Deep Dive into Sling

Tyler Maynard
AEM DEVELOPER

@TylersDesk www.tylermaynard.com

Overview

Sling architecture

System Users

Working with Sling servlets

Sling resolution process

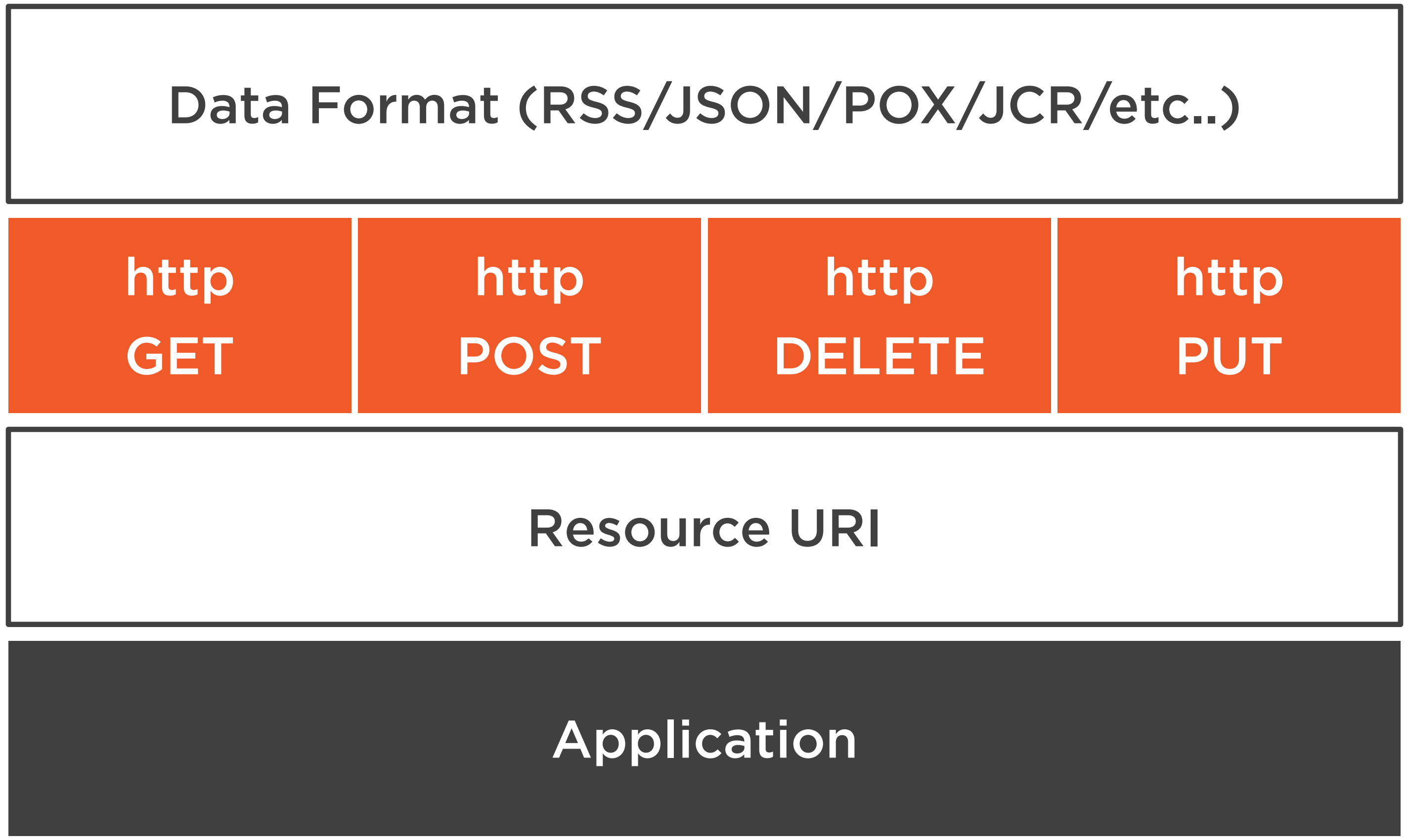
Sling mappings

Sling models

Sling events and event scheduling

The Sling Architecture

RESTful Architecture



RESTful Architectural Properties

Performance and network efficiency

Scalability

Simplicity of interfaces

Modifiability of components

Visibility of communication

Portability of components

Reliability in resistance to internal failure

Advantages of REST

Well documented, established and used methodology

Resource centric

Accessible

No multiple protocols

No specific format for response payload

HTTP Security Model

System Users

Authentication Problem

Authentication needed to access resources

Background tasks need access to resources

Proposed Solution

Prevent overuse and abuse administrative sessions

Access resolvers and sessions without passwords

Service users for service level access

Configure service users to services

Service Authentication Concept

Service

A piece or collection of functionality

Service Name

Used to uniquely identify a service

Subservice Name

each part of the service may be further identified (optional)

$$\text{service-id} = \text{service-name}\ \underbrace{[\ \text{“:”}\ \text{subservice-name}\ ]}_{\text{optional}}$$

Service Authentication Implementation

ServiceUserMapper

ResourceResolver Factory

SlingRepository service

ServiceUserMapper

```
String getServiceUserID(Bundle bundle, String subServiceName);
```

ResourceResolverFactory

```
ResourceResolver getServiceResourceResolver(Map<String, Object>
authenticationInfo) throws LoginException;
```

SlingRepositoryservice

```
Session loginService(String subServiceName, String workspace) throws
LoginException, RepositoryException;
```

