
APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 26-28 SEPTEMBER 2012

# APACHE SLING: BASIC CONCEPTS

Rainer Bartl, Peter Mannel

# About the speakers

**Rainer Bartl**
Senior CMS Developer

**Peter Mannel**
Senior CMS Developer

**pro!vision GmbH**
Wilmersdorfer Str. 50-51
10627 Berlin
http://www.pro-vision.de

PRO!VISION
SOFTWARE CRAFTSMANSHIP

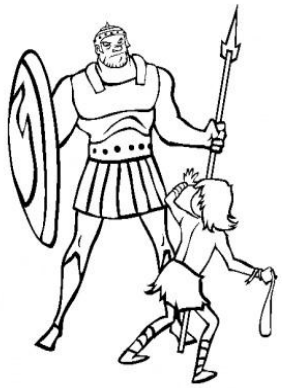

Spezialized on Adobe CQ and it‘s technology stack since 2003

# What is Apache Sling?

- Content-centric web (application) framework
- Built around *RESTful* principles
- Powered by *OSGI*
- Apache *Open Source* project
- Supports Scripting (JSR-233)


# Apache Sling History

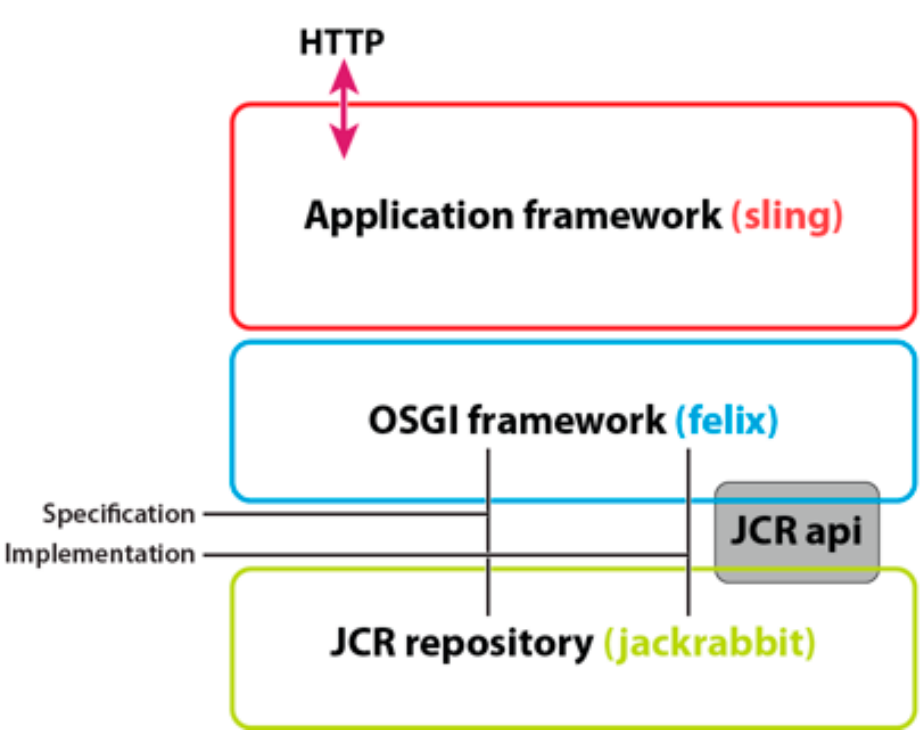
- Started as an internal project at Day Software
- Entered the Apache Incubator in September 2007.
- 2009 top level project of the Apache Software Foundation.
- Quote by Roy Fielding:
  - *The name is biblical in nature. The story of David: the weapon he uses to slay the giant Goliath is a sling. Hence, our David's favorite weapon. It is also the simplest device for delivering content very fast.*



