URL to Script Resolution

This page explains how Sling maps URLs to a script or and servlet.

See also Servlets and Scripts which provides detailed info about how to register servlets.

First of all Sling looks up the resource identified by the URL - typically a path inside the JCR repository, which is annotated by the sling:resourceType property which defines the resource type of that resource. Using this resource type (which is kind of a relative path, eg. "myblog/comment"), scripts or servlets are looked up. For more details about how the initial resource is identified for a specific request URL look at URL decomposition.

Scripts and servlets are itself resources in Sling and thus have a resource path: this is either the location in the JCR repository, the resource type in a servlet component configuration or the "virtual" bundle resource path (if a script is provided inside a bundle without being installed into the JCR repository).

For the whole Truth about script resolution, see the ScriptSelectionTest class. If you see interesting cases that are not covered there, please let us know via the Sling users mailing list.

TODO: explain super types, servlet path mappings, node type resource types (my:type -> my/type)

Fundamental: Scripts and Servlets are equal

In the following discussion, I will write about scripts. This will always include servlets as well. In fact, internally, Sling only handles with Servlets, whereas scripts are packed inside a Servlet wrapping and representing the script.

Base: Resource Type Inheritance

While not exactly part of our discussion, resource type inheritance as implemented for SLING-278 plays a vital role in script resolution.

Each resource type may have a resource super type, which may be defined in various ways. One example is having a sling:resourceSuperType property in the node addressed by the resource type. See http://www.mail-archive.com/sling-dev@incubator.apache.org/msg02365.html and SLING-278 for more details.

If a resource type has no explicit resource super type, the resource super type is assumed to be "sling/servlet/default". That is the resource type used for default script selection is also acting as a basic resource type much like java.lang.Object does for other types in the Java language.

Script Locations

Scripts are looked up in a series of locations defined by the ResourceResolver.getSearchPath() and the resource type (and resource super types) of the requested resource:

```
{scriptPathPrefix}/{resourceTypePath}
```

The pseudo code for iterating the locations would be something like:

```
var type = resource.getResourceType();
while (type != null) {
    for (String root: resourceResolver.getSearchPath()) {
        String path = root + type.toPath();
        findScriptsIn(path);
    }

    if (type == defaultServlet) {
        type = null;
    } else {
        type = getResourceSuperType(type);
        if (type == null) {
            type = defaultServlet;
        }
    }
}
```

All requests are NOT equal

GET and HEAD request methods are treated differently than the other request methods. Only for GET and HEAD requests will the request selectors and extension be considered for script selection. For other requests the servlet or script name (without the script extension) must exactly match the request method.

That is for a PUT request, the script must be PUT.esp or PUT.jsp. For a GET request with a request extension of html, the script name may be html.esp or GET.esp.

Scripts for GET requests

Apart for supporting scripts named after the request method, scripts handling GET and HEAD requests may be named differently for Sling to support a more elaborate processing order.

Depending on whether request selectors are considered, a script may have two forms:

- a. Ignoring request selectors (e.g. there are none in the request URI) {resourceTypeLabel}.{requestExtension}.{scriptExtension}
- b. Handling request selectors {selectorStringPath}.{requestExtension}.{scriptExtension}

The constituents of these script names are as follows:

- {resourceTypeLabel} - The last path segment of the path created from the resource type. This part is optional if the {requestExtension} is used in the script name. The resource type might either be set via the sling:resourceType property on the accessed node or if that property is not there its primary node type (property jcr:primaryType) is taken as fallback.
- {requestExtension} - The request extension. This part may be ommitted if the request extension is "html", otherwise this part is required. If this part is ommitted, the {resourceTypeLabel} is required in the case of ignoring the selectors.
- {scriptExtension} - The extension, e.g. "esp" or "jsp", identifying the scripting langauage used.
- {selectorStringPath} - The selector string converted to a path, along the lines of selectorString.replace('.', '/'). If less selectors are specified in the script name than given in the request, the script will only be taken into consideration if the given selectors are the **first** selectors in the request. This means *sel1/sel2.html.jsp* will be a candidate for the request url */content/test.sel1.sel2.sel3.html* but not for */content/test.sel3.sel1.sel2.html*. So the order of selectors is relevant!

Priority

The rules for script path priorization is defined as follows:

- The more request selectors are matched, the better
- A script including the request extension matches better than one without a request extension (for html only)
- A script found earlier matches better than a script found later in the processing order. This means, that script closer to the original resource type in the resource type hierarchy is considered earlier.

Examples

Let's consider the following script paths for a request of a resource whose resource type is sling\sample and the request selectors are *print.a4* and the request extension is *html*:

- (0) GET.esp

- (1) sample.esp
- (2) html.esp
- (3) print.esp
- (4) print/a4.esp
- (5) print.html.esp
- (6) print/a4.html.esp
- (7) a4.html.esp
- (8) a4/print.html.esp

The priority of script selection would be (starting with the best one): (6) - (4) - (5) - (3) - (2) - (1) - (0). Note that (4) is a better match than (5) because it matches more selectors even though (5) has an extension match where (4) does not. (7) is not a candidate because it does not include the first selector (print) and (8) is not a candidate because it has the wrong order of selectors.

Last modified by Robert Munteanu on Wed Nov 22 22:30:38 2017 +0200