Deprecation of Administrative Authentication

1 - ResourceResolverFactory.getAdministrativeResourceResolver

2 - ResourceProviderFactory.getAdministrativeResourceProvider

3 - SlingRepository.loginAdministrative

Create a System User

Demo

Create a service user

Assign user to a group

Configure the Service User Mapping service

Sling Servlets

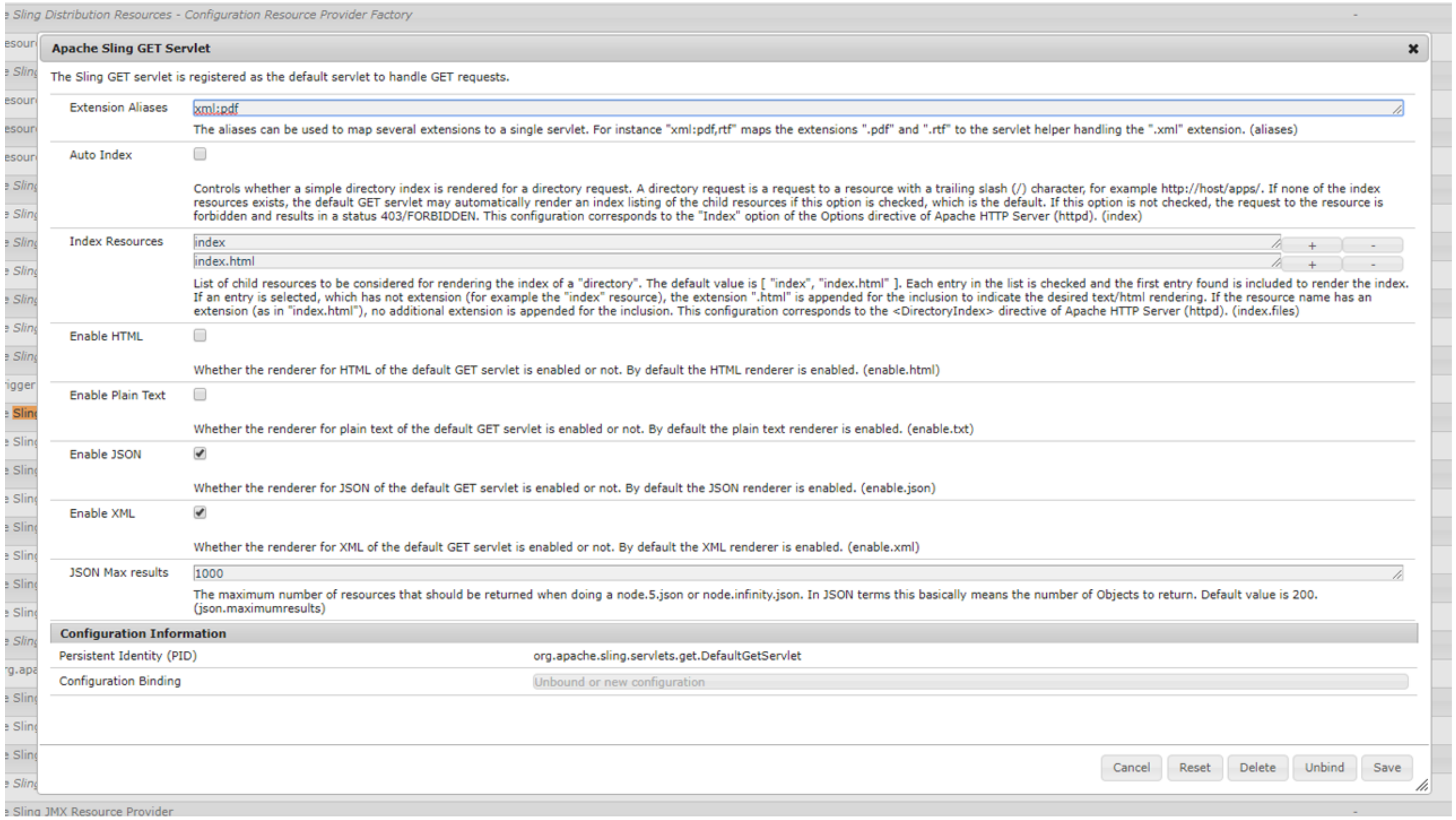
Working with Sling Servlets

Resource selects servlet or script to handle request

GET method

POST method

Configuring the Default Sling GET Servlet



Sling Distribution Resources - Configuration Resource Provider Factory

Apache Sling GET Servlet

The Sling GET servlet is registered as the default servlet to handle GET requests.