# Agenda

- **General**
  - **Architecture**
- Request Handling
  - URL decomposition
  - Dispatching Requests
  - Servlets and Scripts
  - Filters
- Resources
  - Resources
  - Resource Provider and Resource Resolver
  - Resource Resolution Mapping
- Sling API
  - Adapters
  - Sling API

# Sling Architecture




Architectural style

- processes HTTP requests in a RESTful way
- manages bundles as app components
- provides system services to bundles
- stores the content

# Excursion: What is REST?

- *REST* stands for „REpresentational State Transfer”
- “Architectural style for designing distributed systems“
- „The Web as it was meant to be“
- It‘s centered around the transfer of *Representations* of *Resources*

Roy Fielding

adaptTo()

# Excursion: REST Concepts

- A resource can essentially be anything !
- Resources are independent from their *representation*
- An *URL* identifies a resource
- URLs have an implicit hierarchy
- *Methods* perform *Operations* on resources
- HTTP status codes are used to indicate result success

# Excursion: HTTP Methods (Verbs)

- The most common methods are:
  - GET - Retrieve an item or a list of items
  - POST - Create an item
  - PUT - Update an item
  - DELETE - Remove an item

- *Safe* methods shouldn‘t change anything
- *Idempotent* methods shouldn‘t change anything *beyond* their first execution

adaptTo()

