Servlets and Scripts

See also URL to Script Resolution which explains how Sling maps URLs to a script or and servlet.

Servlet Registration

Servlets can be registered as OSGi services. The following service reference properties are evaluated for Servlets defined as OSGi services of type javax.servlet.Servlet (all those property names are defined in org.apache.sling.api.servlets.ServletResolverConstants (since API 2.15.2) or org.apache.sling.servlets.resolver.internal.ServletResolverConstants (before API 2.15.2)):

Name	Description
sling.servlet.paths	A list of absolute paths under which the servlet is accessible as a Resource. The property value must either be a single String, an array of Strings or a Vector of Strings. A servlet using this property might be ignored unless its path is included in the *Execution Paths* (servletresolver.paths) configuration setting of the SlingServletResolver service. Either this property or the sling.servlet.resourceTypes property must be set, or the servlet is ignored. If both are set, the servlet is registered using both ways. Binding resources by paths is discouraged, see caveats when binding servlets by path below.
sling.servlet.resourceTypes	The resource type(s) supported by the servlet. The property value must either be a single String, an array of Strings or a Vector of Strings. Either this property or the sling.servlet.paths property must be set, or the servlet is ignored. If both are set, the servlet is registered using both ways.
sling.servlet.resourceSuperType	The resource super type, indicating which previously registered servlet could intercept the request if the request matches the resource super type better. The property value must be a single String. This property is only considered for the registration with sling.servlet.resourceTypes. (since version 2.3.0 of the org.apache.sling.api.servlets API, version 2.5.2 of the org.apache.sling.servlets.resolver bundle)
sling.servlet.selectors	The request URL selectors supported by the servlet. The selectors must be configured as they would be specified in the URL that is as a list of dot-separated strings such as *print.a4*. In case this is not empty the first selector(s) (i.e. the one(s) on the left) in the request URL must match, otherwise the servlet is not executed. After that may follow arbitrarily many non-registered selectors. The property value must either be a single String, an array of Strings or a Vector of Strings. This property is only considered for the registration with sling.servlet.resourceTypes.
sling.servlet.extensions	The request URL extensions supported by the servlet for requests. The property value must either be a single String, an array of Strings or a Vector of Strings. This property is only considered for the registration with sling.servlet.resourceTypes.
sling.servlet.methods	The request methods supported by the servlet. The property value must either be a single String, an array of Strings or a Vector of Strings. This property is only considered for the registration with sling.servlet.resourceTypes. If this property is missing, the value defaults to GET and HEAD, regardless of which methods are actually implemented/handled by the servlet. A value of * leads to a servlet being

Name	Description
	bound to all methods.
sling.servlet.prefix	The prefix or numeric index to make relative paths absolute. If the value of this property is a number (int), it defines the index of the search path entries from the resource resolver to be used as the prefix. The defined search path is used as a prefix to mount this servlet. The number can be -1 which always points to the last search entry. If the specified value is higher than than the highest index of the search paths, the last entry is used. The index starts with 0. If the value of this property is a string and parseable as a number, the value is treated as if it would be a number. If the value of this property is a string starting with "/", this value is applied as a prefix, regardless of the configured search paths! If the value is anything else, it is ignored. If this property is not specified, it defaults to the default configuration of the sling servlet resolver.
sling.core.servletName	The name with which the servlet should be registered. This registration property is optional. If one is not explicitly set, the servlet's name will be determined from either the property component.name, service.pid or service.id (in that order). This means that the name is always set (as at least the last property is always ensured by OSGi).

For a Servlet registered as an OSGi service to be used by the Sling Servlet Resolver, either one or both of the sling.servlet.paths or the sling.servlet.resourceTypes service reference properties must be set. If neither is set, the Servlet service is ignored.

Each path to be used for registration - either from the sling.servlet.paths property or constructed from the other sling.servlet.* properties - must be absolute. Any relative path is made absolute by prefixing it with a root path. This prefix may be set with the sling.servlet.prefix service registration property. If this property is not set, the first entry in the ResourceResolver search path for the ResourceResolver.getResource(String) method is used as the prefix. If this entry cannot be derived, a simpe slash - / - is used as the prefix.

If sling.servlet.methods is not specified, the servlet is only registered for handling GET and HEAD requests. Make sure to list all methods you want to be handled by this servlet.

Servlet Resource Provider

A ServletMounter listens for javax.servlet.Servlet services. This only applies to OSGi services implementing javax.servlet.Servlet. Each individual servlet will have a dedicated service instance of ServletResourceProvider associated to it, which will provide ServletResources in the resource tree, based on the servlet's registration properties. The actual resource path of such resources differs for servlets registered by type and those registered by path:

Servlet registered by	Full Resource Path
Path	<given path>.servlet
ResourceType	for each selector, extension and method combination one resource with path <resource type>[/[<selector with separator '/'>.][<extension>][<method>]].servlet'.

