Java Servlets versus CGI
Implications for Remote Data Analysis

Stefan Kuhlins and Axel Korthaus

University of Mannheim
Department of Information Systems III
D–68131 Mannheim
Germany
http://www.wifo.uni-mannheim.de/

Abstract. The *Common Gateway Interface* (CGI) was the first attempt to enable the creation of dynamic HTML pages which represent a very suitable concept to meet the requirements of web-based applications for remote data analysis (RDA). CGI scripts are still popular, but by now there are new approaches which should be able to solve the main CGI problems. In this paper, we present the most promising one: *Java Servlets*. We will discuss the advantages and drawbacks of Java Servlets compared to CGI scripts. Moreover, we will do some performance measurements on the basis of simple classification problems and introduce the key functions of the *Java Servlet API*.

1 Motivation

Remote data analysis could be based on a powerful server which stores large amounts of data and makes varied analyses possible. An example of a remote cluster analysis using Java's *Remote Method Invocation* (RMI) for the communication between client and server is described in [2]. On the client side, a Java applet was employed to get input data and present analysis results. Since, due to differing Java versions, commonly used browsers still cause problems running applets, many web programmers do not want to use applets these days. Instead, they take conventional CGI scripts associated with HTML forms.

In a typical scenario (see figure 1), a client loads an HTML form, a user fills out the fields of the form, and the contents are then sent to a server. The server launches a program that reads the input and produces output which the server sends back to the client. The program has access to resources on the server. Therefore, it can perform a data analysis, for example.

The invocation of a program on the server in the scenario mentioned above is typical of an application of the *Common Gateway Interface* (CGI) standard. As we will see later on, things are a little bit different for Java servlets.

⋆ Accepted for 23^{rd} Annual Conference of the Gesellschaft für Klassifikation e. V., 10–12. March 1999 in Bielefeld

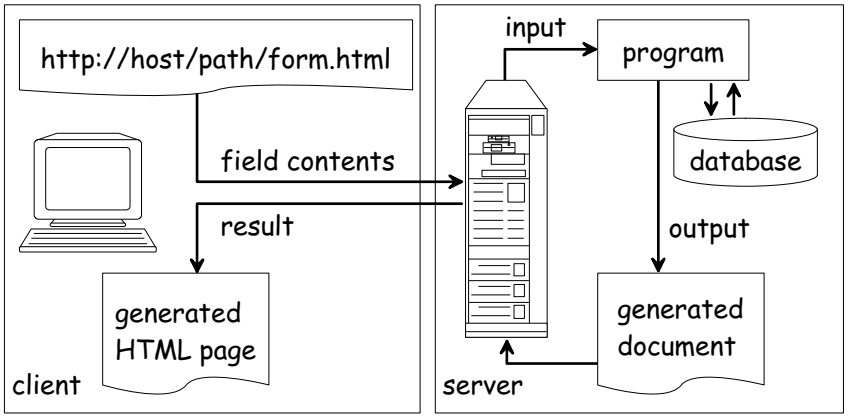

Fig. 1. Processing of HTML forms

2 The Common Gateway Interface

The *Common Gateway Interface* (CGI) is a simple and portable interface for running external programs (*CGI scripts*) on a web server. It depends on the system in which way a CGI script is executed, but usually a server creates a child process for it.

A CGI script reads its input from the standard input stream and as environment variables. Output of a script, e.g. an HTML page or a graphic, is written to the standard output stream. We will explain this with the help of a simple HTML form for cluster analysis.

```
<html>
<head><title>Cluster Analysis</title></head>
<body>
<form action="/cgi-bin/clusteranalysis.pl" method="GET">
    How many clusters?
    <input type=text name="clusters" size=3>
    <input type=submit value=" OK ">
</form>
</body>
</html>
```

Figure 2 shows how a web browser displays this form.

After pressing the `OK` button, the CGI script named in the action attribute of the form tag is called. In the example, the request method is `GET`. Hence, the most important environment variables are:

Most of the time, the request method `POST` is used instead of `GET` because in that case the form data is not part of the URL and no restrictions concerning the environment have to be considered.

