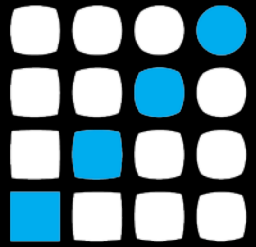

APACHE SLING & FRIENDS TECH MEETUP
10-12 SEPTEMBER 2018

Java9 and OSGi R7 with Apache Felix and Sling

Carsten Ziegeler, Karl Pauls – Adobe

Who are we?

Carsten Ziegeler

- Principal Scientist @ Adobe
- Member of the Apache Software Foundation
- PMC Member of Apache Felix and Sling
- OSGi Expert Groups and Board member

adaptTo()

Karl Pauls

- Computer Scientist @ Adobe
- Member of the Apache Software Foundation
- PMC Member of Apache Felix and Sling (VP Apache Felix)
- Co-Author OSGi in Action

Outline

- Java 9 support
  - Java 9 and OSGi R7
  - Apache Felix
  - Apache Sling
  - The Road ahead
- What else is new in OSGi R7
  - OSGi R7 Highlights
  - OSGi R7 and beyond…

Java 9 support

JPMS – Java Platform Module System

- Modularized JDK
  - 24 modules (e.g., logging, xml, desktop, rmi,…)
  - 6 modules deprecated for removal
    - java.activation, java.corba, java.transaction, java.xml.bind, java.xml.ws, java.xml.annotation
    - Not available by default (needs –add-modules)
  - Deprecation of Unsave

JPMS – Java Platform Module System

- Module system for jvm based applications
  - Modulepath along side classpath
  - Meta-data for exports, requires, and services (module-info.java)
  - Module level accessibility
    - Public no longer public (only public and exported and readable is accessible)
    - Includes reflection
  - No split packages
- ModuleLayer for recursive use cases
- Allows developers to build custom platforms based only on the required modules (via jlink tool)

Multi-Release JAR

- New type of JAR called multi-release JAR
  - Allows the JAR to support multiple major Java versions
- In a nutshell
  - Simple JAR with „Multi-Release: true“ in Manifest
  - Can provide version dependent resources in
    META-INF/versions/N (for N>=9)
  - Highest matching versioned resource overrides

Java 9 and OSGi R7

Java imports

- Until now, osgi.ee was used to define required Java version
- Modularized Java enables to build custom platforms
  - Includes java.* packages
- Subsequently,
  - OSGi R7 now allows imports for java.*
  - osgi.ee should only be used for bytecode level
- Java exports still only possible by the system bundle
  - Effectively, still bootdelegated

Multi-Release JAR files in OSGi

- OSGi R7 adds support for multi-release JAR files
  - An OSGi bundle file can be a multi-release JAR
  - Bundle class path entries can be multi-release JAR files
- R7 Framework supports supplemental manifest files
  - Supplement “Import-Package“ and “Require-Capability“ for different versions
  - Via OSGI-INF/MANIFEST.MF in the versioned directories

e.g.:

META-INF/versions/9/OSGI-INF/MANIFEST.MF

Supporting R6

- OSGi R6 prohibits bundles from importing java.*
- Bundles that must work on OSGi R6 and earlier should:
  - Not import the java.* packages in the main Manifest
  - Package the bundle as a multi-release JAR and import java.* packages in supplemental manifests
- R6 frameworks will ignore supplemental manifests
- R7 frameworks will use them and they are only relevant starting with java >= 9

Apache Felix

adaptTo()

Runtime Discovery of JPMS packages

- Starting with Felix 6.0.0 system package exports are calculated dynamically on java >= 9
  - Optionally, uses constraints are calculated as well
    - Adds ~1 sec to first start-up time (results are cached)
- Module list can be specified and only available modules will be considered
- Exported packages can be overridden on a per module basis

Platform packages

- Framework property: felix.systempackages.calculate.uses=true
  - Turn on uses constraint calculation for system packages
- Framework property: felix.systempackages.substitution=true
  - Enable property substitution in org.osgi.framework.system.packages org.osgi.framework.system.packages.extra
  - ${felix.jpms.<module-name>} property added per detected module containing exported packages (with leading comma)
    - Custom definitions via override through framework properties

Platform packages example

Override packages provided by java.sql:

```
-Dfelix.jpms.java.sql=",javax.sql;version=\"0.0.0.9_JavaSE\";uses:=\"javax.transaction.xa\",javax.transaction.xa;version=\"1.1.0\""
```

Use only java.base and and java.sql (if it is there):

```
-Dorg.osgi.framework.system.packages="org.osgi.framework;version=\"1.9\"${felix.jpms.java.base}${felix.jpms.java.sql}"
```

Give packages for java < 9 and for JPMS based jdks:

```
-Dorg.osgi.framework.system.packages="org.osgi.framework;version=\"1.9\"
  ${sling.jre-${java.specification.version}}${sling.jre-${felix.detect.jpms}}
```

Apache Sling

JPMS packages via provisioning

- Starting with launchpad.base 6.0.0 we can use the provisioning model to specify available exports
  - Uses constraints calculation is on by default
  - Property substitution is on by default
  - Without any override of framework packages Felix defaults are used (i.e., all available packages)
- Sling 11 SNAPSHOT uses same set of exports for all Java versions
  - Missing packages added via bundles

Demo

https://github.com/karlpauls/adaptto-r7

The Road ahead

Java 10,11,...