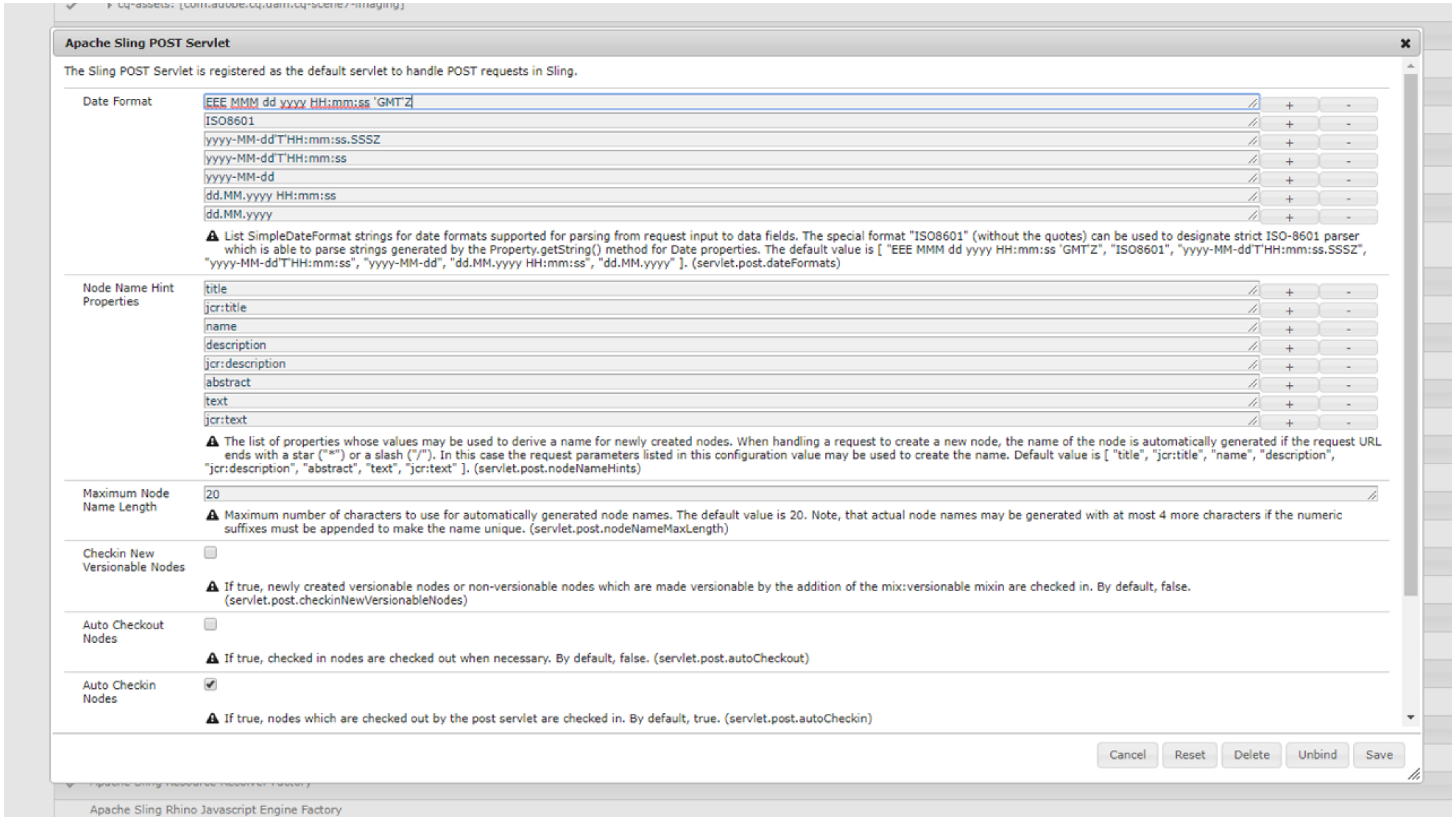
Extension Aliases	xml:pdf The aliases can be used to map several extensions to a single servlet. For instance "xml:pdf,rtf" maps the extensions ".pdf" and ".rtf" to the servlet helper handling the ".xml" extension. (aliases)
Auto Index	☐ Controls whether a simple directory index is rendered for a directory request. A directory request is a request to a resource with a trailing slash (/) character, for example http://host/apps/. If none of the index resources exists, the default GET servlet may automatically render an index listing of the child resources if this option is checked, which is the default. If this option is not checked, the request to the resource is forbidden and results in a status 403/FORBIDDEN. This configuration corresponds to the "Index" option of the Options directive of Apache HTTP Server (httpd). (index)
Index Resources	index index.html List of child resources to be considered for rendering the index of a "directory". The default value is ["index", "index.html"]. Each entry in the list is checked and the first entry found is included to render the index. If an entry is selected, which has not extension (for example the "index" resource), the extension ".html" is appended for the inclusion to indicate the desired text/html rendering. If the resource name has an extension (as in "index.html"), no additional extension is appended for the inclusion. This configuration corresponds to the <DirectoryIndex> directive of Apache HTTP Server (httpd). (index.files)
Enable HTML	☐ Whether the renderer for HTML of the default GET servlet is enabled or not. By default the HTML renderer is enabled. (enable.html)
Enable Plain Text	☐ Whether the renderer for plain text of the default GET servlet is enabled or not. By default the plain text renderer is enabled. (enable.txt)
Enable JSON	☑ Whether the renderer for JSON of the default GET servlet is enabled or not. By default the JSON renderer is enabled. (enable.json)
Enable XML	☑ Whether the renderer for XML of the default GET servlet is enabled or not. By default the XML renderer is enabled. (enable.xml)
JSON Max results	1000 The maximum number of resources that should be returned when doing a node.5.json or node.infinity.json. In JSON terms this basically means the number of Objects to return. Default value is 200. (json.maximumresults)

Configuration Information

Persistent Identity (PID)	org.apache.sling.servlets.get.DefaultGetServlet
Configuration Binding	Unbound or new configuration

Sling JMX Resource Provider

Configuring the Sling POST Servlet



Apache Sling POST Servlet

The Sling POST Servlet is registered as the default servlet to handle POST requests in Sling.

Date Format	EEE MMM dd yyyy HH:mm:ss 'GMT'Z	+	-
	ISO8601	+	-
	yyyy-MM-dd'T'HH:mm:ss.SSSZ	+	-
	yyyy-MM-dd'T'HH:mm:ss	+	-
	yyyy-MM-dd	+	-
	dd.MM.yyyy HH:mm:ss	+	-
	dd.MM.yyyy	+	-

List SimpleDateFormat strings for date formats supported for parsing from request input to data fields. The special format "ISO8601" (without the quotes) can be used to designate strict ISO-8601 parser which is able to parse strings generated by the Property.getString() method for Date properties. The default value is ["EEE MMM dd yyyy HH:mm:ss 'GMT'Z", "ISO8601", "yyyy-MM-dd'T'HH:mm:ss.SSSZ", "yyyy-MM-dd'T'HH:mm:ss", "yyyy-MM-dd", "dd.MM.yyyy HH:mm:ss", "dd.MM.yyyy"]. (servlet.post.dateFormats)

Node Name Hint Properties	title	+	-
	jcr:title	+	-
	name	+	-
	description	+	-
	jcr:description	+	-
	abstract	+	-
	text	+	-
	jcr:text	+	-

The list of properties whose values may be used to derive a name for newly created nodes. When handling a request to create a new node, the name of the node is automatically generated if the request URL ends with a star ("*") or a slash ("/"). In this case the request parameters listed in this configuration value may be used to create the name. Default value is ["title", "jcr:title", "name", "description", "jcr:description", "abstract", "text", "jcr:text"]. (servlet.post.nodeNameHints)

Maximum Node Name Length: 20

Maximum number of characters to use for automatically generated node names. The default value is 20. Note, that actual node names may be generated with at most 4 more characters if the numeric suffixes must be appended to make the name unique. (servlet.post.nodeNameMaxLength)

Checkin New Versionable Nodes: ☐

If true, newly created versionable nodes or non-versionable nodes which are made versionable by the addition of the mix:versionable mixin are checked in. By default, false. (servlet.post.checkinNewVersionableNodes)

Auto Checkout Nodes: ☐

If true, checked in nodes are checked out when necessary. By default, false. (servlet.post.autoCheckout)

Auto Checkin Nodes: ☑

If true, nodes which are checked out by the post servlet are checked in. By default, true. (servlet.post.autoCheckin)

Creating Content with POST

```
<form method="POST" action="http://host/mycontent" enctype="multipart-
form/data">
        <input type="text" name="title" value=""/>
        <input type="text" name="description" value="" />
</form>
```

Creating Content with POST via CURL

```
curl -u admin:admin
            -F"jcr:primaryType=nt:unstructured"
            -Ftitle="some title text"
            -Ftext="some body text content“
      http://host/some/new/content
```

