REST... with Peace

Content Management with Apache Sling

The Problem (abstract)

- Store large amounts of different types of content
- Associate meta data to content
- Access control
- Search for stuff
- Get notified on changes
- ...and all that in a lightweight fashion

sounds familiar, huh? :-)

JCR: Some History

- The content management market was fragmented heavily: many vendors, many implementations
- Users were locked in to specific solutions
- Even the most fundamental concept of content management was not standardized: the content repository
- The Content Repository for Java API was proposed in JSR 170 by Day Software, finalized in 2005
- JCR 2.0 in 2009, API released as an OSGI bundle

JCR 1.0: http://jcp.org/en/jsr/detail?id=170

JCR 2.0: http://jcp.org/en/jsr/detail?id=283

JCR in a Nutshell

- A hierarchical model for storing structured and unstructured content in content repositories
- Conformance Level 1:
  - reading, nodes and properties, XPath queries, export
- Conformance Level 2:
  - write, references and referential integrity, access control, import
- Optional features:
  - versioning, transactions, SQL queries, locking, observation

Apache Jackrabbit

- Implements all features of JCR 1.0 and 2.0
- Actively developed, current version is 2.2.4

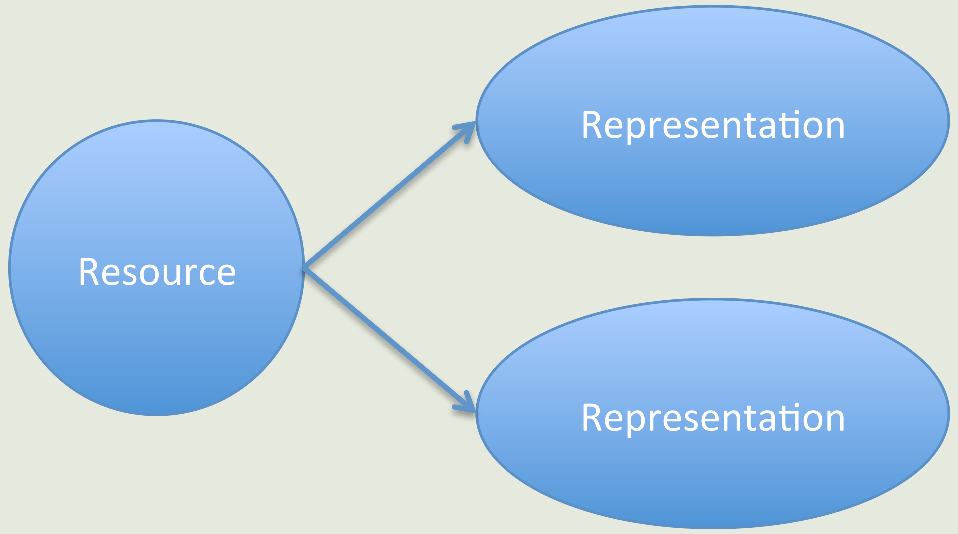
Apache Jackrabbit™

http://jackrabbit.apache.org/

- Uses a mix of database (Apache Derby) and filesystem persistence by default

a Bit of REST

- It‘s all about resources
- Focused on data, not actions (unlike SOAP, RPC)
- CRUD via HTTP
- Distinguishes between (abstract) resources and (concrete) representations
- Probably HTTP‘s most overlooked feature:
  content negotiation based on HTTP headers (e.g. Accept, Accept-Language)



Roy Fielding‘s thesis: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Apache Sling

- REST-based framework on top of Apache Jackrabbit
- Apache top-level project since 2009
- OSGI-driven (Apache Felix)
- Scripting (JSP, SS-JS, Scala), open to other languages
- Can be deployed standalone or inside a servlet container, Launchpad is ready-to-run
- Convention-over-configuration
- What it‘s *not*: a full-featured, „classic“ web application framework

http://sling.apache.org/site/index.html

The Content Model

- Everything evolves around resources, nodes and properties
- Sling allows you to store structured or unstructured content anywhere you want:

