|  |
| --- |
| http://www2.sys-con.com/itsg/virtualcd/java/images/top.jpg |
|

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| http://www2.sys-con.com/itsg/virtualcd/java/images/side.jpg | |  |  |  | | --- | --- | --- | |  | http://www2.sys-con.com/itsg/virtualcd/java/archives/0505/williamson2/title.gif  Java has brought much in the way of programming advancement within easy reach of developers. Powerful constructs such as multithreading and advance communication are now relatively easy for you to use in your programs. Java has also brought a strong sense of object orientation. The ability to reuse code and not reinvent the wheel each time you program is a great improvement over alternative programming languages.  But for a developer to reuse code, the language has to support the mechanisms that make it easy to build complex systems by simply plugging objects together. Java has these mechanisms - the servlet API is no stranger to object reuse. This article outlines the different methods by which servlets can communicate data to one another. This communication promotes the concept of solving a problem only once and then applying the solution to many different areas.  As this article will illustrate, you, as a servlet developer, have a lot of choices on how you can best reuse your servlet classes.  **Common Base Class**  A servlet is run in response to a client connection. It is designed to be lightweight and to process the client request as quickly as possible, thus freeing up the server so it can immediately process the next request that comes in. A servlet should never try to do everything itself. In a large system it is better to produce many smaller servlets, as opposed to a small number of large servlets. The longer a servlet spends servicing a request, the fewer clients the servlet can process in a given time.  If smaller servlets are to be built, rather than using the one-servlet-solution-fits-all approach, the ability to share information between the servlets is paramount.  One of the easiest ways for a servlet to share information (this works for classes as well) is to share common objects through inheritance - where the servlets are extended from a class other than HttpServlet or GenericServlet. To illustrate this concept, consider the example shown in Figure 1.  Figure 1 **Figure 1:**  Suppose we had two servlets in our system: one to log a user onto our system and the other to log him. Let's assume that the reason we want a user to log on and log off again is to enable us to connect to a specific database given the username and password. Now, let us assume that they will share various methods, such as a simple check to see if the user is already logged on. We can develop methods for this simple check and place a copy of them in each servlet. However, since it's a database connection, let's also assume that we will be sharing database connections between the servlets.  Many different methods are available to facilitate method sharing and data pooling. For this section, we will use the simplest method. Instead of each servlet extending the HttpServlet, each will extend the capabilities of ServletBase, which is a custom class we will build. This class will itself extend the HttpServlet class, but it will also add other methods to allow the servlets to intelligently handle the database connections.  Listing 1 shows the basic skeleton implementation for the ServletBase class. You'll notice that it looks very similar to a normal servlet, except it doesn't override the service(...) method or any of the doXXX(...) methods. It does, however, override the init(...) method. In this instance it needs to open up a connection to the database which will be shared through the static Connection variable. This will be performed when the servlet first initializes.  However, please note a very important fact. The init(...) method will be called multiple times if multiple servlets inherit this class because each particular servlet needs to be initialized and needs to go through the complete servlet life cycle. Since we don't want to have additional database connections open, we'll check to see if it has been opened or not, and if so, we skip over that section.  The two additional methods, isLoggedOn(...) and logUserOn(...), are support functions that are accessible by all servlets that extend this class.  So now that we have developed the new base class, let's look at how we can use it. Listing 2 shows the logonServlet that will log a user onto the system. This class, instead of extending the HttpServlet class, extends the ServletBase class. Notice that this class does not override the init(...) method because the base class is taking care of this for us.  The API doesn't require the init(...) method to be overridden like the previous versions insisted it should be, so it may be left out without any adverse effects.  The logonServlet can now call methods from the underlying base class without having to wrestle with any object references. This allows class to easily share common methods and data without keeping a reference to a collection class.  This ability is not hard, and it's not new. It is used throughout Java to create groups of logical units, and it can easily be used in the world of servlets, where the need to share data is even more important.  **Sharing Data**  If the previous method seems a little clunky and too much of a hassle, the 2.1 version of the servlet API introduces a way to share objects among servlets. It operates along the same lines as the session management functionality, but it is done completely on the server side.  In the session management, you associated an object with a given client session. The session manager attempted to return the same object to you each time that particular client made a request. This was done using cookies or URL rewriting. Objects were stored using a unique string to reference them.  Servlets can now benefit from this technology through the ServletContext interface. Each servlet runs within some sort of environment. This environment describes various parameters associated with the servlet environment, such as the document root and any mappings that may exist. This is known as the ServletContext. A servlet belongs to only one ServletContext and the server's Web administrator controls this.  For example, a server running virtual hosts would have a different ServletContext for each virtual host. This has the advantage of logically grouping servlets while maintaining different security models for each type.  