If multiple servlets are registered for the same metadata the one with the highest service ranking is returned in the virtual resource tree. The resources expose the following properties:

Property Name	Description
sling:resourceType	the resource type to which the servlet is registered. Is equal to the absolute resource path.
sling:resourceSuperType	the resource super type. Is sling/bundle/resource if not explicitly set.
servletName	the name of the servlet
servletClass	the fully-qualified class name of the underlying servlet

In addition each such resource can be adapted to a Servlet.

Caveats when binding servlets by path

Binding servlets by paths has several disadvantages when compared to binding by resource types, namely:

- path-bound servlets cannot be access controlled using the default JCR repository ACLs
- path-bound servlets can only be registered to a path and not a resource type (i.e. no suffix handling)
- if a path-bound servlet is not active, e.g. if the bundle is missing or not started, a POST might result in unexpected results. usually creating a node at /bin/xyz which subsequently overlays the servlets path binding
- the mapping is not transparent to a developer looking just at the repository

Given these drawbacks it is strongly recommended to bind servlets to resource types rather than paths.

Registering a Servlet using Java Annotations

The "new" (as of 2018) Sling Servlet annotations were presented by Konrad Windszus at adaptTo() 2018.

If you are working with the default Apache Sling development stack you can either use

- OSGi DS 1.4 (R7) component property type annotations (introduced with DS 1.4/OSGi R7, supported since bnd 4.1 being used in bnd-maven-plugin 4.1.0+ and maven-bundle-plugin 4.1.0+),
- OSGi DS annotations (introduced with DS 1.2/OSGi R5, properly supported since bnd 3.0, being used in maven-bundle-plugin 3.0.0) or
- Generic Felix SCR or Sling-specific @SlingServlet annotations from Apache Felix Maven SCR Plugin to register your Sling servlets:

The following examples show example code how you can register Servlets with Sling

1. OSGi DS 1.4 (R7) component property type annotations for Sling Servlets (recommended)

```
@Component(
service = { Servlet.class },
@SlingServletResourceTypes(
    resourceTypes="/apps/my/type",
    methods= "GET",
    extensions="html",
    selectors="hello")
public class MyServlet extends SlingSafeMethodsServlet {

    @Override
    protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws Servl
        ...
    }
}
```

This is only supported though if you use either the bnd-maven-plugin or the maven-bundle-plugin in version 4.0.0 or newer and use Sling which is at least compliant with OSGi R6 (DS 1.3). There is no actual run-time dependency to OSGi R7! The configuration for the bnd-maven-plugin should look like this in your pom.xml

```
<build>
  ...
  <plugins>
    <plugin>
      <groupId>biz.aQute.bnd</groupId>
      <artifactId>bnd-maven-plugin</artifactId>
      <version>4.0.0</version>
      <executions>
        <execution>
          <goals>
            <goal>bnd-process</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    ...
  </plugins>
  ...
</build>
...
<dependencies>
  ...
  <!-- dependency towards the custom component property type annotations for Sling Servlets -->
  <dependency>
    <groupId>org.apache.sling</groupId>
    <artifactId>org.apache.sling.servlets.annotations</artifactId>
    <version>1.2.4</version>
  </dependency>
  ...
</dependencies>
```

Please refer to the Javadoc of the package for other related annotations.

Starting with version 1.2.4 of the org.apache.sling.servlets.annotations you can also generate a value for the sling.servlet.resourceSuperType registration property, by using the resourceSuperType annotation property (its default value is sling/bundle/resource). In order for the property to be taken into consideration, your Sling instance has to provide version 2.5.2 or newer of the org.apache.sling.servlets.resolver bundle.

2. Simple OSGi DS 1.2 annotations (use only if you cannot use approach 1.)

```
@Component(
service = { Servlet.class },
property = {
    SLING_SERVLET_RESOURCE_TYPES + "=/apps/my/type"
    SLING_SERVLET_METHODS + "=GET",
    SLING_SERVLET_EXTENSIONS + "=html",
    SLING_SERVLET_SELECTORS + "=hello",
  }
)
public class MyServlet extends SlingSafeMethodsServlet {

    @Override
    protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws Servl
        ...
    }
}
```

3. The @SlingServlet annotation (evaluated by maven-scr-plugin, use only if you can neither use 1. nor 2.)

```
@SlingServlet(
    resourceTypes = "/apps/my/type",
    selectors = "hello",
    extensions = "html",
    methods = "GET")
public class MyServlet extends SlingSafeMethodsServlet {

    @Override
    protected void doGet(SlingHttpServletRequest request, SlingHttpServletResponse response) throws Servl
        ...
    }
}
```

Automated tests

The launchpad/test-services module contains test servlets that use various combinations of the above properties.

The launchpad/integration-tests module contains a number of tests (like the [ExtensionServletTest|https://github.com/apache/sling-org-apache-sling-launchpad-integration-tests/blob/master/src/main/java/org/apache/sling/launchpad/webapp/integrationtest/servlets/resolution/ExtensionServletTest.java] for example) that verify the results.

Such tests run as part of our continuous integration process, to demonstrate and verify the behavior of the various servlet registration mechanisms, in a way that's guaranteed to be in sync with the actual Sling core code. If you have an idea for additional tests, make sure to let us know!

