TeaServlet Architecture

July 12, 2000

Please send all comments to: opensource@dig.com

DOCUMENT REVISION AND EVOLUTION

Primary Author(s)	Description of Version	Date Completed
Michael Rathjen	Initial public release, based on interviews with Brian O'Neill.	June 30, 2000
Michael Rathjen	Minor update.	July 12, 2000

PREFACE

This document provides an overview of the TeaServlet Architecture for Java developers. It is not intended to be a user manual for TeaServlet.

There are a number of other documents complementary to this reference that cover topics associated with Tea and Tea-related tools. For additional Tea-related documentation, please visit http://opensource.go.com/.

CONTENTS

1. INTRODUCTION

The Tea Servlet architecture is a complete template environment that delivers web pages via the servlet API. Tea templates cannot directly acquire or modify information, but rely instead on their host to provide data returned from called functions. This is where the Tea Servlet plays an important part.

The role of the TeaServlet is to execute the Tea templates and their scripted data into a web page by taking the scripted data, executing Java internally, and sending the web page to an output buffer. The TeaServlet performs this conversion each time a page is requested, which provides the ability to serve dynamic content through defined templates.

2. FUNCTIONAL RELATIONSHIPS

In order to be more useful than standard HTML, Tea templates call upon special functions that are managed in contexts and returned by applications. The TeaServlet needs to support multiple contexts, so a merged context and a merged application are created. This saves the developer from having to merge the contexts manually beforehand.

2.1 Applications and Contexts in TeaServlet

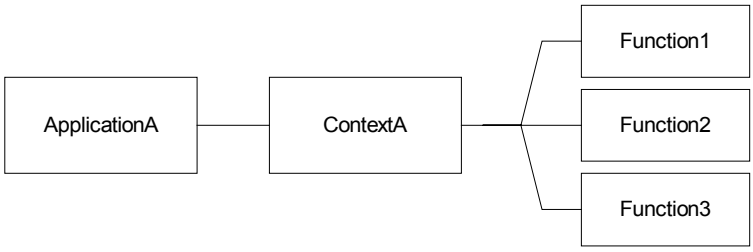

Figure 1 - The Role of a Context

Figure 1 illustrates the relationship between functions and an application. An application serves to return a context (a collection of functions) to a Tea template call, where each application can provide only one context. In a similar fashion, Tea templates can accept only one context. However, the TeaServlet can manage multiple contexts by merging the contexts and applications, as shown in Figure 2.

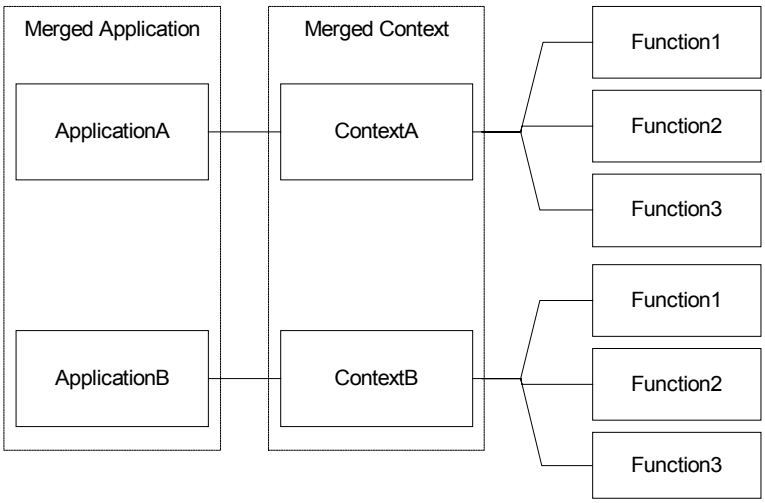

Figure 2 - The Relationship between Applications and the Merged Application

When the merged application is called to provide a context type, it gathers contexts from each individual application. The functions of each context are contained in the merged class. The merged application takes each context and passes it to the instance of the merged context class. This generated class is a wrapper; it delegates function calls to the appropriate applications. If any included context implements interfaces or is an interface, the merged class implements the interfaces.