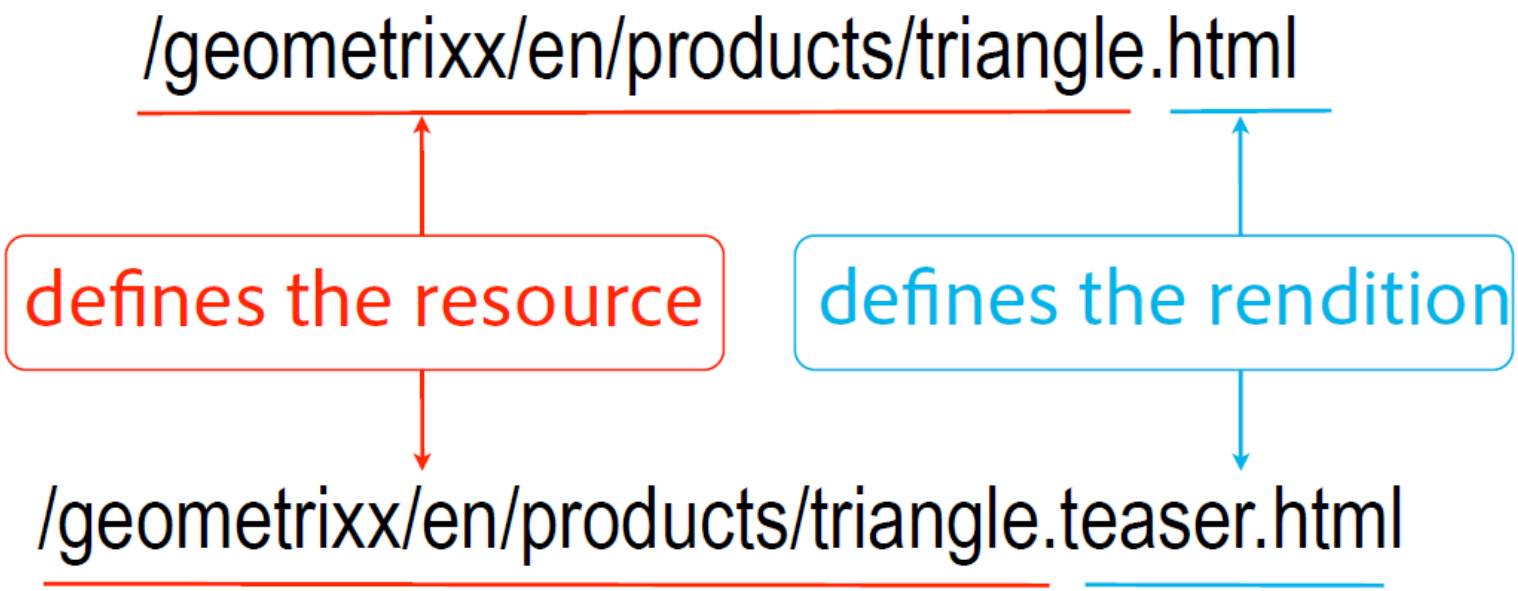
# Agenda

- General
  - Architecture
- **Request Handling**
  - **URL decomposition**
  - **Dispatching Requests**
  - **Servlets and Scripts**
  - **Filters**
- Resources
  - Resources
  - Resource Provider and Resource Resolver
  - Resource Resolution Mapping
- Sling API
  - Adapters
  - Sling API

# Sling Request Processing Steps

- Decompose the URL
- Search for a file indicated by the URL
- Resolve the resource (using the path from the URL)
- Resolve rendering script/servlet
  - Don‘t call scripts directly in Sling!
  - Primary source: Resource Type
- Create rendering chain
  - Configurable (Servlet) filters
  - Rendering Servlet
- Invoke rendering chain

# URL Decomposition



# URL Decomposition - Resource Path

- **Resource Path**
  - The substring of the request URL before the first dot (.)
  - Might be the complete URL if there‘s no dot

  Example: **/some/path**.s1.s2.html

# URL Decomposition - Selectors

- **Selectors**
  - Substring between the first dot and the dot leading the extension
  - Used for alternative methods of rendering the content
  - Multiple selectors possible
  - Optional

  Example: /some/path.**s1.s2**.html

# URL Decomposition - Extension

- **Extension**
  - The string between the last dot after the resource path and the next slash
  - Specifies the content format
  - Optional

  Example: /some/path.s1.s2.**html**

# URL Decomposition - Suffix Path

- **Suffix Path**
  - Path starting with the slash up to the end of the request URL
  - At least a dot must be in the URL to let Sling detect the suffix path
  - Can be used to provide additional information for the processing script

Example: /some/path.s1.s2.html**/suffix.html**

# URL Decomposition Examples

| URI | Resource Path | Selectors | Extension | Suffix Path |
|---|---|---|---|---|
| /a/b | /a/b | null | null | null |
| /a/b.html | /a/b | null | html | null |
| /a/b.s1.html | /a/b | s1 | html | null |
| /a/b.html/c/d | /a/b | null | html | /c/d |

# Resource Resolution

- Now Slings tries to find the **resource** addressed
- Mappings are applied
- Registered *ResourceProviders* are asked for a resource with the resulting path
- Most of the time the default *JcrResourceProvider* will end up delivering the resource by looking it up in the backing repository
- But the resource might also be provided by anything mapped into the resource tree by a suitable *ResourceProvider*

# Resource Type

- To allow for different processing for different kinds of resources, Sling has the notion of a *resource type*
- The *resource type* determines the script that is used for processing the resource
- Declaration of the resource type with *sling:resourceType*
- Since resource types are treated as a path in the script selection process you should use path-like, slash delimited resource types like:

  *company/homepage*
- Fallback: converting of the primary node type of the resource to a path (by replacing the colons with slashes) and trying that one (e.g. *nt:file* becomes *nt/file)*

adaptTo()

# Resource Super Type

- A resource can not only define its resource type, but also a *resource super type* via the *sling:resourceSuperType* property
- If a resource super type is defined for a resource, script resolution will fallback to this super type before falling back to the default scripts
- If a resource type has no explicit resource super type, the default resource super type is assumed to be *sling/servlet/default*
- The resource super type can either be declared on the resource node itself or for its resource type

# Script Locations

- To determine the location of the script to process the resource, Sling converts the resource type to a path
- If this path is absolute (i.e. starts with a slash), it is used as-is.