Create a Sling Servlet

Demo

Create Servlet using the path method

Update Servlet to use ResourceType method

Understanding the Sling Resolution Process

Everything is a Resource

Path

Name

ResourceType

Resource First Request Processing

$$\begin{tikzcd}
\begin{array}{c}\textbf{Incoming}\\\textbf{request}\end{array} \arrow[r] & \begin{array}{c}\textbf{Resolve}\\\textbf{to a}\\\textbf{resource}\end{array} \arrow[r] & \begin{array}{c}\textbf{Servlet}\\\textbf{or}\\\textbf{rendering script}\end{array}
\end{tikzcd}$$

Basic Steps of Processing Requests

1. Properties of the content item itself

2. The HTTP method used to make the request

3. Simple naming convention within the URL

Resolving Resource Steps

Decompose the URL

Search for a file indicated by the URL

Resolve the Resource

Resolve the rendering script/servlet

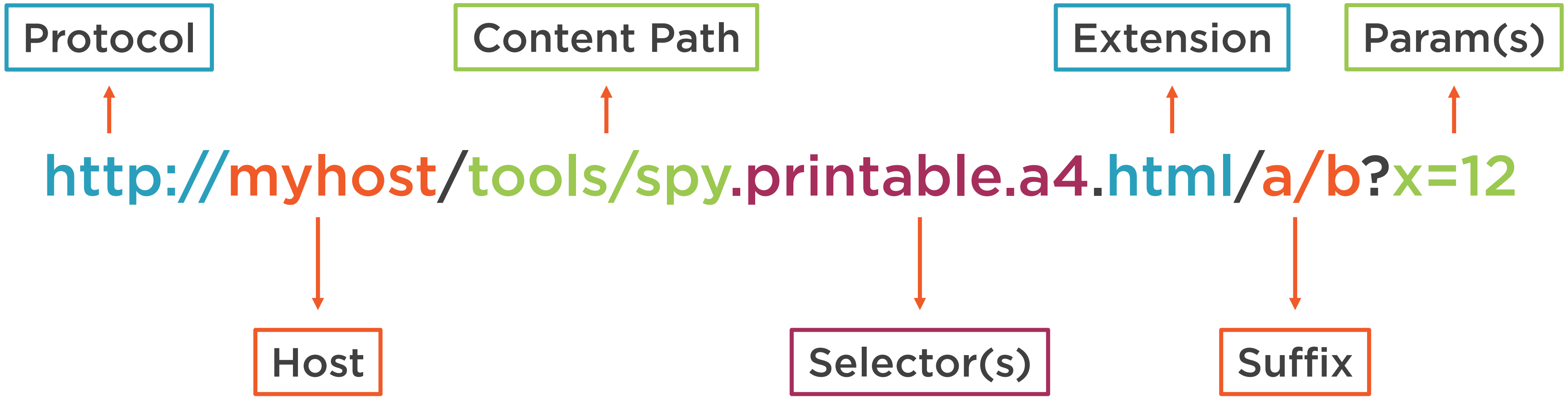
Create rendering chain

Invoke rendering chain

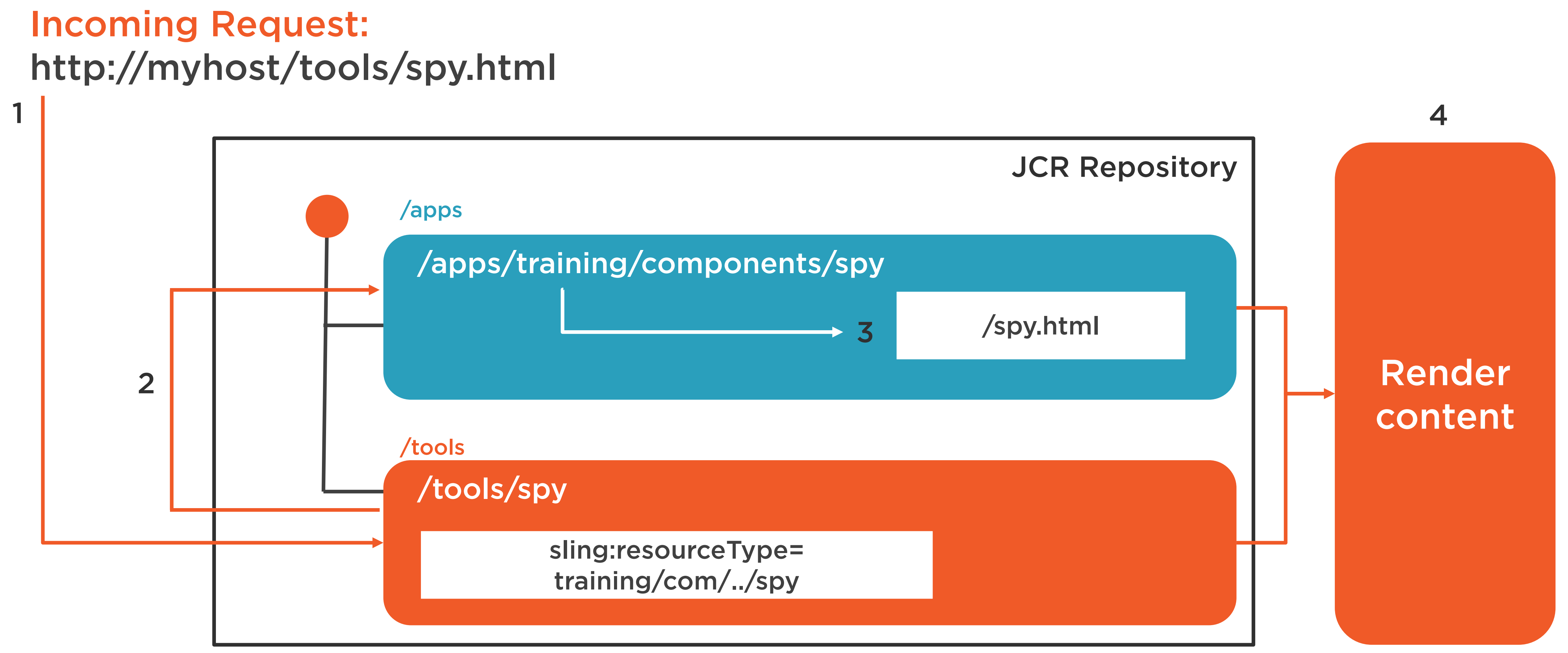
Decomposing the URL

http://myhost/tools/spy.printable.a4.html/a/b?x=12

Decomposing the URL

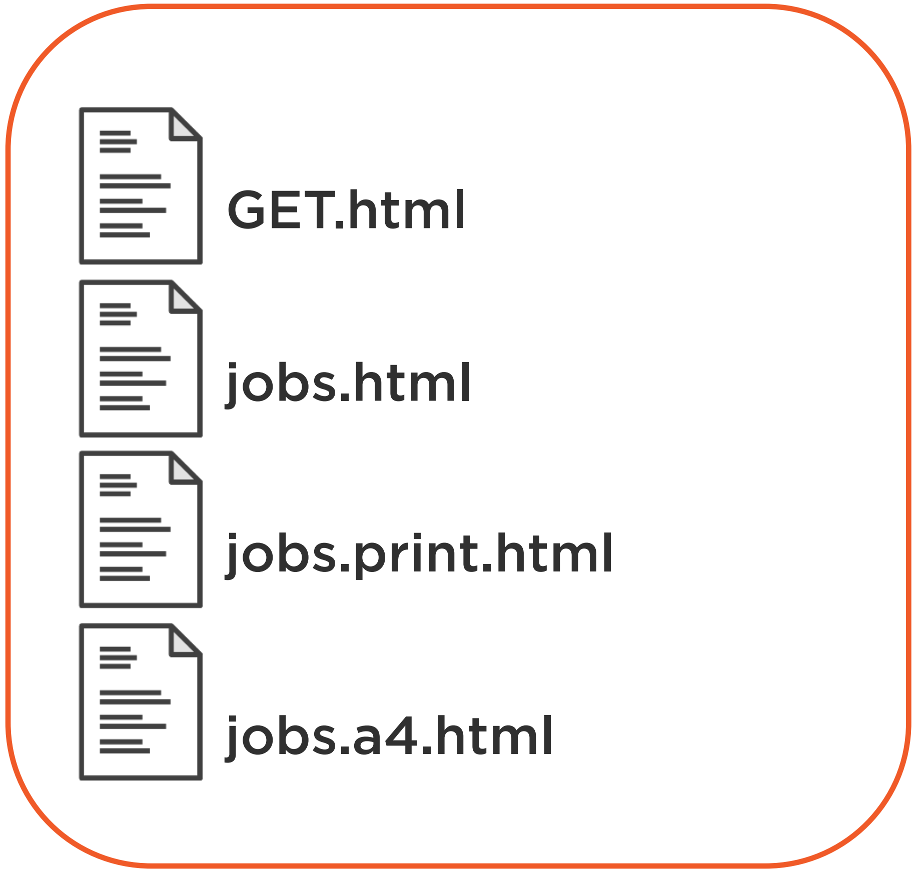


Resolving Requests Example



Locating and Rendering Scripts

Files in repository under **hr/jobs**



Request
URL: /content/developer.print.html
sling:resourceType= hr/jobs

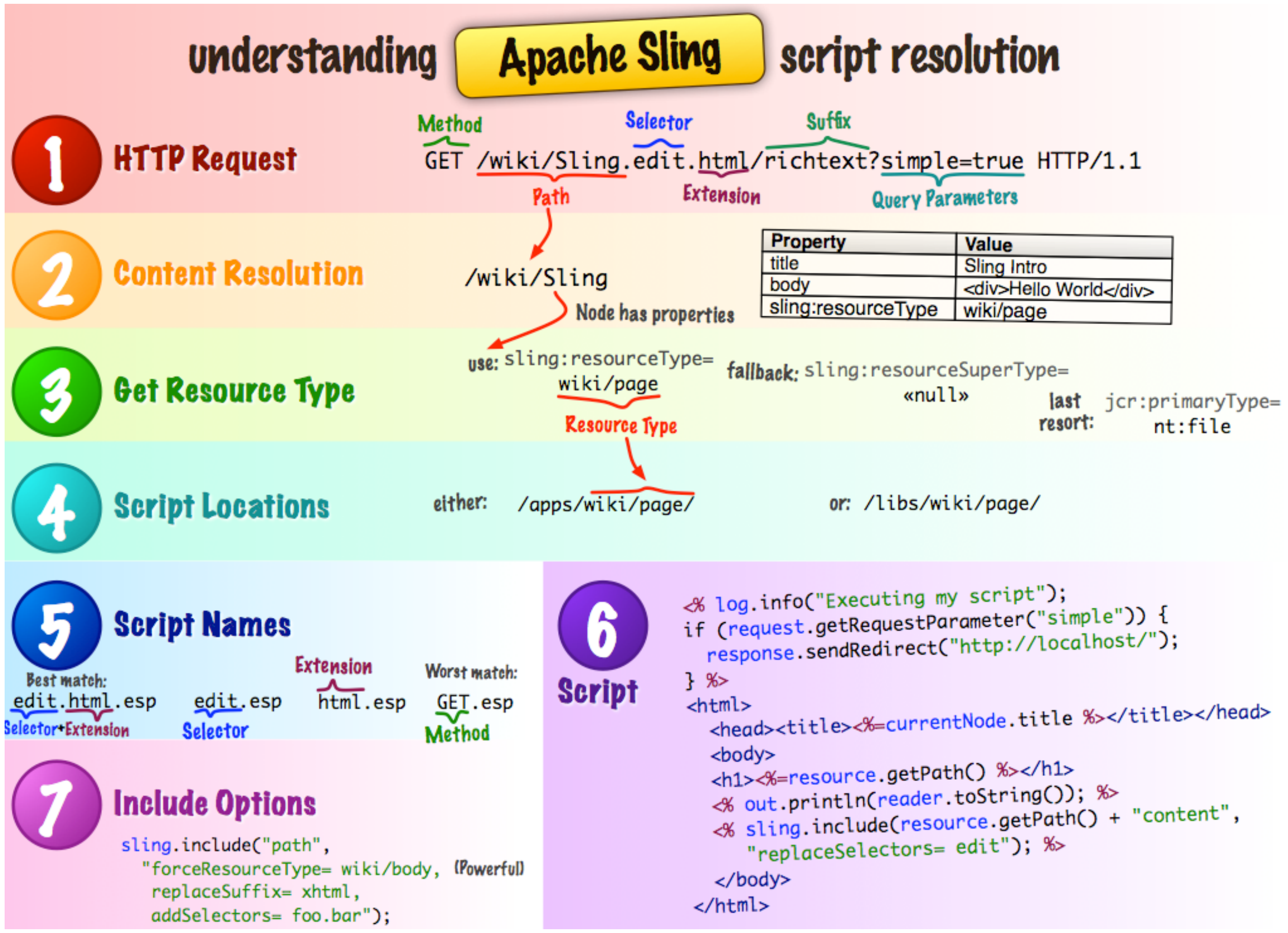
Result Order
1) jobs.print.html
2) jobs.html
3) GET.html
4) jobs.a4.html