2.2 Function Priority in the Merged Context

One or more contexts may have the same function, or different functions may have the same name. When contexts are merged containing conflicting function names, the first function added to the merged context will have preference. Any additional functions with the same name will not be added to the merged context using that name. Those functions can still be called by using a more complete name. An example follows:

Two applications, **appA** and **appB**, are contained in a merged application and they both have functions named **foo**. If function **foo** is called from a template, then the actual **foo** called is from **appA**, not **appB**. Either **foo** can be directly called by using complete addressing, such as **appA.foo** and **appB.foo**. Note that this order preference is not alphabetical. If **appA** and **appB** were reversed in the configuration order, function **foo** would be **appB.foo**.

There is also a default context built into the TeaServlet that provides a number of default functions. The functions in this context have the highest order preference and cannot be overridden with user-supplied contexts or their corresponding applications. The default context is not associated with any application.

3. TEASERVLET PROCESS FLOWS

3.1 Sequence of Initialization

When the TeaServlet is first started, it performs a series of tasks, including initializing applications, merging contexts, and loading and compiling templates.

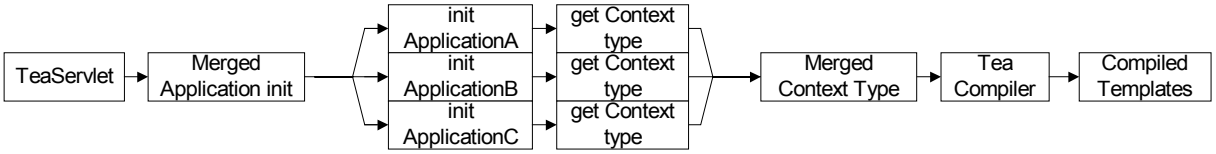

Figure 3 - TeaServlet Sequence of Initialization

3.1.1 Merged Application

The TeaServlet begins by creating an empty merged application based upon an initial set of parameters. The TeaServlet then calls an init method on the merged application, resulting in the loading of each application. The merged application subsets the initial set of parameters to ensure that each individual application receives the correct parameters.

Each application's init method is called, and the application is queried for what context it provides. If an application provides no context, it returns null to the query and during execution the merged application will not ask the application to create a context.

After querying applications and gathering the context types they provide, the merged application creates an auto-generated merged context.

3.1.2 Compiled Templates

The TeaServlet finishes initialization when the merged context class passes the templates to the Tea compiler. By option, the resulting compiled files can be saved to disk. During future initializations, the TeaServlet compares the dates of the source file to the class file. The TeaServlet then recompiles those source files that are newer than their class files. If the TeaServlet encounters template errors, it still loads as many templates as it can in an attempt to get into a usable state.

To ensure that user-defined classes don't conflict with standard Java classes, a prefix is added during compilation. For example, if **foo.java** is compiled to **foo.class**, then **foo.tea** cannot also be compiled to **foo.class**. A package prefix is added to the class file, and it becomes **com.go.teaservlet.template.foo.class**.

3.2 Sequence of Execution

All Java servlets receive a request, process the request, and send a response. When the TeaServlet receives a servlet request, it performs a series of tasks that results in a web page sent

as a servlet response. This series of tasks restarts for *every hit*, allowing the content to change from one hit to the next.

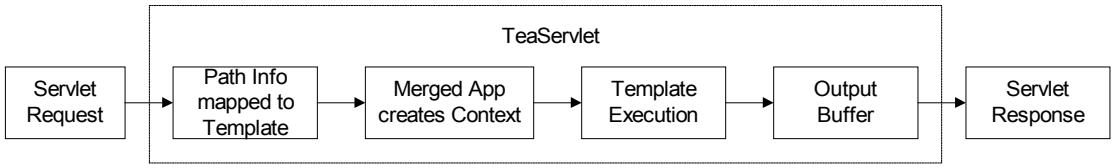

Figure 4 - TeaServlet Sequence of Execution

3.2.1 Servlet Request

The servlet engine sends a request to the TeaServlet in the form of a doGet or doPost method. The TeaServlet treats both methods identically. This feature lets templates be created as receivers of form-posted data.