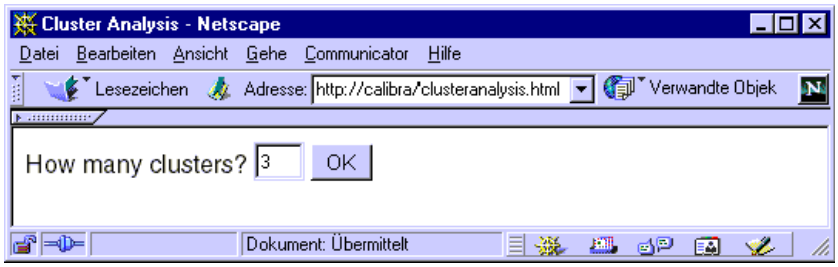

Fig. 2. Example of an HTML form

REQUEST_METHOD	GET
REQUEST_URI	/cgi-bin/clusteranalysis.pl?clusters=3
QUERY_STRING	clusters=3

Table 1. Important environment variables (GET)

Here, the CGI script reads the form data (`clusters=3`) from the standard input stream.

On principle, CGI scripts can be written in any programming language. One of the most common languages used for this purpose is Perl because it provides built-in support for text processing. In the following, we list a complete Perl script that parses the form input and generates an HTML page.

```
#!/usr/bin/perl

use CGI;
$q = new CGI;
$clusters = $q->param('clusters');
# ... compute results
print $q->header(-pragma => 'no-cache'),
    $q->start_html('Cluster Analysis Result'),
    $q->h1('Cluster Analysis Result'),
    "\nNumber of clusters: $clusters",
    "\n<!-- ... print results -->\n",
    $q->end_html;
```

There are several Perl libraries available for handling forms. In our example, `CGI.pm` is loaded. The resulting HTML page is shown in figure 3.

CGI scripts are suffering from certain drawbacks. Especially, the performance is low, because every invocation of a script leads to a new child process of the web server. They are potentially unsafe and there are many chances to make mistakes. Since a bad script can crash the whole server, many system administrators do not accept CGI scripts at all. Consequently, there is a need for a better technique. That's where Java servlets come into play.

REQUEST_METHOD	POST
REQUEST_URI	/cgi-bin/clusteranalysis.pl
CONTENT_LENGTH	10
CONTENT_TYPE	Application/x-www-form-urlencoded

Table 2. Important environment variables (POST)

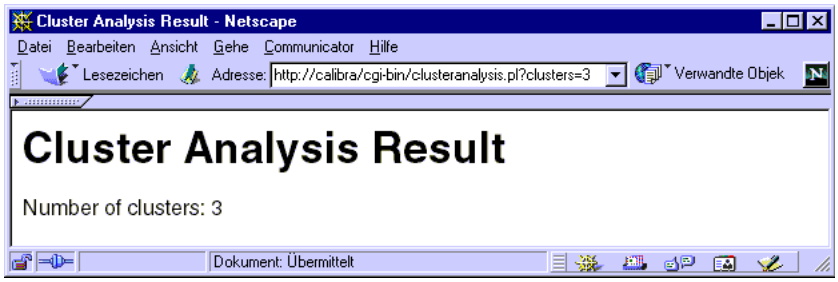

Fig. 3. Output of the CGI script

3 Java Servlets

Servlets are the server side counterparts of applets. They run in a special environment of the server which supports the servlet API. Servlets do not have a user interface but can access all server resources.

Servlets are easy to use and understand. Their performance is good, because once a servlet is loaded, the server only makes simple method calls and does not need to create a new process for each request.

At the time of writing this article, the current version of the Java servlet API is 2.1 ([6]). It defines the interface `javax.servlet.Servlet` and the two standard implementations `javax.servlet.GenericServlet` and `javax.servlet.http.HttpServlet`.

Figure 4 shows the lifecycle of a servlet. After creation and initialization a servlet is ready to answer requests. If the servlet is no longer needed, it has the chance to release resources that have been reserved during initialization. Finally it will be garbage collected.

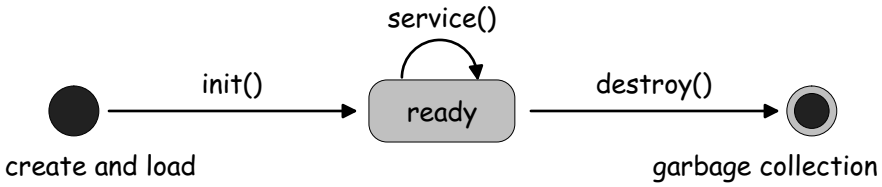

Fig. 4. The lifecycle of a servlet

To define a servlet the interface `javax.servlet.Servlet` has to be implemented. But instead of defining all methods of this interface it is easier to subclass one of the standard implementations `javax.servlet.GenericServlet` or `javax.servlet.http.HttpServlet`. In this case, only those methods have to be redeclared whose default implementations are not appropriate.