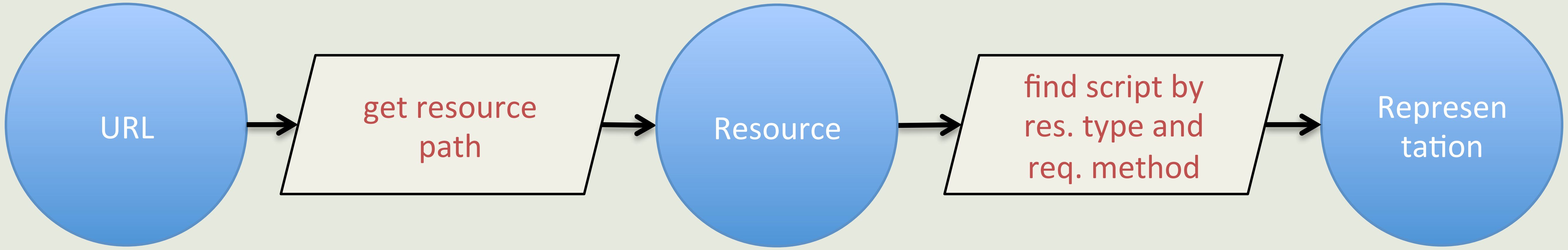
```
curl -F"foo=bar“ http://localhost:8080/repo
curl -F“foo=baz“ -F“sling=rocks“ http://localhost:8080/repo
curl -F“andAnotherThing=true“ http://localhost:8080/repo/
curl -X DELETE http://localhost:8080/repo
```

- Resource types give structure to content:

```
curl -F"sling:resourceType=sling:Folder" $SAURL/repo
```

How Does it Work?

- Data-centric, resource-first approach
- Simply put:



- There‘s more: wrapping and decorating resources

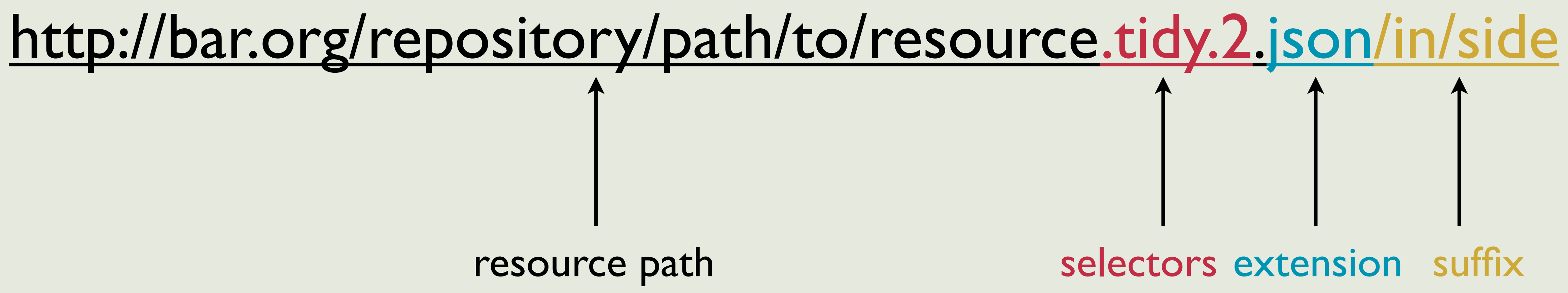
Defining Resource Types

- Two options for defining resource types:
  - Imperative: via Java
  - Declarative:
    - CND (Compact Node Definition)
    - XML, JSON (Sling)
- Allows for an exact definition of the content model
- Supports sub-classing, mixins

```
[wcmpp:resource]         > nt:folder, sling:Resource
    - title              (string) mandatory
    - state              (string) mandatory
    + changelog          (wcmpp:changelog) autocreated
    + tags               (wcmpp:tags) autocreated
    + content            (nt:file) version

[wcmpp:tags]             > nt:folder, sling:Resource
    + *                  (nt:base) = wcmpp:tag

[wcmpp:tag]              > nt:base, sling:Resource

[wcmpp:changelog]        > sling:OrderedFolder
    + item               (wcmpp:changelogentry)

[wcmpp:changelogentry]   > nt:folder, sling:Resource
    - text               (string) mandatory

[wcmpp:image]            > wcmpp:resource
```

