RFP-171 Web Resources

Draft

9 Pages

Abstract

Web Applications require more and more access to web resources like Javascript files. These web resources can be served from content delivery network over the Internet but this has implications for HTTP/HTTPS access, security and availability (especially in the future). Therefore, applications serve these web resources from their own server. The OSGi environment is a perfect environment to handle these web resources. This RFP seeks a proposal for how to package Web Resources in an OSGi.

0 Document Information

0.1 Table of Contents

0.2 Terminology and Document Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in 1.

```
Source code is shown in this typeface.
```

0.3 Revision History

The last named individual in this history is currently responsible for this document.

Revision	Date	Comments
Initial	*03-12-14*	*Initial* *Peter.Kriens@aQute.biz*

1 Introduction

Web Applications require more and more access to web resources like Javascript files. These web resources can be served from content delivery network over the Internet but this has implications for HTTP/HTTPS access, security and availability (especially in the far future). Therefore, applications often serve these web resources from their own server. The OSGi environment is a perfect environment to handle these web resources since it has a very clear life cycle. This RFP seeks a proposal for how to package Web Resources in an OSGi system.

This RFP originates from the OSGi enRoute work. In this project, a number of services were identified, designed and implemented based on their needs for web based applications. This document analyzes the application domain and defines the problem that need to be solved.

2 Application Domain

A w*eb resource* is a an artifact that must be available from an HTTP(S) server so that other web resources and/or dynamic content can refer to it via a URL. A web resource is static, it does not require any processing before it is delivered. For example, the Angular JS Javascript library contains of a coupled of hundred files that need to be available to the HTML page that contains the application.

2.1 CDNs

The preferred mode to serve static resources is over a Content Delivery Network. A CDN provides high bandwidth and since the same URL is shared between many applications the browser has a higher chance of finding the web resource in its cache.

However, in practice CDN are not always a correct solution.

- Many applications have a governing process that requires that any web resources are locally served so their content in runtime can be verified.
- HTTPS served pages require their resources to be served over HTTPS as well, which are not always available in the CDN.
- Though many CDNs will be around forever, it is not clear that over long term all the content will be available, potentially killing an application in the future.
- A CDN requires network connectivity to the Internet. Malfunctions and closed intranets make them sometimes an impossibility.

For these reasons, *web applications* frequently need to serve web resources from their own servers. Since these resources do not have to be dynamically generated they are usually served separately by web servers that are highly optimized and provide extensive caching support for static resources.

However, in Java it is also popular to deliver web resources from the JAR files. The WAR format specifies that any content in the `META-INF/resource` directory in a JAR in the WAR files `lib` directory is served as a web resource, the remaining part of the path of the resource is then mapped to the root of the web server. For example, the web resource `META-INF/resource/angular/1.3.1/angular.js` will be available as `/angular/1.3.1/angular.js`. This model is described in [3].

2.2 Paths

The paths to the web resources are generally unmanaged. The URLs to the web resources are used in many different places in a typical project. Managing these paths can be quite a challenge. These paths are also shared between other applications that run on the same web server. To prevent, the often short names, of the web resources to clash with other applications, some application servers can rewrite URLs so that each application gets its own namespace.

A simple solution would be to use long path names but this is not user friendly, there is a desire to have short URLs.

2.3 Caching

The HTTP(S) protocol has extensive support for caching. Headers in the protocol provide information to proxies and the browser of what can and what can not be cached and how long. Over time a large number of best practices have been evolved that are non-trivial to manage.

Caching is crucial for good performance since javascript applications have been growing up over the last few years. Downloading a megabyte in compressed Javascript is no exception anymore. The average page in 2014 includes about 300k of Javascript across 18 different files (HTTP Archive [7].) This number is likely to increase.

Debugging must take caching into account. With a development IDE like bndtools the edit-build-debug cycle is very short that it becomes fully interactive. However, caches can slow this down if they need to be cleared for each cycle.

2.4 Merging

A solution that is becoming more popular is merging all the Javascript sources in a single resource. This is much more efficient than the dependency manager in the browser going back and forth and traversing the transitive dependency tree. Merging can then also include Javascript files from *application plugins*. Since these plugins are then loaded they get control and register themselves with the application or a module system like Angular.

During the debug cycle, there is often also the desire to use Javascript files that are not *minified*.

2.5 Dependencies

Most Javascript libraries use `package.json` files defined in [5]. These files are similar to pom.xml in maven. They describe the library and the names of the their dependencies. There are alternatives with bower [6]. and others. Most dependency managers work by searching a number of directories defined by a path.

Through these dependency managers it is possible to find the closure of Javascript web resources that should work together.

2.6 Wrapping

Wrapping web resources in a bundle creates a dependency on the development life cycle of the web resources. For example, if a new Angular JS is released then developers need to quickly have access to this web resource. However, in the WAR model a new lib JAR must be created with the content. Since each developer that uses Angular has this problem a group effort would be beneficial. This is the same problem as OSGi has with bundle wrapping. In this case, a group created webjars.org [4]. In this project, they allow people to define a Github project that pushes a JAR to maven central that stores the web resources in `META-INF/resource/<name>/<version>`. The webjars.org site then creates an index on maven central. This project is quite popular and contains hundreds of webjars.