  Resource Type: */apps/company/homepage*

  Search Locations: */apps/company/homepage*
- If the path is relative, the ResourceResolver prepends each of the configured search paths to it and searches the resulting paths (in the order they are configured)

  Path [ "*/apps*", "*/libs*" ]

  Resource Type: *company/homepage*

  Search Locations:
  - */apps/company/homepage*
  - */libs/company/homepage*

# Script Names

- Depending on whether request selectors are considered, a script name may have two forms:
  - Without selectors:

    *{resourceTypeLabel}.{requestExtension}.{requestMethod}.{scriptExtension}*
  - With selectors:

    *{selectorStringPath}.{requestExtension}.{requestMethod}.{scriptExtension}*

- *{resourceTypeLabel}* - Last path segment of the path created from the resource type. (optional).
- *{selectorStringPath}* - The selector string. (optional)
- *{requestExtension}* - The request extension (default html)
- *{requestMethod}* - The request method (e.g. “GET” or “POST”)
- *{scriptExtension}* - The extension, e.g. “esp” or “jsp”, identifying the scripting language.

# Script Selection Priority

- If multiple scripts apply for a given request, the script with the best match is selected. The more specific a match is, the better it is.

- The priority of the script name components are (from highest to lowest):
  1. Selectors (the more selector matches the better)
  2. Extension
  3. Resource type label
  4. HTTP Method