Rendering Scripts Super Types

sling:resourceSuperType property of the resource

sling:resourceSuperType property of the node to which the sling:resourceType points

Understanding Apache Sling Script Resolution



```
GET /wiki/Sling.edit.html/richtext?simple=true HTTP/1.1
```

Property	Value
title	Sling Intro
body	<div>Hello World</div>
sling:resourceType	wiki/page

```
use: sling:resourceType=
     wiki/page
fallback: sling:resourceSuperType=
          «null»
last resort: jcr:primaryType=
             nt:file
```

```
edit.html.esp    edit.esp    html.esp    GET.esp
```

```
<% log.info("Executing my script");
if (request.getRequestParameter("simple")) {
  response.sendRedirect("http://localhost/");
} %>
<html>
  <head><title><%=currentNode.title %></title></head>
  <body>
  <h1><%=resource.getPath() %></h1>
  <% out.println(reader.toString()); %>
  <% sling.include(resource.getPath() + "content",
     "replaceSelectors= edit"); %>
  </body>
</html>
```

```
sling.include("path",
  "forceResourceType= wiki/body,
   replaceSuffix= xhtml,
   addSelectors= foo.bar");
```

The Resource Resolver and Sling Mappings

Resource Resolver Abstraction

The path resolution

Access to the persistence layer(s)

Resource Mapping Node Types

sling:ResourceAlias

sling:MappingSpec

sling:Mapping

Resource Mapping Properties

sling:match

sling:redirect

sling:status

sling:internalredirect

sling:alias

Resource Mapping Rules

Resource in path has sling:match -> value used in segment

Resource has sling:redirect or sling:internalRedirect -> used as table entries

Examples of Resource Mapping

```
/etc/map
      +-- http
           +-- example.com.80
           |    +-- sling:redirect = "http://www.example.com/"
           +-- www.example.com.80
           |    +-- sling:internalRedirect = "/example"
           +-- any_example.com.80
           |    +-- sling:match = ".+\.example\.com\.80"
           |    +-- sling:redirect = "http://www.example.com/"
           +-- localhost_any
           |    +-- sling:match = "localhost\.\d*"
           |    +-- sling:internalRedirect = "/content"
           |    +-- cgi-bin
           |    |    +-- sling:internalRedirect = "/scripts"
           |    +-- (stories)
           |    |    +-- sling:internalRedirect = “/anecdotes/$1”
           +-- regexmap
                +-- sling:match = "$1.example.com/$2"
                +-- sling:internalRedirect = "/content/([^/]+)/(.*)"
```

Examples of Resource Mapping - 1

```
etc/map
  +--http
    +--example.com.80
    |   +--sling:redirect=“http://www.example.com/”
    ...
```

Regular Expression	Redirect	Internal
http/example.com.80	http://www.example.com	no

Examples of Resource Mapping - 2

```
etc/map
  +--http
    ...
    +--www.example.com.80
    |   +--sling:internalRedirect="/example"
    ...
```

Regular Expression	Redirect	Internal
`http/www.example.com.80`	`/example`	yes

Examples of Resource Mapping - 3

```
etc/map
  +--http
    ...
    +-- any_example.com.80
    |   +-- sling:match = ".+\.example\.com\.80"
    |   +-- sling:redirect = "http://www.example.com/"
    ...
```

Regular Expression	Redirect	Internal
http/.+.example.com.80	http://www.example.com	no

Examples of Resource Mapping - 4

```
etc/map
  +--http
    ...
    +--localhost_any
    |   +-- sling:match = "localhost\.\d*"
    |   +-- sling:internalRedirect = "/content"
    ...
```

Regular Expression	Redirect	Internal
`http/localhost.\d*`	`/content`	yes

Examples of Resource Mapping - 5

```
etc/map
  +--http
    ...
    +--localhost_any
    ...
    |  +-- cgi-bin
    |  |  +-- sling:internalRedirect = “/scripts”
    ...
```

Regular Expression	Redirect	Internal
http/localhost.\d*/cgi-bin	/scripts	yes

Examples of Resource Mapping - 6

```
etc/map
  +--http
    ...
    +--localhost_any
    ...
    |  +-- (stories)
    |  |  +-- sling:internalRedirect = "/anecdotes/$1"
    ...
```

Regular Expression	Redirect	Internal
`http/localhost.\d*/(stories)`	`/anecdotes/stories`	yes

Adapting Resources and Sling Models

Adapting Resources

```
Node node = resource.adaptTo(Node.class)
```

Implementing Adaptable.adaptTo()

By the object itself

By an AdapterFactory

Combination of the two

Working with Sling Models

Annotation-driven

Use standard annotations

Pluggable

Support resource properties, SlingBindings, OSGi services, request attributes

Adaptable

Classes and Interfaces

Work with Sling inftrastructure

Using Sling Models

Have a model object (Java class or interface)