In the following, we will present some of the key methods:

- `void init(ServletConfig)`

This method is called only once for initialization while loading the servlet. The `ServletConfig` parameter contains information about the server environment and start-up parameters.

- `void destroy()`

This method is called when the server terminates the servlet and before garbage collection takes place.

- `String getServletInfo()`

This method provides information about the servlet, such as its purpose, author, copyright etc.

A HTTP servlet subclasses `javax.servlet.http.HttpServlet` with added HTTP-specific functionality. This superclass defines methods `doGet`, `doPost`, `doPut` etc. with sensible default behaviors for the corresponding HTTP methods like `GET`, `POST`, `PUT` etc. If `javax.servlet.GenericServlet` is subclassed instead, there is only the protocol-independent `service` method with two parameters:

- `void service(ServletRequest req, ServletResponse res)`

Often, the `doPost` or `doGet` methods have to be redeclared. Both have two parameters:

- ```
  void doGet(HttpServletRequest req,
                 HttpServletResponse res)
  ```
- ```
  void doPost(HttpServletRequest req,
                 HttpServletResponse res)
  ```

They must be able to handle multiple requests at the same time.

Input and output are possible as text or binary data (see table 3).

Two recommendable references concerning Java servlets are [6] and [1].

	Input	Output
Text	`req.getReader()`	`res.getWriter()`
Binary	`req.getInputStream()`	`res.getOutputStream()`

Table 3. Methods for input and output

3.1 Example

In our example form, only the value of the action attribute has to be changed in order to use a servlet instead of a CGI script. It depends on the configuration of the web server where servlets reside. Usually, this is the `/servlet/` path.

```
<html>
<head><title>Cluster Analysis</title></head>
<body>
<form action="/servlet/ClusterAnalysis" method="GET">
    How many clusters?
    <input type=text name="clusters" size=3>
    <input type=submit value=" OK ">
</form>
</body>
</html>
```

For the client it makes no difference whether the form uses a servlet or a CGI script. Therefore, the layout of the form is the same in both cases, see figure 2.

Now, we are able to define a simple HTTP servlet that processes the form input.

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

public class ClusterAnalysis extends HttpServlet {
   public String getServletInfo()
      { return "Cluster Analysis V 1.0"; }
   public void doGet(HttpServletRequest req,
         HttpServletResponse res)
         throws ServletException, IOException {
      String clusters[]
         = req.getParameterValues("clusters");
      // ... compute results
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<html><head>"
         + "<title>Cluster Analysis Result</title></head>"
         + "<body><h1>Cluster Analysis Result</h1>"
```