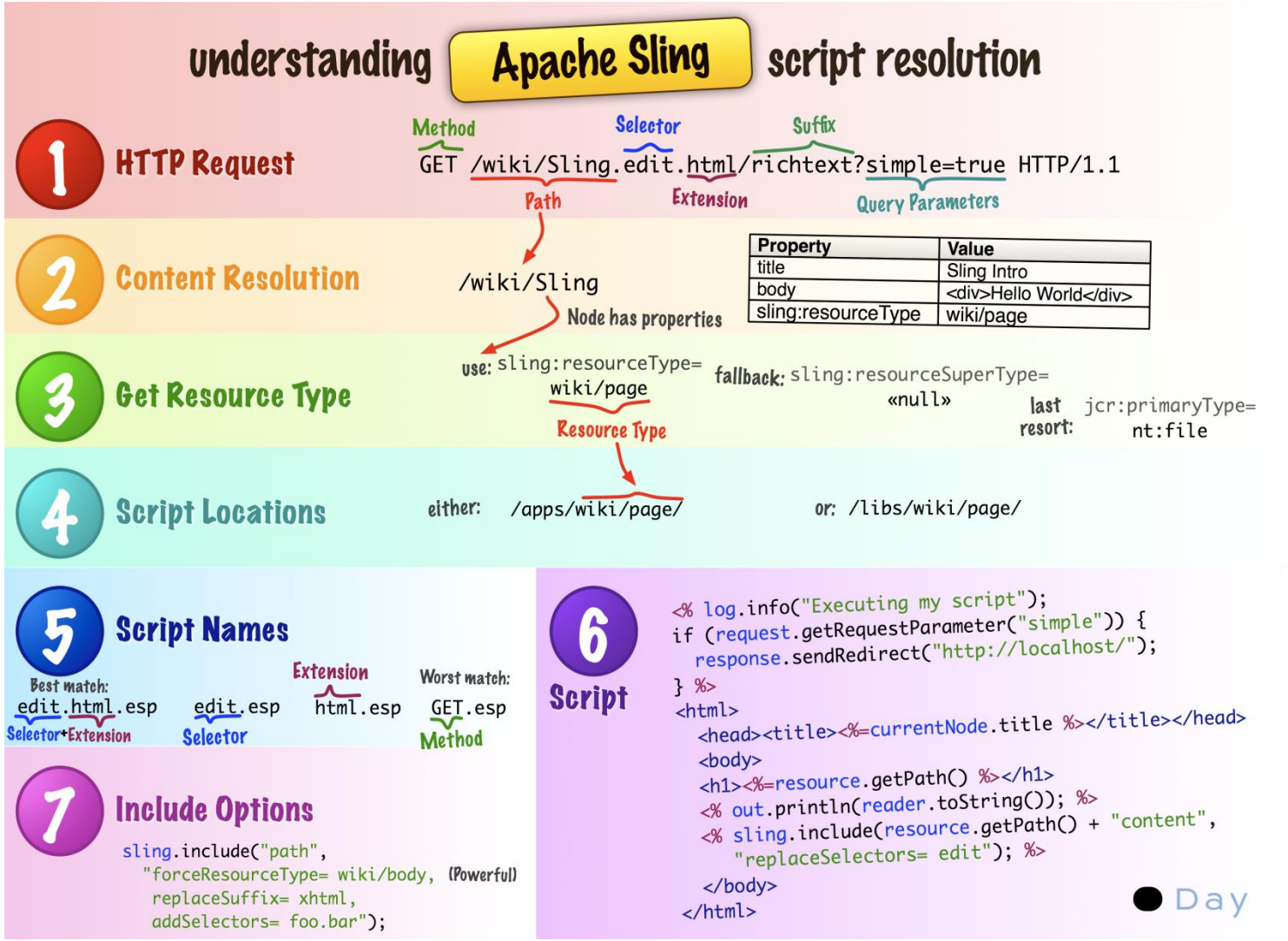
# Script Selection Priority Example

- Consider a request to access the resource
  */content/adaptto/sling/basics/stock/overview.beverage.beer.html*
  with the resource type
  *adaptto/sling/basics/stock*
  Assuming we have the following list of scripts in the correct location then the order of preference would be (7) - (6) - (5) - (4) - (3) - (2) - (1)
  1. GET.jsp
  2. stock.jsp
  3. html.jsp
  4. beverage.jsp
  5. beverage.html.jsp
  6. beverage/beer.jsp
  7. beverage/beer/html.jsp

# SlingRequestDispatcher

- Include or forward a request to another resource:
  - *forward()* is like a server-side *redirect* where control is completely handed over to another resource
  - *include()* is like a server-side *include* where the output of the targeted resource is included into the response
- *SlingRequestDispatcher* is the Sling-specific implementation of the Servlet *RequestDispatcher* interface which accepts an additional *RequestDispatcherOptions* parameter
- With these options you can:
  - Force the targeted resource to be processed with another resource type
  - Add selectors to the request or replace them altogether
  - Replace the suffix path

# Sling Cheat Sheet



# Scripting Options

- Sling supports scripting via JSR 223
- Out of the box Sling provides these scripting languages:
  - JSP
  - Groovy
  - JavaScript
- *DefaultSlingScript* acts as a facade for Servlets and Scripts

```
class DefaultSlingScript implements SlingScript, Servlet, ServletConfig {
...
}
```

# Scripting Options

- Some options that are available as contributions include:
  - Python
  - Ruby
  - Scala
  - Freemarker
  - Velocity
  - Java (as .java source files)
- See http://localhost:8080/system/console/config in the „Script Engines“ tab for available script engines

# Sling JSP Tag Library

- Sling includes a Tag Library to simplify JSP development
- Include it in your JSP with:
- @taglib prefix=*"sling"* *uri=http://sling.apache.org/taglibs/sling/1.0*
- *<sling:defineObjects>*
  - Exposes the regularly used scripting objects like:
    - request
    - response
    - resource
    - out
- *<sling:include>*
  - Includes a resource rendering into the current page

# Servlets

- Registration via OSGi HTTP Service
  - Bare-bones OSGi way of registering servlets, used by the SlingMainServlet
  - Servlets registered via the HTTP service bypass Sling processing

- Registration via Whiteboard
- *SlingServletResolver* listens for *Servlet* services
  - Servlet services must at least declare either *sling.servlet.path* or *sling.servlet.resourceType* property to be taken account by Sling
  - Sling registers the servlets as resources in the (virtual) resource tree
  - Servlet resources are adaptable to *javax.servlet.Servlet*