The ServletContext can store objects on behalf of servlets running within its context. There are methods that allow a servlet to add and remove objects and to determine what the objects are that are stored. These objects can be retrieved and modified by other servlets running within the context.  Let's now look at how we could share a database connection among servlets using the ServletContext mechanisms. Listing 3 shows a simple example where we create a new database connection from some method and ask the ServletContext to store it for us under the name of "database.connection" using the setAttribute(...) method.  Retrieving the database connection again is just as easy as setting it. Listing 4 shows another servlet, running within the same context, which attempts to retrieve the object back.  If the object has not been stored, the getAttribute(...) method will return null. As you can see, the whole storage and retrieval system operates very much like the cookie system - so this can be thought of as server-side cookies!  Listing 5 illustrates another useful method from the ServletContext interface. The ability to discover which objects are in existence is a very handy feature; for example, it enables you to discover whether any initialization has been performed. The getAttributeNames() method returns an Enumeration to the list of stored objects, which can then be returned using the getAttribute() method.  We have now seen methods for storing and retrieval of data. One additional method allows you to remove a given object from the ServletContext. This method, removeAttribute(...), ensures that the ServletContext no longer holds a reference to the object.  So the ability to share data among servlets running within the same context is straightforward enough. But what if the servlet isn't running within the same context? What if a servlet running on another machine or another virtual host has data that the current servlet needs? Have no fear; help is at hand.  The 2.1 servlet API introduces a new interface that allows servlets to get an external reference to a context. This is done through an overloaded version of getServletContext(...) which takes in the URL of the servlet you wish to retrieve the context for. An example of this call is shown in Listing 6.  Assuming the remote host allows this to be performed, the ServletContext will be returned. However, if it is not, then the method will return null. In calling this method, it will not invoke the servlet running in the other context, nor will it interpret any processing that that remote servlet may be doing at that point.  Using this new mechanism from the servlet API, you can easily share objects with other developers, including other contexts.  **Servlets Working Together**  In the previous section, we looked at a number of ways to share objects between servlets. In most instances this is used to share common data resources such as a database connection. This can allow for more efficient operation within the server. Now, thanks to the new addition to the servlet API, servlets can work together in yet another way.  In past version releases, a technique known as servlet chaining was possible. In servlet chaining, the output of one servlet fed the input to another servlet. The second servlet, or the next one in the chain, could perform additional processing before sending the result out to the client. Although this was useful, it was a bit clunky to operate. To begin with, the chaining mechanism had to be set up by the Web administrator, and it could not be performed dynamically. Also, the servlet had no control over whether it should be in a chain. From the developer's point of view, no API control was possible.  The new API has made this feature much more accessible. You have the ability to insert the contents of one servlet within the output of another, and you can pass a request on to another servlet for processing.  The following sections will show you how you can use these features in your servlets.  **Forwarding a Request**  You may find a time when it's better to pass a request on to another servlet for processing. For example, consider a servlet that acts as a front-end processor for a search engine. Depending on the result of the query, different servlets could be used to generate the actual output. The first servlet would take in all the search parameters, run the query, and pass the results on to a specific servlet for output. The advantage over the conventional servlet-chaining model is that the servlet can decide at runtime which output servlet will be used as opposed to relying on a Web administrator to set up the mapping. This is particularly useful in an ISP environment where the developer has to rely on a third party to get this right.  To handle this scenario and the one in the next section, a new interface was developed. The RequestDispatcher interface is the glue that controls the flow between servlets and output streams such as HTTP clients, files, or even JavaServer Pages (JSP).  Before you can send control to another servlet, you must first get the necessary RequestDispatcher for that particular servlet. This is controlled through the method call getRequestDispatcher(...) from the ServletContext interface. You specify the URI to the servlet, and assuming no security violations incur, the necessary RequestDispatcher reference is returned.  From here, you can make a call to the forward(...) method, which passes on the request to the servlet represented by theRequestDispatcher. Let's look at an example of this in action.  Listing 7 shows a typical use of this mechanism. Assume that the forwardServlet shown takes in a search query from the user. It first retrieves an instance to the context it is running within, then using this reference a call to retrieve the RequestDispatcher for the xyzServlet that will be used to generate our output for us.  Before we pass on the request for processing, we run the query. The next servlet will be used to process the results for this query, so we need some mechanism to pass the results to the next servlet.  As demonstrated in the previous section, sharing objects between servlets is not that big a problem; we simply ask the ServletContext to store the data for us. But this is not satisfactory, as these objects take no account of the client request. Storing the results this way would be very dangerous since only one copy per ServletContext would exist, as opposed to one copy per client request.  You could use session management to get around this problem, but since the data hasn't actually made it back to the client, it's not an ideal solution.  Thanks to the ServletRequest object, help is available. There are methods that allow servlets to ask that the ServletRequest object store objects for it on its behalf, ensuring that the object remains intact per client session. The first servlet can then pass data to the second servlet without worrying about multiple-client sessions.  