Need to use it in AEM

Map POJO to Sling Resource

Saves time

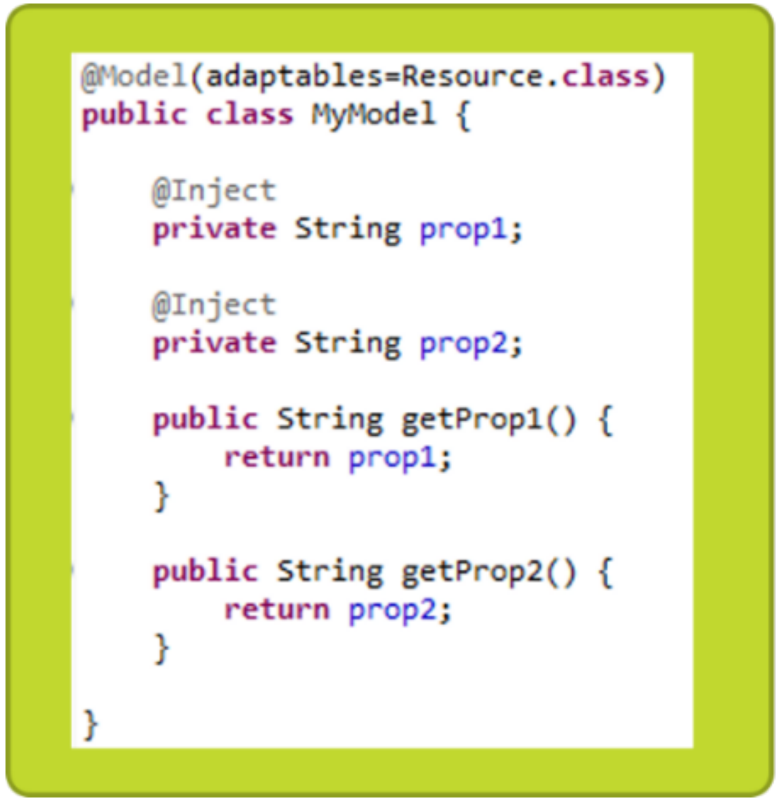
Classes and interfaces

OOTB resource properties, SlingBindings, OSGi services, etc

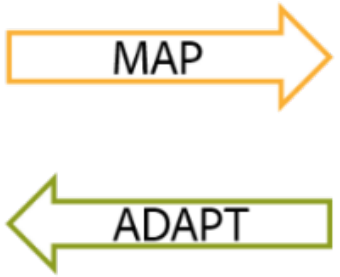
Adaptable

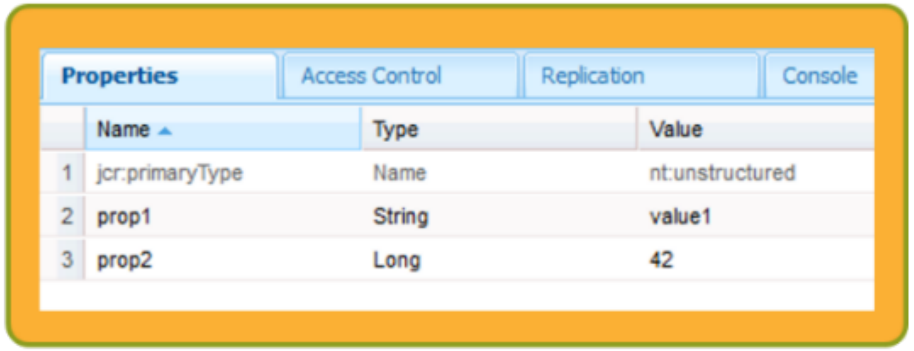
Mock dependencies

Understanding Sling Models



```
@Model(adaptables=Resource.class)
public class MyModel {

    @Inject
    private String prop1;

    @Inject
    private String prop2;

    public String getProp1() {
        return prop1;
    }

    public String getProp2() {
        return prop2;
    }

}
```





	Name	Type	Value
1	jcr:primaryType	Name	nt:unstructured
2	prop1	String	value1
3	prop2	Long	42

Sling Model Classes

```
@Model(adaptables=Resource.class)
public class MyModel {
    @Inject
    private String propertyName;
```

Sling Model Interfaces

```
@Model(adaptables=Resource.class)
public interface MyModel {
  @Inject
  private String getPropertyName();
}
```

Sling Models with Sling Adapter

```
MyModel model = resource.adaptTo(MyModel.class)
```

@Named Annotation

```
public class myModel {

    @Inject @Named("secondPropertyName")
```

Sling Model Annotations

Sling Model Annotation	Code Snippet
@Model	`@Model(adaptables = Resource.class)`

@Inject	`@Inject private String propertyName; (class)`
	`@Inject String getPropertyName(); (interface)`

@Default	`@Inject @Default(values=”AEM”) private String technology;`
@Optional	`@Inject @Optional private String otherName`

Sling Model Annotations Continued

Sling Model Annotation	Code Snippet
@Named	`@Inject @Named(“title”) private String page Title;`

@Via	`@Model(adaptables=SlingHttpServletRequest.class)`
	`Public interface SlingModelDemo {`
	`@Inject @Via(“resource”) String getPropertyName(); }`

@Source	`@Model(adaptables=SlingHttpServletRequest.class)`
	`@Inject @Source(“script-bindings”) Resource getResource();`

@PostConstruct	`@PostConstruct`
	`protected void sayHello() { logger.info(“hello”); }`

Injector Specific Annotations

Less code to write

More robust

Better IDE support

Create a Sling Model

Demo

Create StockModel Sling model

Create servlet

Create resource to map model to

Test servlet

Event Handling

Publishing Events

1. Get EventAdmin service

2. Create event object

3. Send the event

Sending Job Events

org.osgi.service.event.EventHandler (interface)

org.apache.sling.event.JobConsumer (interface)

