Architecture

The following is a short list of highlights of Sling:

- **OSGi** — The Sling application is built as a series of OSGi bundles and makes heavy use of a number of OSGi core and compendium services.
- **Sling API** — To implement content based Web applications with Sling, an API has been defined, this extends the Servlet API and provides more functionality to work on the content.
- **Request Processing** — Sling takes a unique approach to handling requests in that a request URL is first resolved to a resource, then based on the resource (and only the resource) it selects the actual servlet or script to handle the request.
- **Resources** — The central mantra of Sling is the *Resource*, which represents the resource addressed by any request URL. It is the resource that is first resolved when handling a request. Based on the resource, a first servlet or script is then accessed to actually handle the request.
- **Servlets and Scripts** — Servlets and Scripts are handled uniformly in that they are represented as resources themselves and are accessible by a resource path.
- **Launchpad** — Sling uses a very thin launcher to integrate with an existing servlet container, launching Sling as a Web application or providing a main class to represent a standalone Java application.

The following sections elaborate on each of these highlights.

OSGi

OSGi is a consortium that has developed a specification to build modular and extensible applications. This offers various benefits. We deal mainly with two parts of the specifications: The Core Specification, which defines the OSGi Framework and Core Services, and the Compendium Services Specification, which defines a host of services that extend the functionality of the OSGi Framework.

OSGi Framework

The OSGi Framework is made up of three layers – Module, Lifecycle, and Services – that define how extensible applications are built and deployed. The responsibilities of the layers are:

- **Module** — Defines how a module, or a *Bundle* in OSGi-speak, is defined. Basically, a bundle is just a plain old JAR file, whose manifest file has some defined entries. These entries identify the bundle with a symbolic name, a version and more. In addition there are headers which define what a bundle provides Export-Package and what a bundle requires to be operative Import-Package and Require-Bundle.
- **Lifecycle** — The lifecycle layer defines the states a bundle may be in and describes the state changes. By providing a class, which implements the BundleActivator interface and which is named in the Bundle-Activator manifest header, a bundle may hook into the lifecycle process when the bundle is started and stopped.
- **Services** — For the application to be able to interact, the OSGi Core Specification defines the service layer. This describes a registry for services, which may be shared.

Compendium Services

Based on the OSGi Framework specification, the Compendium Services specification defines a (growing) number of extension services, which may be used by applications for various tasks. Of these Compendium Services, Sling is using just a small number:

- **Log Service** — Sling comes with its own implementation of the OSGi Log Service specification. The respective bundle not only provides this implementation, it also exports the SLF4J, Log4J and Commons Logging APIs needed for the Sling application to perform logging.
- **Http Service** — Sling leverages the OSGi Http Service to hook into a servlet container to provide the Web Application Framework mechanism.

- **Configuration Admin Service** — To simplify configuration of services in Sling, the OSGi Configuration Admin service is used. This provides a uniform API to configure services and to build configuration management agents.
- **Metatype Service** — The OSGi Metatype Service defines a way to describe the data types. Sling uses this service to describe the configurations that may be created using the Configuration Admin Service. These meta type descriptions are used by configuration management agents to present to user interface to manage the configurations.
- **Event Admin Service** — Sling uses the OSGi EventAdmin service to dispatch events when scheduling tasks.
- **Declarative Services** — One of the most important (beside the Log Service) services used by Sling is the Declarative Services Specification. This specification defines how to declaratively create components and services to have the Declarative Services runtime actually manage the lifecycle, configuration and references of components.

Sling API

The Sling API is an extension to the Servlet API which provides more functionality to interact with the Sling framework and also to extend Sling itself and to implement Sling applications.

Request Processing

Traditional Web Application framework emply more or less elaborate methods to select a Servlet or Controller based on the request URL, which in turn tries to load some data (usually from a database) to act upon and finally to render the result somehow.

Sling turns this processing around in that it places the data to act upon at the center and consequently uses the request URL to first resolve the data to process. This data is internally represented as an instance of the Resource interface. Based on this resource as well as the request method and more properties of the request URL a script or servlet is then selected to handle the request.

See the Servlets page for more information.

Resources

The Resource is one of the central parts of Sling. Extending from JCR's *Everything is Content*, Sling assumes *Everthing is a Resource*. Thus Sling is maintaining a virtual tree of resources, which is a merger of the actual contents in the JCR Repository and resources provided by so called resource providers.

Each resource has a path by which it is addressed in the resource tree, a resource type and some resource metadata (such as file size, last modification time). It is important to understand, that a Resource instance actually is only a handle to the actual data. By virtue of the adaptTo(Class<Type>) method, a resource may be coerced into another data type, which may then be used while processing the request. Examples of data types are javax.jcr.Node and java.io.InputStream.

See the Resources page for more information.

Servlets and Scripts

Scripts are usually provided as content in a JCR repository. But since Sling is using a resource tree, a script actually is represented as a Resource and may be provided from within a Bundle (by virtue of the bundle resource provider) or even from the platform file system (by virtue of the file system resource provider).

Accessing scripts in the resource tree, allows for a very easy to understand mapping from resource type to some script path.

Having found the script resource, we still need access to the appropriate script language implementation to evaluate the script. To this avail, Sling is making use of the Resource.adaptTo(Class<Type>) method: If a script language implementation is available for the extension of the script name an adaptor for the script resource can be found, which handles the evaluation of the script.

Besides scripting languages, such as ECMAScript, Groovy, JSP, Sling also supports regular servlets. To be able to use servlets for request processing, such servlets must be registered as OSGi services for the javax.servlet.Servlet interface and provide a number of service registration properties, which are used to use the servlets. In fact servlets thus registered as OSGi services are mapped into the resource tree by means of a servlet resource provider. This resource provider mapps the servlets into the resource tree using the service registration properties to build one or more resource paths for the servlet.

As a result of mapping servlets into the resource tree and the possibility to adapt resource to an adaptor data type, scripts and servlets may be handled completely transparently: The servlet resolver just looks for a resource matching the resource type and adapts the resource found to javax.jcr.Servlet. If the resource happens to be

provided by a servlet resource provider, the adapter is of course the servlet itself. If the resource happens to be a script, the adapter is a servlet facade which internally calls the script language implementation to evaluate the script.

See the Servlet Resolution page for more information.

Launchpad

Sling may be launched as a standalone application using the Sling Application or as a Web Application running inside any Servlet API 2.4 or newer Servlet Container.

The Sling Application is a standalone Java Application which is really small: Just the main class and some glue classes. The OSGi framework as well as the OSGi API libraries are packaged as a JAR file, which is loaded through a custom classloader. This enables to update the framework and/or OSGi API libraries from within Sling by updating the system bundle.

The Sling Servlet is equally small as the Sling Application. It uses the Felix HttpService bridge as the glue between the servlet container and the OSGi framework.

As we have seen, Sling may be launched as a standalone Java Application or as a Web Application inside any compliant Servlet Container. To hide the differences of the launching mechanism, Sling internally registers a Servlet with an OSGi HttpService. Regardless of how Sling is launched, the Felix implementation of the OSGi HttpService specification is used. When Sling is launched as a standalone Java Application, Felix HttpService uses an embedded version of the Jetty servlet container. When Sling is launched as a Web Application, the Felix HttpService Bridge is used.

Optionally, PAX Web's implementation of HttpService can be used when Sling is launched as a standalone Java Application. See the Maven Launchpad Plugin page for information on how to do this.

See The Sling Launchpad for more information.

Last modified by Bertrand Delacretaz on Fri Sep 29 15:36:08 2017 +0200