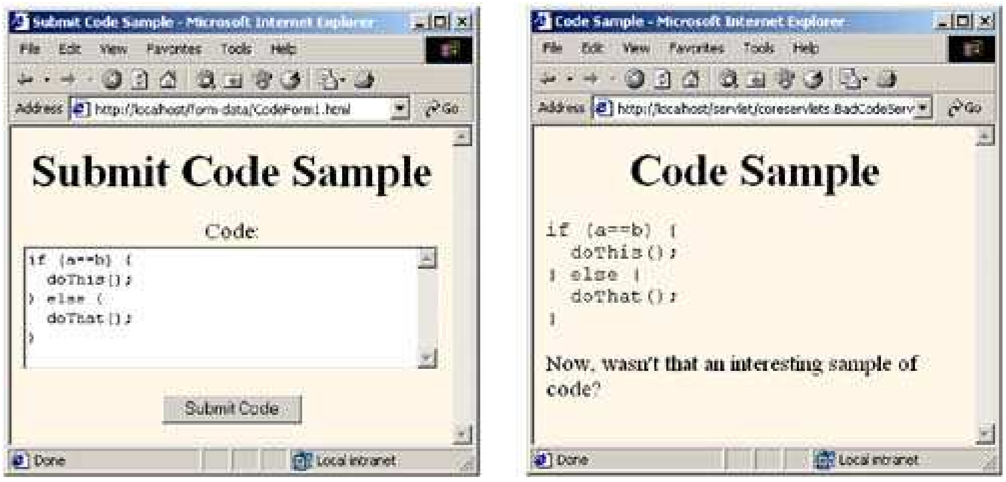
```
package coreservlets;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletUtilities {
  ...

  /** Replaces characters that have special HTML meanings
   *  with their corresponding HTML character entities.
   */
  // Note that Javadoc is not used for the more detailed
  // documentation due to the difficulty of making the
  // special chars readable in both plain text and HTML.
  //
  //  Given a string, this method replaces all occurrences of
  //  '<' with '&lt;', all occurrences of '>' with
  //  '&gt;', and (to handle cases that occur inside attribute
  //  values), all occurrences of double quotes with
  //  '&quot;' and all occurrences of '&' with '&amp;'.
  //  Without such filtering, an arbitrary string
  //  could not safely be inserted in a Web page.

  public static String filter(String input) {
    if (!hasSpecialChars(input)) {
      return(input);
    }
    StringBuffer filtered = new StringBuffer(input.length());
    char c;
    for(int i=0; i<input.length(); i++) {
      c = input.charAt(i);
      switch(c) {
        case '<': filtered.append("&lt;"); break;
        case '>': filtered.append("&gt;"); break;
        case '"': filtered.append("&quot;"); break;
        case '&': filtered.append("&amp;"); break;
        default: filtered.append(c);
      }
    }
    return(filtered.toString());
  }

  private static boolean hasSpecialChars(String input) {
    boolean flag = false;
    if ((input != null) && (input.length() > 0)) {
      char c;
      for(int i=0; i<input.length(); i++) {
        c = input.charAt(i);
        switch(c) {
          case '<': flag = true; break;
          case '>': flag = true; break;
          case '"': flag = true; break;
          case '&': flag = true; break;
        }
      }
    }
    return(flag);
  }
}
```

Example: A Servlet That Displays Code Snippets

As an example, consider the HTML form of Listing 4.8 that gathers a snippet of the Java programming language and sends it to the servlet of Listing 4.9 for display.

Now, when the user enters normal input, the result is fine, as illustrated by Figure 4-9. However, as shown in Figure 4-10, the result can be unpredictable when the input contains special characters like `<` and `>`. Different browsers can give different results since the HTML specification doesn't say what to do in this case, but most browsers think that the "`<b`" in "`if (a<b) {`" starts an HTML tag. But, since the characters after the `b` are unrecognized, browsers ignore them until the next `>`, which is at the end of the `</PRE>` tag. Thus, not only does most of the code snippet disappear, but the browser does not interpret the `</PRE>` tag, so the text after the code snippet is improperly formatted, with a fixed-width font and line wrapping disabled.

Figure 4-9. `BadCodeServlet`: result is fine when request parameters contain no special characters.


Submit Code Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/form-data/CodeForm1.html
Go
Submit Code
Done
Local intranet
Code Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.BadCodeServ
Go
Done
Local intranet

Submit Code Sample

Code:

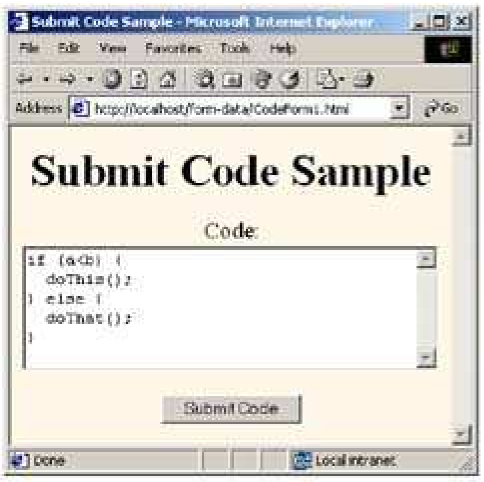
```
if (a==b) {
  doThis();
} else {
  doThat();
}
```

Code Sample

```
if (a==b) {
  doThis();
} else {
  doThat();
}
```

Now, wasn't that an interesting sample of code?

Figure 4-10. `BadCodeServlet`: result has missing and incorrectly formatted sections when request parameters contain special characters.