3.2.2 Path Info Mapped to Template

The URI is processed by the servlet engine, where it is broken down into components such as the servlet path, path info, and query string. The TeaServlet then maps the path info to the appropriate Tea template as shown in Figure 5.

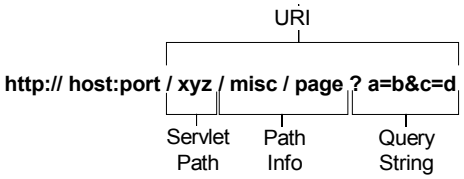

Figure 5 - URI Breakdown

Note that there is no **.tea** extension in the path info. The TeaServlet does not map the URI based upon file extensions. The path is translated to a fully qualified class name. In the example from Figure 5, the class name would be **com.go.teaservlet.template.misc.page**.

If parameters from the query string are requested, they are passed to the template. Although many search engine spiders will not index URLs that contain a query string, TeaServlet supports custom separators. This lets developers use custom characters for query string separators.

The following configuration example, placed into the properties file, lets the TeaServlet to process additional patterns for the query string separator characters '?', '=' and '&'.

```
separator.query = .html/
separator.value = -
separator.parameter = /
```

Using the example setup above, a request for **/path/directory?user=9999&show=address** can also be requested as **/path/directory.html/user-9999/show-address**.

3.2.3 Template Execution

If the TeaServlet finds the template based upon the URI breakdown, it creates an application request object and a response object. The request object determines which template to use and the response object is the output buffer. The TeaServlet then calls the merged application for the merged context.

Each context object is passed into the merged application, and the output buffer is created. The content from the executing template is stored in the output buffer until complete.

If a redirect is executed, the template processing stops and the browser is told where to go. If there is an error in the content, the buffer is discarded entirely and the servlet engine error handler becomes responsible for a new response. The servlet engine error handler can write standard responses for standard errors. If any exception occurs, it is captured, logged, and an "HTTP 500 Internal Server Error" is passed to the user. Template aborts are not logged as errors.

3.2.4 Output Buffer

The TeaServlet includes a buffer that holds data while the template executes. The executing template may pull data from several different sources. Figure 6 shows an example of a web page constructed from an output buffer.

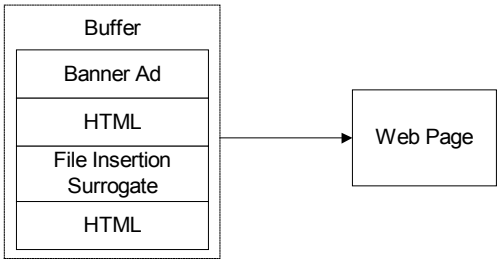

Figure 6 - TeaServlet Buffer

The output buffer is designed to provide several features, such as efficient capture of output, character conversion, and predetermined content length.

3.2.4.1 Output Capture

Unlike a string buffer, which doubles in size when it grows and must repetitively copy itself, the output buffer consists of a linked list of smaller buffers. The output buffer resizes itself as needed and captures the output stream efficiently.

3.2.4.2 Character Conversion

The output buffer provides character conversion from Java unicode to other character set encodings, such as ISO-8859-1 and UTF-8. Any constant strings are captures and the character conversion is cached, improving performance.

3.2.4.3 Predetermined Content Length

Having a predetermined content length allows the TeaServlet to use persistent connections, increasing transmission efficiency.

3.2.5 Servlet Response

If the template is executed without errors, it sets the content length (computed by the output buffer) and writes the buffer to the output stream, completing the servlet response. Once the servlet response is sent, the TeaServlet is finished for one hit.

3.3 Sequence of Destruction

After completing the hit, the TeaServlet calls the merged application's destroy method, which in turn calls each application's destroy method.

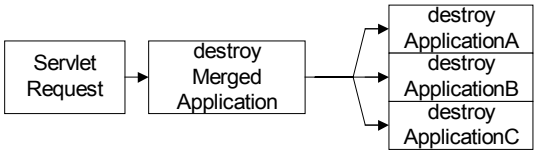

Figure 7 – Sequence of Destruction