URL Decomposition

- Sling has a smart method of decomposing URLs to find the right resource representation:


http://bar.org/repository/path/to/resource.tidy.2.json/in/side
resource path
selectors
extension
suffix

- URL decomposition is Sling‘s way of Content Negotiation

- Advantage: you don‘t need to mess around with HTTP headers

Servlets and Scripts

- Two methods of adding servlets and scripts: (well, truth being told, scripts are servlets as well)

  1. Register as OSGI services
     a. using SCR annotations →

     ```
     sling.servlet.*:
     paths, resourceTypes, selectors,
     extensions, methods, prefix
     ```

     ```
     * @scr.service interface="javax.servlet.Servlet"
     * @scr.property name="sling.servlet.resourceTypes" value="wcmpp/feed"
     * @scr.property name="sling.servlet.methods" value="GET"
     ```

     b. using code (not Sling-ful)

  2. Store in `/apps` →

     ```
     path component derived from resource type,
     e.g. a GET handling server-side JavaScript for
     resource type cms/images needs to be stored in
     /apps/cms/images/GET.esp
     ```

- Servlet registration is more fine-grained

Queries

- JCR supports XPath (deprecated with JCR 2.0) and SQL (JCR 2.0) queries
- Via Java:

```
Resource resource = request.getResource();
Session session = repository.login();
QueryManager qmngr = session.getWorkspace().getQueryManager();

Query query = qmngr.createQuery("//element(*, wcmpp:image)", "xpath");
NodeIterator result = query.execute().getNodes();
```

- Using the query servlet:

```
http://localhost:8080/repo.query.json?statement=//*[@jcr:primaryType='wcmpp:image']
&property=jcr:content/jcr:mimeType
```

  - Parameters: offset, rows, property

Access Control

- Privilege based access-control model, based on JCR
- Pre-defined privileges are applied to nodes using specific selectors:
  - modifyAce, acl, deleteAcl
- Privileges:
  - jcr:read, jcr:write, jcr:all
  - jcr:modifyProperties, jcr:addChildNodes, jcr:modifyAccessControl, jcr:nodeTypeManagement, ...

You need to register users before setting privileges:

```
curl -F:name=cag -Fpwd=password -FpwdConfirm=password -Fanyproperty=value
http://localhost:8080/system/userManager/user.create.html
```

```
curl -FprincipalId=cag -Fprivilege@jcr:read=granted http://localhost:8080/test/node.modifyAce.html
```

Listening to Events

- A repository sends low-level events:
  - TOPIC_RESOURCE_ADDED, TOPIC_RESOURCE_REMOVED, TOPIC_RESOURCE_CHANGED, TOPIC_RESOURCE_PROVIDER_ADDED
- You‘d typically filter these generic events and transform them into application specific events

```
 /**
  * @scr.component  immediate="true"
  * @scr.service interface="org.osgi.service.event.EventHandler"
  * @scr.property name="event.topics" valueRef="org.apache.sling.api.SlingConstants.TOPIC_RESOURCE_ADDED"
  */

public class MyHandler implements EventHandler {

   public void handleEvent(Event event) { }

}
```

```
String propPath = (String) event.getProperty(SlingConstants.PROPERTY_PATH);
String propResType = (String) event.getProperty
(SlingConstants.PROPERTY_RESOURCE_TYPE);
if (propPath.startsWith("/tmp/dropbox") && propResType.equals("nt:file")) {
   final Dictionary<String, Object> props = new Hashtable<String, Object>();
   props.put(EventUtil.PROPERTY_JOB_TOPIC, JOB_TOPIC);
   props.put("resourcePath", propPath);
   Event dropboxJobEvent = new Event(EventUtil.TOPIC_JOB, props);
   eventAdmin.sendEvent(dropboxJobEvent);
}
```

<eop>

Thank you for your attention.
Questions?

Claus Augusti <claus@formatvorlage.de>
Frontend Architect 1&1 Development Hosting

Kudos to the Apache Foundation. You guys *ROCK* :-)