```
            + "Number of clusters: " + clusters[0]
              // ... print results
            + "</body></html>");
    }
}
```

Even those who have not seen a servlet before should understand what this servlet is doing. It produces an HTML page similar to the one in figure 3.

3.2 Possible Usage

As already shown, servlets can be used for processing HTML forms. They can dynamically generate HTML pages, e.g. presenting results of database queries or data analyses. Moreover, they are able to manage sessions which is important for successive requests. For example, based on the result of a cluster analysis it is possible to select one cluster for a subsequent analysis.

3.3 Saving State Information

Since a servlet is loaded once and can answer many requests it is possible to save state information between requests in its instance variables. For example, in order to perform a subsequent cluster analysis, it might be sufficient to specify the relative increment or decrement of the number of clusters.

This feature has to be used with care because a web server can load more than one instance of a servlet at the same time. Moreover, a web server has the right to remove a loaded servlet.

4 Performance

Two points are vital, if the performance of servlets and CGI scripts is to be compared. First, the time needed for a direct response, and second, the time needed for computing a result. For CGI scripts the latter depends on the programming language. Since we don't want to compare the performance of Java with Perl, C or C++, we will concentrate on the first point.

The time needed for a direct response can be measured with a really simple program like the famous *"hello, world"* whose computing time is nearly zero. For classic CGI scripts the loading time is the same for every response. For servlets, on the other hand, it depends on whether a servlet is already loaded by the web server or not. Usually, a web server loads a servlet when the first request takes place but it is also possible to load servlets directly after starting the server. Then, the servlet is already in memory for subsequent requests. Since this is typical of servlets requested frequently, we will compare the response times of a loaded servlet and a CGI script.

Our initial test environment consisted of the following components which are available for free:

- *Apache Web Server*, Version 1.3.4 ([3])
- *Apache JServ*, Version 1.0 ([4])
- *Java Development Kit*, JDK 1.1.7 ([5])
- *Java Servlet Development Kit*, JSDK 2.0 ([6])

Theoretically, performance of a servlet should be better than that of a CGI script. But with the above configuration, the CGI script was three times faster than the servlet. The reason for this strange result is that the Apache server itself is not implemented in Java and therefore needs to load a *Java Virtual Machine* for executing a servlet.

If Sun's *Java Web Server* (JWS 1.1.3, [7]) is used instead of Apache, servlets work as expected. Here the servlet runs four times faster than the CGI script with Apache (see figure 5). To be fair it should be mentioned that some new techniques like *"Fast CGI"* exist to speed up CGI scripts.

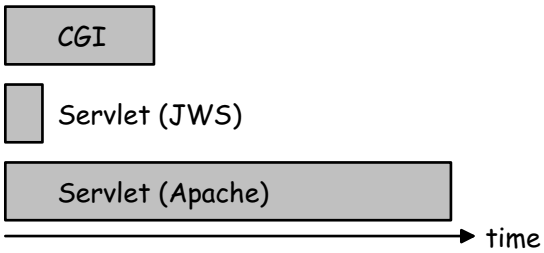

Fig. 5. Comparison of direct response times

5 Summary

Since servlets are implemented in Java, they inherit the features of this programming language, i.e. they are safe, simple, portable, and run on all Java compatible systems. They can do everything CGI scripts can do.

Table 4 shows a comparison of Java servlets and CGI scripts on the basis of five critical points.

	Java servlet	CGI script
Programming language	Java	any
Execution	method call	external program
Parameters	`ServletRequest`	environment variables
Run time checks	Java Virtual Machine	language specific
Security	Java Security Manager	system specific

Table 4. Java servlets compared to CGI scripts

Most internet providers do not permit their clients to install their own CGI scripts because of security risks. Unfortunately, many providers do not support Java servlets yet. But we are sure that this will change in the future.

Based on Java servlets there are some other useful techniques like *Java Server Side Includes* (JSSI) and *Java Server Pages* (JSP). In both cases it is possible to place the output of a servlet or the result of a Java statement with special tags (`<SERVLET ...>` or `<% ... %>`, respectively) in HTML pages. Combined with Java applets on the client side, powerful solutions for client/server communication are possible. An example was mentioned in the motivation section.

Putting all Java-based solutions together it is possible to do the whole programming on server and client side in only one single language, namely Java. It is no longer necessary to use specialized languages for the different tasks.

However, solutions based on Java applets on the client imply several compatibility problems concerning different Java versions supported by the commonly used web browsers. By contrast, purely server-based Java solutions provide the advantage that the client only receives documents like HTML pages and does not have to be concerned with Java at all.

To sum up, Java servlets have the potential to substitute CGI scripts. They make elegant solutions for many problems including remote data analysis possible.

References

1. HUNTER, J. (1998): Java Servlet Programming, O'Reilly & Associates Inc., Sebastopol.
2. KUHLINS, S., SCHADER, M. (1999): Remote Data Analysis Using Java, in: Studies in Classification, Data Analysis, and Knowledge Organization, Gaul, W., Locarek-Junge, H. (Eds.), Classification in the Information Age, Springer-Verlag, June 1999, pp. 359–367.
3. APACHE GROUP (1999): Apache Web Server, http://www.apache.org/.
4. JAVA APACHE PROJECT (1999): Apache JServ, http://java.apache.org/jserv/index.html.
5. SUN MICROSYSTEMS INC. (1999a): Java Development Kit, http://java.sun.com/products/jdk/.
6. SUN MICROSYSTEMS INC. (1999b): Java Servlet Development Kit, http://www.javasoft.com/products/servlet/index.html.
7. SUN MICROSYSTEMS INC. (1999c): Java Web Server, http://www.sun.com/software/jwebserver/index.html.