# Default Servlets

- *SlingMainServlet*
  - Entry point into the Sling request handling after authentication
- *DefaultGetServlet*
  - Provides default renderers for resources (HTML, TXT, JSON, XML)
- *SlingPostServlet*
  - Allows extensive content manipulation (create, delete, update) through its REST API
- *DefaultErrorHandlerServlet*
  - Default error handler servlet registered at the end of the global search path

# SlingSafeMethodServlet

- Sling contains the *SlingSafeMethodsServlet* base class as a helper for implementing read-only servlets

- When overwriting its *doXXX* methods, you don‘t need to cast the request and response objects to have access to the Sling-specific methods

# Use Cases of Different Bindings

- Resource Type
  - Primary and preferred way of binding a script to a resource.
  - Provides a separation between a resource and its representation.

- Path
  - Discouraged in Sling!
  - Bypasses access control
  - Use only for servlets which are  independently of the existence of any specific resource

# Use Cases of Different Bindings

- Selectors
  - Often used for a different representation of a resource (e.g. a detail view)
- Extension
  - Use this for alternative representations of a resource as another file type (PDF, PNG, JSON etc.)
- Method
  - You'll primarily use this to implement some specific data storage by binding a servlet to the POST method also.

# Filters in Sling

- Used for authentication, post-processing of markup etc.
- Filters in Sling are services implementing the *javax.servlet.Filter* interface
- Two service properties are relevant:
  - **sling.filter.scope** – Indicates the filter chain the filter should be part of. Required! Sling won't pick up your filter if it isn't set.
  - **service.ranking** – Indicates the priority of the filter in the chain. The higher the number the earlier the filter will be invoked.
- Current filter configuration can be inspected in the „Sling Servlet Filter" tab at http://localhost:8080/system/console/config

# The SlingPostServlet

- Handles all POST operations to a Sling instance by default
- Create, modify, copy, move, delete and import resources
- Can be extended with additional POST operations/PostProcessor
- SlingPostServlet is an OSGi bundle

- Sample:

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

This sample form will set the "*title*"-property on a node at "*/new/content*".

# The SlingPostServlet - Node Value Suffixes

- Provides instructions about how to process node properties
- They are typically on hidden input fields

Sample:

```
<form method="POST" action="http://host/new/content" enctype="multipart/form-data">
        <input type="text" name="title" value="" />
        <input type="hidden" name="title@DefaultValue" value="Some Value" />
        <input type="checkbox" name="checked" />
        <input type="hidden" name="checked@TypeHint" value="Boolean" />
        <input type="Submit" />
</form>
```

Indicates that the „title“ property should be set to „Some Value“, unless the value of the „title“ parameter is not blank.

Indicates that the value of the „checked“-Property should be set on the target Node as „Boolean“.

adaptTo()

# Agenda

- General
  - Architecture
- Request Handling
  - URL decomposition
  - Dispatching Requests
  - Servlets and Scripts
  - Filters
- **Resources**
  - **Resources**
  - **Resource Provider and Resource Resolver**
  - **Resource Resolution Mapping**
- Sling API
  - Adapters
  - Sling API

# Resources

What is a **Resource**?

- Extending from JCR's *Everything is Content*, Sling assumes that (almost) ***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.

adaptTo()

# JCR-based Resources

- Since Sling is built upon JCR, these (JCR-based) Resources are the default and you'll encounter them most often
- JCR-based Resources are provided with the default *JcrResourceProvider*
- The *JcrResourceProvider* is always available and is always asked last.
  - Resources provided by other Resource providers may never be overruled by repository based Resources.