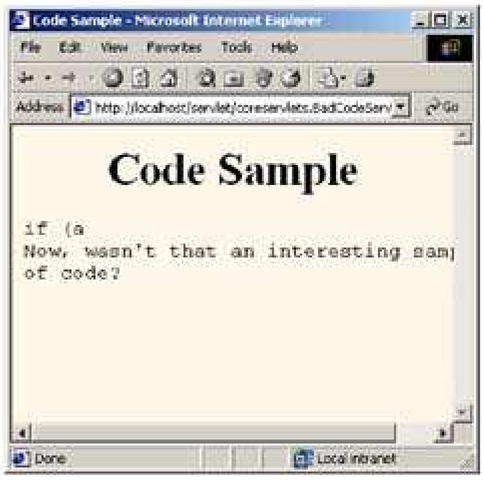

Submit Code Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/form-data/CodeForm1.html
Go
Done
Local intranet

Submit Code Sample

Code:

```
if (a<b) {
  doThis();
} else {
  doThat();
}
```

Submit Code


Code Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.BadCodeServ
Go
Done
Local intranet

Code Sample

```
if (a
Now, wasn't that an interesting sam
of code?
```

Listing 4.10 shows a servlet that works exactly like the previous one except that it filters the special characters from the request parameter value before displaying it. Listing 4.11 shows an HTML form that sends data to it (except for the ACTION URL, this form is identical to that shown in Listing 4.8). Figure 4-11 shows the result of the input that failed for the previous servlet: no problem here.

Listing 4.8 CodeForm1.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Submit Code Sample</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1 ALIGN="CENTER">Submit Code Sample</H1>
<FORM ACTION="/servlet/coreservlets.BadCodeServlet">
  Code:<BR>
  <TEXTAREA ROWS="6" COLS="40" NAME="code"></TEXTAREA><P>
  <INPUT TYPE="SUBMIT" VALUE="Submit Code">
</FORM>
</CENTER></BODY></HTML>
```

Listing 4.9 BadCodeServlet.java

```
package coreservlets;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

/** Servlet that reads a code snippet from the request
 *  and displays it inside a PRE tag. Fails to filter
 *  the special HTML characters.
 */

public class BadCodeServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Code Sample";
    String docType =
      "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 " +
      "Transitional//EN\">\n";
    out.println(docType +
                "<HTML>\n" +
                "<HEAD><TITLE>" + title + "</TITLE></HEAD>\n" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=\"CENTER\">" + title + "</H1>\n" +
                "<PRE>\n" +
                getCode(request) +
                "</PRE>\n" +
                "Now, wasn't that an interesting sample\n" +
                "of code?\n" +
                "</BODY></HTML>");
  }

  protected String getCode(HttpServletRequest request) {
    return(request.getParameter("code"));
  }
}
```

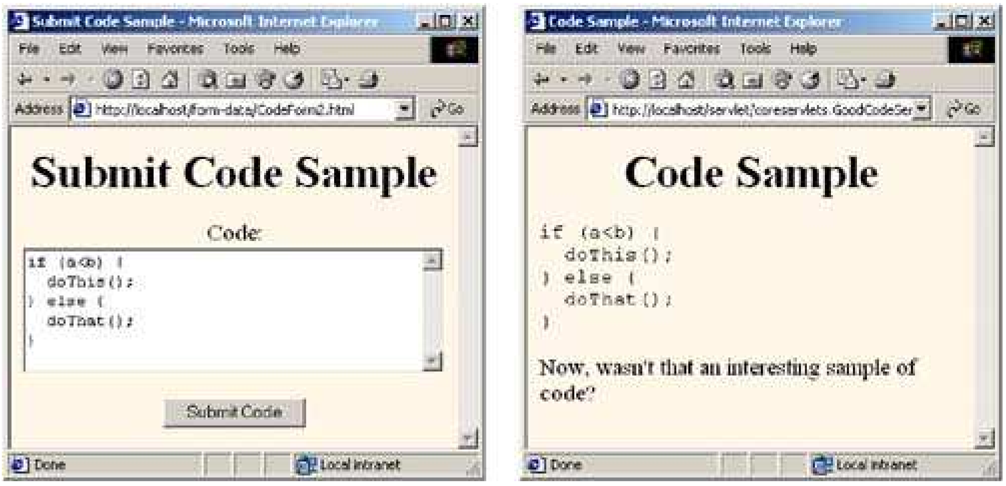
Listing 4.10 GoodCodeServlet.java

```
package coreservlets;

import javax.servlet.http.*;

/** Servlet that reads a code snippet from the request and displays
 *  it inside a PRE tag. Filters the special HTML characters.
 */

public class GoodCodeServlet extends BadCodeServlet {
  protected String getCode(HttpServletRequest request) {
    return(ServletUtilities.filter(super.getCode(request)));
  }
}
```

Listing 4.11 CodeForm2.html

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML><HEAD><TITLE>Submit Code Sample</TITLE></HEAD>
<BODY BGCOLOR="#FDF5E6">
<CENTER>
<H1 ALIGN="CENTER">Submit Code Sample</H1>
<FORM ACTION="/servlet/coreservlets.GoodCodeServlet">
  Code:<BR>
  <TEXTAREA ROWS="6" COLS="40" NAME="code"></TEXTAREA><P>
  <INPUT TYPE="SUBMIT" VALUE="Submit Code">
</FORM>
</CENTER></BODY></HTML>
```

Figure 4-11. `GoodCodeServlet`: result is fine even when request parameters contain special characters.


Submit Code Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/form-data/CodeForm2.html
Go
Done
Local intranet
Code Sample - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost/servlet/coreservlets.GoodCodeSer
Go
Done
Local intranet

Submit Code Sample

Code:

```
if (a<b) {
  doThis();
} else {
  doThat();
}
```

Submit Code

Code Sample

```
if (a<b) {
  doThis();
} else {
  doThat();
}
```

Now, wasn't that an interesting sample of code?