In the webjar.org model dependencies are handled through maven.

2.7 OSGi enRoute

In OSGi enRoute a “web resource server” servlet was developed to handle web resources. Initially the `static` directory in the bundle was mapped to the server but now also the `META-INF/resource` directory is supported. The web resource server would then handle searching and overlapping directories.

Since OSGi had the requirement-capability model, OSGi enRoute also developed a `osgi.enroute.webresource` namespace. This namespace defined the name of the capability to be the web resource path. Bundles providing this path would then add a capability with this name:

```
Provide-Capability: osgi.enroute.webresource; \
  osgi.enroute.webresource='/google/angular/1.3'; \
  version='1.3.1'
```

Using the Manifest Annotations in bnd, it was then possible to require a version of Angular in the application code.

2.8 Terminology + Abbreviations

3 Problem Description

Managing web resources is an important aspect of any web application. Handling these web resources developers encounter the following problems:

- Managing the dependencies of the Javascript resources is error prone and hard to do. It is not possible to use the OSGi Require-Capability model to create a closure of web resources
- There is no way to map OSGi bundled web resources to a web path
- Managing the web paths for the resources is cumbersome, especially if each resource needs its own versioned path
- Difficult to manage plugins of an application that require additional web resources.
- There is no easy way to use the results of the webjars.org project

Therefore, this RFP seeks a specification that defines how web resources can be stored in bundles while taking advantage of the Require-Capability model.

4 Use Cases

4.1 Management Application

Al Bundle has to make a management application for a successful startup. He chooses Angular for the browser and OSGi for the server. Al wraps the angular Javascript in a bundle, storing the resources in `META-INF/resource/google/angular/1.3.1/`. He adds a capability:

```
Provide-Capability: osgi.webresource; \
  osgi.webresource=/google/angular; \
  version:Version=1.3.1
```

He does a similar wrapping for d3.js. His HTML index.html page now looks like:

```
<html>
 …

  <script src="/osgi.webresources/com.example.mngmt-1.2.3.js">
</html>
```

Al uses a macro in bnd to set the bsn and version. Then Al adds the following requirements to the manifest (of course he actually uses the new bnd annotations for this):

```
Require-Capability: \
  osgi.webresource; \
    filter:="(&(osgi.webresource=/google/angular)(version>=1.3.1))"; \
    resources:List<String>="angular.js,angular-aria.js"; \
    priority=100000,
  osgi.webresource; \
    filter:="(&(osgi.webresource=/mbostock/d3)(version>=3.4.13))"; \
    resources:List<String>="d3.v3.min.js"
```

When this page is loaded, the script URL is interpreted by a servlet. It extracts the BSN and version and locates the bundle.

All requirements to web resources are then inspected, the specified requirements are now found. They were wired by the framework to the angular and d3 capabilities. The requirements are then sorted by their priority (descending) so I is now angular and then d3.

The first time, a new file is created for the `/osgi.webresources/com.example.mngmt-1.2.3.js` path.

They requirements are then sorted by priority. For each requirement, the wired capability is found. The capability's bundle is then designated to append the resources specified in the requirements (`angular.js` and `angular-aria.js`) from the found bundle. A similar process takes place for the d3 requirement.

The file is then placed in a cache that is cleared when the bundle is modified (which in general should change the URL), the cache is cleared.

5 Requirements

5.1 General

- G0010 – Provide a mechanism to wrap web resources as bundles
- G0020 – Define how web resources are mapped to the web path
- G0030 – Define how web resource bundles can be used with the Require-Capability model.
- G0040 – Allow the use of Javascript dependency managers like npm and/or bower to control the Requirements and Capabilities for automatic creation of wrapped bundles.
- G0050 – Provide a web path scheme to make sure names do not clash

5.2 Merged Resources

- M0010 – Provide a way to have a bundle dependent URL that merges all javascript resources needed for an HTML page
- M0020 – Allow priorities, e.g. Angular needs before any angular modules
- M0030 – Allow application plugins to be delivered as bundles.
- M0040 – Define a debug mode where web resources are not cached
- M0050 – Define a debug mode where it is possible to switch the minified sources for the original sources, if they are available.

6 Document Support

6.1 References

[1]. Bradner, S., Key words for use in RFCs to Indicate Requirement Levels, RFC2119, March 1997.

[2]. Software Requirements & Specifications. Michael Jackson. ISBN 0-201-87712-0

[3]. https://alexismp.wordpress.com/2010/04/28/web-inflib-jarmeta-infresources/
[4]. http://www.webjars.org/
[5]. https://www.npmjs.org/doc/files/package.json.html

[6]. http://bower.io/
[7]. http://httparchive.org/

6.2 Author's Address

Name	Peter Kriens
Company	aQute
Address	9c, Avenue St. Drezery
Voice	\+33467542167
e-mail	Peter.kriens@aQute.biz

6.3 End of Document