# Bundle-based Resources

- Resources can be provided by files (*resourceType=nt:file*) and directories (*resourceType=nt:folder)* contained in an OSGi bundle
- The *BundleResourceProvider* is responsible for mapping the directory tree into the resource tree
- Your bundle has to declare the resources it provides with the header *Sling-Bundle-Resources*
- It‘s very convenient to add it as instruction in the maven-bundle-plugin configuration:

```
<configuration>
  <instructions>
       <Sling-Bundle-
  Resources>/resource/tree;path:=/bundle/tree</Sling-Bundle-Resources>
  </instructions>
</configuration>
```

# Filesystem Resources

- *FilesystemResourceProvider* can mount a directory from the local file system into the resource tree
- As with other similar providers files use the *nt:file* resource type, directories the *nt:folder* resource type
- *FilesystemResourceProvider* can be configured via the OSGi configuration console

# ResourceProvider

- Provides access to resources from different locations through a single ResourceResolver
- Registered to a path in the virtual resource tree, it‘s able to provide resources below that location
- ResourceResolver asks ResourceProviders in the order of longest match of the root path (e.g. R1 registered below */content/a* will have higher priority than R2 registered below */content*)
- JCR ResourceProvider is registered at the root **(/)** of the virtual resource tree
- Last fallback is always the JCR repository

# Resource Resolver

- Gateway for resources, abstracts the path resolution
- Abstracts access to the persistence layer(s)
- *resolve() :* central method for matching a path to a resource
- *map() :* inverse operation to *resolve(),* yielding an external path for a resource path
- Generated by *ResourceResolverFactory* based on current user credentials
- Available through *SlingHttpServletRequest.getResourceResolver()*
- Resource Enumeration through *listChildren()*

# JCR Resource Resolver

- Default ResourceResolver in Sling, resolves Resources through JCR
- Configuration options
  - Search paths (*/apps* and */libs* by default)
  - Mappings visible in the web console: http://localhost:8080/system/console/jcrresolver
  - Resolve Map - Maps URLs to Resources
  - Mapping Map - Maps Resource Paths to URLs
- The current JCR session can be obtained with

```
request.getResourceResolver().adaptTo(Session.class);
```

# Querying Resources

- The ResourceResolver provides two methods for querying Resources:
  - *ResourceResolver.findResources(query, language)*
  - *ResourceResolver.queryResources(query, language)*
- Those are convenience wrappers around the JCR Query functionality
- Only repository based resources are supported, as currently there‘s no way for ResourceProviders that are not repository based to support queries

# Resource Resolution Mappings

- Mappings of URLs to resources can be configured in a configuration tree below */etc/map*
- A path is constructed from the request using *{scheme}/{host.port}/{uri_path}*
- This path is mapped against the content structure under */etc/map* and the properties from the longest match are used for the mapping
- If the result of the mapping is an URL, it is mapped to a path again (until a path results)
- If a path is found the standard strategy of resolving resources is applied:
  - If the path is absolute, it is used as-is
  - If it is relative, the configured search paths are prepended and the first matching resource returned

# Mapping Properties and Node Types

- Important Properties:
  - *sling:match* – Allows using a regular expression for matching instead of the node name
  - *sling:redirect* – Send a redirect response to the client with the value of this property
  - *sling:internalRedirect* – Modifies the path internally for further resource resolution
  - *sling:alias* – Defines an alternative name for the node this property is set on

- Sling comes with node types to simplify creating this structure:
  - *sling:Mapping* – Primary node type
  - *sling:MappingSpec* – Mixin node type

# Mapping Example

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

# Mapping Example

| Regular Expression | Redirect | Internal | Description |
|---|---|---|---|
| http/www.example.com.80 | /example | yes | Prefixes the URI paths of the requests sent to this domain with the string */example* |
| http/.+\.example\.com\.80 | http://www.example.com | no | Redirects all requests to sub domains to *www*. The actual regular expression for the *host.port* segment is taken from the *sling:match* property. |
| http/localhost\.\d* | /content | yes | Prefix the URI paths with */content* for requests to *localhost*, regardless of the actual port the request was received on. This entry only applies if the URI path does not start with *stories* because this is a longer match entry. The actual regular expression for the *host.port* segment is taken from the *sling:match* property. |
| http/localhost\.\d*/(stories) | /anecdotes/stories | yes | Prepend the URI paths starting with */stories* with */anecdotes* for requests to *localhost*, regardless of actual port the request was received on. |