Demo

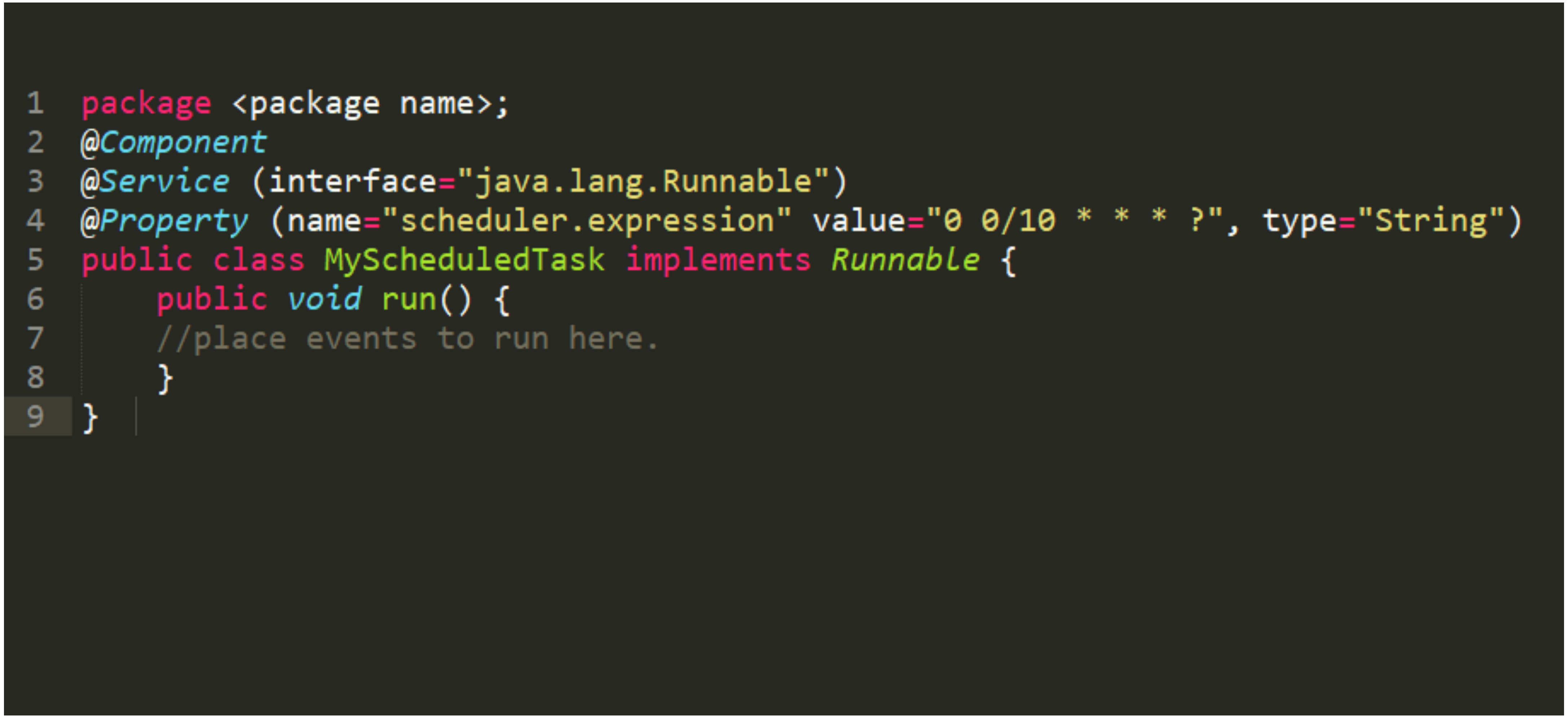
Create job consumer

Activate a page

Validate in logs

Working with Sling Schedules

OSGi Service Fired by Quartz



```java
package <package name>;
@Component
@Service (interface="java.lang.Runnable")
@Property (name="scheduler.expression" value="0 0/10 * * * ?", type="String")
public class MyScheduledTask implements Runnable {
    public void run() {
    //place events to run here.
    }
}
```

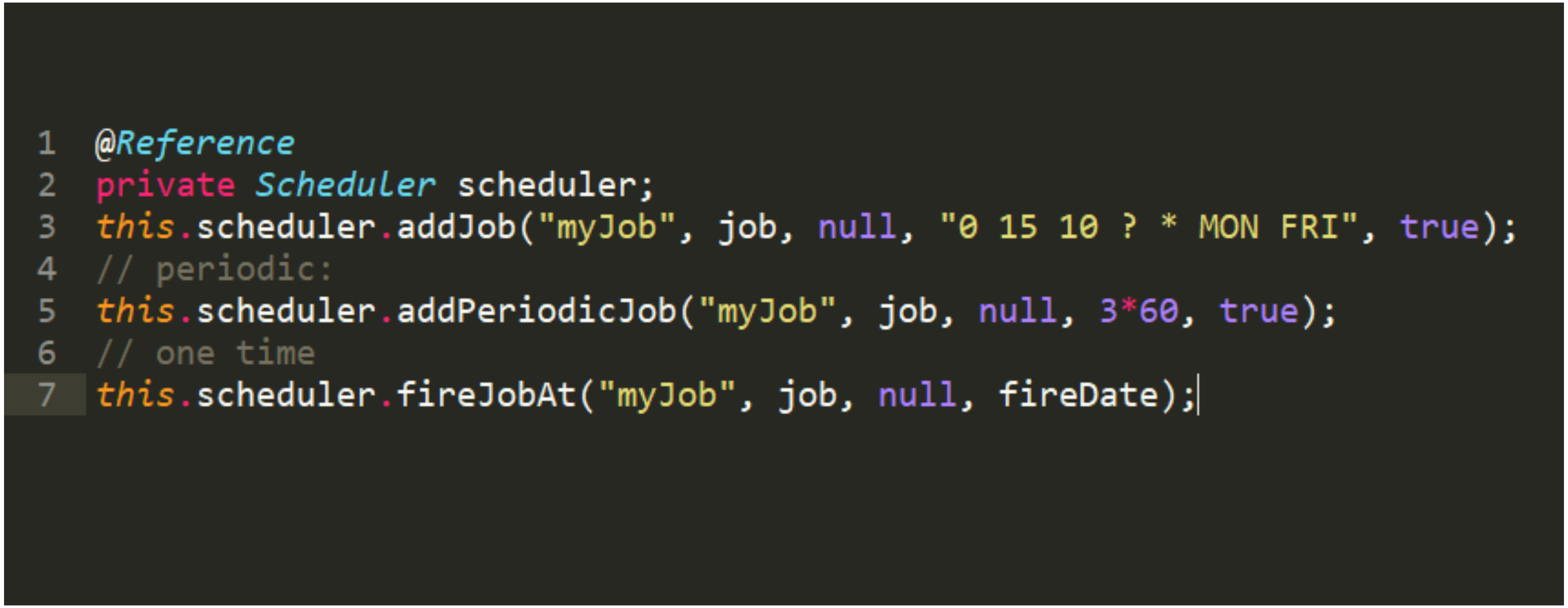
Scheduling Jobs

```
@Property(name=“scheduler.period”, value=“10”, type=“Long”)
```

Preventing Concurrent Execution

```
@Property(
    name="scheduler.concurrent",
    value="false",
    type="Boolean",
    private="true"
)
```

Scheduling Jobs Programmatically



```java
@Reference
private Scheduler scheduler;
this.scheduler.addJob("myJob", job, null, "0 15 10 ? * MON FRI", true);
// periodic:
this.scheduler.addPeriodicJob("myJob", job, null, 3*60, true);
// one time
this.scheduler.fireJobAt("myJob", job, null, fireDate);
```

Demo

Schedule a job

Check job is running

Update configuration

Overview

Sling architecture

System users and Sling servlets

Sling's resolution process

Sling mappings

Sling models

Event and event scheduling with Sling