Example: Registration by Path

```
sling.servlet.paths = [ "/libs/sling/sample/html", "/libs/sling/sample/txt" ]
sling.servlet.selectors = [ "img" ]
sling.servlet.extensions = [ "html", "txt", "json" ]
```

A Servlet service registered with these properties is registered under the following paths:

- /libs/sling/sample/html
- /libs/sling/sample/txt

The registration properties sling.servlet.selectors and sling.servlet.extensions *are ignored* because the servlet is registered only by path (only sling.servlet.paths property is set).

Example: Registration by Resource Type etc.

```
sling.servlet.resourceTypes = [ "sling/unused" ]
sling.servlet.selectors = [ "img", "tab" ]
sling.servlet.extensions = [ "html", "txt", "json" ]
```

A Servlet service registered with these properties is registered for the following resource types:

- <prefix>/sling/unused/img/html
- <prefix>/sling/unused/img/txt
- <prefix>/sling/unused/img/json
- <prefix>/sling/unused/tab/html
- <prefix>/sling/unused/tab/txt
- <prefix>/sling/unused/tab/json

As explained the Servlet is registered for each permutation of the resource types, selectors and extension. See above at the explanation of sling.servlet.prefix how <prefix> is defined.

It is more common to register for absolute resource types or at least explicitly define sling.servlet.prefix as well, because otherwise you are in most cases not sure under which absolute path the Servlet is registered (and therefore by which other paths it might get overwritten).

Servlet Lifecycle Issues

The Servlet API specification states the following with respect to the life cycle of Servlets:

> *The servlet container calls the init method exactly once after instantiating the servlet.*

This works perfectly in a regular servlet container which both instantiates and initializes the servlets. With Sling the tasks of instantiation and initialization are split:

- The provider of the Servlet service takes care of creating the servlet instance
- The Sling Servlet Resolver picks up the Servlet services and initializes and destroys them as needed

So Sling has not way of making sure a Servlet is only initialized and destroyed once in the life time of the Servlet object instance.

The provider of the Servlet service on the other can cope with this situation by making sure to drop the servlet instance once it is destroyed. The mechanism helping the provider here is the OSGi Service Factory.

Scripts are Servlets

The Sling API defines a SlingScript interface which is used to represent (executable) scripts inside of Sling. This interface is implemented in the scripting/core bundle in the DefaultSlingScript class which also implements the javax.servlet.Servlet.

To further simplify the access to scripts from the Resource tree, the scripting/core bundle registers an AdapterFactory to adapt Resources to Scripts and Servlets (the SlingScriptAdapterFactory). In fact the adapter factory returns

instances of the DefaultSlingScript class for both Scripts and Servlets.

From the perspective of the Servlet resolver, scripts and servlets are handled exactly the same. In fact, internally, Sling only handles with Servlets, whereas scripts are packed inside a Servlet wrapping and representing the script.

Default Servlet(s)

As explained in the Resolution Process section above, a default Servlet is selected if no servlet (or script) for the current resource type can be found. To make the provisioning of a default Servlet as versatile as provisioning per resource type Servlets (or scripts), the default Servlet is selected with just a special resource type sling/servlet/default.

The actual Servlet or Script called as the default Servlet is resolved exactly the same way as for any resource type. That is, also for the default Servlet selection, the request selectors and extension or method are considered. Also, the Servlet may be a Servlet registered as an OSGi service or it may be a Script stored in the repository or provided by any bundle.

Finally, if not even a registered default Servlet may be resolved for the request, because none has been registered, the servlets/resolver bundle provides a fall back the DefaultServlet with the following functionality:

- If an NonExistingResource was created for the request the DefaultServlet sends a 404 (Not Found)
- Otherwise the DefaultServlet sends a 500 (Internal Server Error), because normally at least a NonExistingResource should be created

---

OptingServlet interface

If a registered servlet implements the OptingServlet interface, Sling uses that servlet's accepts(SlingHttpServletRequest request) method to refine the servlet resolution process.

In this case, the servlet is only selected for processing the current request if its accept method returns true.

While an opting servlet seems to be a nice way of picking the right servlet to process the request, the use of an opting servlet is not recommended: the main reason is that it complicates the request processing, makes it less transparent what is going on during a request and prevents optimizations like caching the script resolution in an optimal manner. The other static options are usually sufficient for all use cases.

Servlet Resolution Order

The following order rules are being followed when trying to resolve a servlet for a given request URL and request method and multiple candidates would match. Then the following candidate is being picked (if one rule doesn't lead to one winner, the next rule is being evaluated):

1. The one with the highest number of matching selectors + extension
2. The one which is registered to a resource type closest to the requested one (when traversing the resource type hierarchy up)
3. The one with the highest service.ranking property

In case of an OptingServlet not matching the next candidate is being used.

Error Handler Servlet(s) or Scripts

Error handling support is described on the Errorhandling page.

Last modified by Radu Cotescu on Tue Dec 18 14:29:30 2018 +0100