# Namespace Mangling

- Paths in Sling often contain colons as they‘re used for namespacing in the underlying JCR Repository
- Namespace mangling encloses the namespace prefix in underscores and removes the colon
- Only namespaces registered with JCR are processed to prevent mangling of paths that might contain underscores
- The mangling takes place in the *map()* methods (to take care of paths leaving the system)
- The unmangling is done in the *resolve()* methods (to handle incoming paths)
- Example:
  - /content/_a_sample/jcr:content/jcr:data.png
    - gets mangled to :
  - /content/_a_sample/_jcr_content/_jcr_data.png

# Agenda

- General
  - Architecture
- Request Handling
  - URL decomposition
  - Dispatching Requests
  - Servlets and Scripts
  - Filters
- Resources
  - Resources
  - Resource Provider and Resource Resolver
  - Resource Resolution Mapping
- **Sling API**
  - **Adapters**
  - **Sling API**

# Adapters

- **adaptTo()** (interface *Adaptable*) on Resource/ResourceResolver-Implementations
  - Resource → JCR Node
  - Resource → Value Map
  - Resource → CQ Page
  - Resource Resolver → JCR Session
  - Resource Resolver → CQ Page Manager
  - …

Code Sample:

```
ValueMap valueMap = getResource().adaptTo(ValueMap.class);
```

# Sling API Overview

- SlingHttpServletRequest and SlingHttpServletResponse
- RequestPathInfo
- Resource
- ResourceResolver
- ResourceUtil
- ValueMap and PersistableValueMap
- Adaptable

# (Sling)Request/Response, RequestPathInfo

- SlingHttpServletRequest
  - defines the interface to provide client request information to a servlet. It adds specific methods for the interaction with the Sling framework
- SlingHttpServletResponse
  - currently empty, merely exists to parallel SlingHttpServletRequest
- RequestPathInfo
  - provides information about the request URL

# Resource

- **Resource** extends **Adaptable**
  - Example:

```
Resource resource = resourceResolver.getResource(path);
Resource parentResource = resource.getParent();
Iterator<Resource> resourceIterator =
  parentResource.listChildren();

while (resourceIterator.hasNext()) {
  Resource childResource = resourceIterator.next();
  if(childResource.isResourceType("comp/book")) {
    // do something
  }
}
```

# ResourceResolver

- ResourceResolver extends Adaptable
  - defines two kinds of methods to access resources: The **resolve** methods and the **getResource** methods

| Method Kind | Access Algorithm |
|---|---|
| **resolve** | The path is always assumed to be absolute. Uses elaborate resource resolution algorithm. This kind of method is intended to resolve request URLs to resources. |
| **getResource** | Directly access resources if the path is absolute. For relative paths the search path is applied. This method is intended to be used by request processing scripts to access further resources as required. |

# ValueMap

- ***ValueMap*** extends ***Map*<String, Object>**
  - The *ValueMap* offers an easy way to access the properties of a resource.
  - With most resources you can use the **Resource.adaptTo** method to adapt the resource to a value map
  - Example:

```
ValueMap vMap = resource.adaptTo(ValueMap.class);
String str1 = vMap.get("propName" String.class);
String str2 = vMap.get("propName", "");
```

- ***PersistableValueMap*** extends ***ValueMap***
  - Extension of the *ValueMap* which allows to modify and persist the properties.


APACHE SLING & FRIENDS TECH MEETUP

BERLIN, 26-28 SEPTEMBER 2012

# Thank you for your attention!

Any questions?