As you'll see in Listing 7, the method for storing data is exactly the same as was demonstrated for storing objects using the ServletContext. The equivalent getAttribute(...) methods exist to allow the servlet to retrieve the objects again.  When you are forwarding a request to another servlet for processing, the first servlet is not permitted to do anything to the output, including setting any HTTP status fields. As soon as the first servlet makes an attempt to retrieve an output stream, then the call to forward(...) will fail. The calling servlet is permitted to continue running after the call, but it must not make any attempt to communicate back to the original client.  **Inserting a Request**  The previous section looked at when the complete request is passed on to another servlet for processing. Although this procedure is very useful, it is restrictive - only one servlet can produce the final output. There are times where it would be nice if servlets could cooperatively work together to create the output. For example, you could have one servlet conduct or control the flow calling other servlets to insert content as and when.  Fortunately the API supports this very type of use. The previous section introduced the forward(...) method from the RequestDispatcher interface. To insert content into the current output stream, the insert(...) method is available. This method operates in exactly the same way as forward(...) in that it accepts a ServletRequest and ServletResponse as parameters. Only this time, the called servlet is not meant to set any headers. If it does (for example, any cookie setting), then it is not guaranteed to be successful. As before, data can be passed to the servlet using the ServletRequest interface that was shown in the previous section.  Listing 8 shows an example of the insert(...) method. As before, it retrieves a reference to the RequestDispatcher of the servlet we wish to use. We create some output from the original servlet and then call the secondary servlet 10 times so it can insert its output to the original client request.  There is no limit to the number of times or number of servlets you can ask to insert output for. Using this technique, you have the ability to create sophisticated servlet systems that can all work together closely, thus reducing the total number of servlets that the server must develop and handle.  **Summary**  This article introduced the features that are available to you for sharing data between servlets. This functionality, as well as the ability to have servlets share the responsibility of processing the client request by collaborating on the output generation, was a major step forward in servlet ideology.  [alan@sys-con.com](mailto:alan@sys-con.com)  http://www2.sys-con.com/itsg/virtualcd/java/images/SourceCode.jpg    **Listing 1: ServletBase**    public class ServletBase extends HttpServlet{  static Connection databaseConnection = null;  public void init(ServletConfig \_config) throws ServletException{  super.init(\_config);  if ( databaseConnection == null )  //- Open up the database connection  }  protected boolean isLoggedOn( String \_username ){  return true;  }  protected boolean logUserOn( String \_username ){  return true;  }  }    **Listing 2: Using the NewSerletBase Class**    public class logonServlet extends ServletBase{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  if ( isLoggedOn( \_req.getParameter("USERNAME") ){  //- Display a message indicating they are already logged on  }else{  logUserOn( \_req.getParameter("USERNAME") );  }  }  }    **Listing 3: Storing an Object**    public class logonServlet extends HttpServlet{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  ServletContext thisContext = getServletContext();  //-- Assume some method creates a new connection class  Connection newConnection = createConnection();  thisContext.setAttribute( "database.connection", newConnection );  //-- Return some output to the client  }  }    **Listing 4: retrieving an Object**    public class logoffServlet extends HttpServlet{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  ServletContext thisContext = getServletContext();  //-- Assume some method creates a new connection class  Connection newConnection = thisContext.getAttribute(  "database.connection");  if ( newConnection == null )  //- Database has not been opened yet  //-- Return some output to the client  }  }    **Listing 5: Looking at All the Objects**    public class allServlet extends HttpServlet{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  ServletContext thisContext = getServletContext();  //-- Assume some method creates a new Connection class  Enumeration E = thisContext.getAttributeNames();  while ( E.hasMoreElements() ){  String name = (String)E.nextElement();  System.out.println( "Object: " + name );  }  }  }    **Listing 6: Retrieving Remote Contexts**    public class otherServlet extends HttpServlet{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  ServletContext otherContext = getServletContext  ("http://<otherdomain>/servlet/allServlet");  //-- Assume some method creates a new Connection class  Enumeration E = otherContext.getAttributeNames();  while ( E.hasMoreElements() ){  String name = (String)E.nextElement();  System.out.println( "Object: " + name );  }  }  }    **Listing 7: Forwarding a Request**    public class forwardServlet extends HttpServlet{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  ServletContext xt = getServletContext();  RequestDispatcher xyzServlet = xt.getRequestDispatcher  ("http://<domain>/servlet/xyzServlet");  //- Do any preliminary processing  \_req.setAttribute( "database.results", new Results() );  xyzServlet.forward( \_req, \_res );  }  }    **Listing 8: Inserting Content**    public class insertServlet extends HttpServlet{  public void service(HttpServletRequest \_req, HttpServletRe-  sponse \_res) throws ServletException{  ServletContext xt = getServletContext();  RequestDispatcher xyzServlet = xt.getRequestDispatcher  ("http://<domain>/servlet/xyzServlet");  PrintWriter Out = \_res.getWriter();  Out.println( "This is from the insertServlet " );  for(int x=0; x < 10; x++ )  xyzServlet.insert( \_req, \_res );  Out.println( "This is the end of the print servlet " );  }  } |  |   All Rights Reserved  Copyright ©  2004 SYS-CON Media, Inc.    E-mail: [info@sys-con.com](mailto:info@sys-con.com)  Java and Java-based marks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. SYS-CON Publications, Inc. is independent of Sun Microsystems, Inc. |