- Java 10 essentially the same as Java 9
  - Apache Felix calculates osgi.ee dynamically
- Java 11 removes deprecated modules
  - Most can be replaced by bundles or added to the modulepath/image
- Java LTS Releases: 11 and then every 3 years
- Non-LTS Releases: 9, 10, 12...(every 6 months)
- Java 12 might make all available modules loaded by default (possibly this even happens for 11)
  - Source for LTS/Non-LTS: oracle.com

Tooling

- Maven has no support for Multi-Release JAR
  - Workarounds possible
- BND doesn‘t support Multi-Release JAR
  - https://github.com/bndtools/bnd/issues/2227
- BND doesn‘t support java.* dependencies
  - https://github.com/bndtools/bnd/issues/2507

adaptTo()

Summary and Outlook

- True interoperability between JPMS and OSGi still not possible as OSGi framework has to be on the classpath for now (and not on the module classpath)
- OSGi R7 improves using OSGi on JPMS
  - Runtime discovery of packages together with java.* imports allows developers to build custom runtimes
  - Multi-release JAR supports provides path for R6 BC
- Reflective access restrictions still a challenge for frameworks
- Possibility to maybe create custom runtimes based on provisioning/feature model in the future

What else is new in OSGi R7

adaptTo()

OSGi R7 Highlights

- Bundle Annotations
  - *All* manifest entries through annotations
- Package Exports and Versioning
  - @Version, @ProviderType, @ConsumerType
  - **@Export**

Bundle Annotations

@Capability, @Requirement, @Header

```
@Requirement(namespace="osgi.implementation",
             name="osgi.http",
             version="1.1.0")
@Header(name=Constants.BUNDLE_CATEGORY,
        value="adaptto")
```

Infer Requirements from Feature Usage

- When using Declarative Services:

```
@Requirement(namespace = ExtenderNamespace.EXTENDER_NAMESPACE,
     name = ComponentConstants.COMPONENT_CAPABILITY_NAME,
     version = ComponentConstants.COMPONENT_SPECIFICATION_VERSION)
```

- Infered by using DS annotations

```
@Component
```

Infer Requirements from Feature Usage

```
@Requirement(namespace = ExtenderNamespace.EXTENDER_NAMESPACE,
  name = ComponentConstants.COMPONENT_CAPABILITY_NAME,
  version = ComponentConstants.COMPONENT_SPECIFICATION_VERSION)
public @interface RequireServiceComponentRuntime {


@RequireServiceComponentRuntime
public @interface Component {
   ...
}
```

adaptTo()

Declarative Services Highlights

- Improved activation
  - Activation objects assigned to fields
  - Constructor injection
- Component Property Type annotations

Declarative Services – Field Activation Objects

```
@Activate
private MyConfiguration cfg;

@Activate
private BundleContext bundleContext;
```

Declarative Services – Constructor Injection

```
public GameServlet(@Reference GameController ref,
          MyConfiguration myCfg) {
...
}
```

Component Property Type Annotations

- Simplify Component Configuration

```
@ComponentPropertyType
public @interface ServiceDescription {

    String value();
}
```

adaptTo()

OSGi R7 Highlights - Basics

- LogService
  - Logger interface similar to slf4j
- Converter
  - Object conversion
- Push Streams and Promises
  - Asynchronous programming model
  - Streams

Using the Converter

```
Converter c = Converters.standardConverter();

// Convert scalars
int i = c.convert("123").to(int.class);

UUID id=c.convert("067e6162-3b6f-4ae2-a171-2470b63dff00")
         .to(UUID.class);

List<String> ls = Arrays.asList("978", "142", "-99");
short[] la = c.convert(ls).to(short[].class);
```

Convert into typed data

```
@interface MyAnnotation {
      int refresh() default 500;
      String temp() default "/tmp";
}

Map<String, String> myMap = new HashMap<>();
myMap.put("refresh", "750");
myMap.put("other", "hello");

MyAnnotation myAnn = c.convert(myMap).to(MyAnnotation.class);

int refresh = myAnn.refresh();  // 750
String temp = myAnn.temp();     // "/tmp"
```

adaptTo()

OSGi R7 Highlights - Web

- Http Whiteboard
  - Improvements (Global Filters)
  - Component Property Types
- JAX-RS
  - A whiteboard model for JAX-RS

adaptTo()

Http Whiteboard Annotations

```
@Component(service = Servlet.class)

@ServiceRanking(200)
@ServiceDescription("Best Servlet in the World")

@HttpWhiteboardServletPattern("/game")
@HttpWhiteboardContextSelect(
  "(" + HttpWhiteboardConstants.HTTP_WHITEBOARD_CONTEXT_NAME
  +"=" + AppServletContext.NAME + ")")
public class GameServlet extends HttpServlet {
```

JAX-RS with DS

```
@Component(service = TestService.class)
@JaxrsResource
@Path("service")
public class TestService {

    @Reference
    private GameController game;

    @GET  @Produces("text/plain")
    public String getHelloWorld() {
        return "Hello World";
    }
}
```

adaptTo()

JAX-RS Support

- Get, Post, Delete with Parameters
- Application support
- JAX-RS extension support (Filters, Interceptors, etc)
- Annotations for Declarative Services

OSGi R7 Highlights

- Configurator and Configuration Admin
  - Configuration Resources
  - Improved factory configuration handling
  - Configuration Plugin improvements

Configuration Resource

```
configurations: {
  "my.special.component" : {
    "some_prop:Integer": 42,
    "and_another": "some string"
  },
  "and.a.factory.cmp~foo" : {
     ...
  }
}
```

OSGi R7 Highlights

- Cluster Information
  - Support for using OSGi frameworks in clustered environments.
- Transaction Control
  - An OSGi model for transaction life cycle management.

OSGi R7 and beyond...

- Upcoming OSGi R7 Enterprise release
  - CDI - Context and Dependency Injection support OSGi
- R8 Plans
  - App Packaging and Java 11 JPMS
  - Realtime OSGi
  - Industry 4.0
  - Microprofile I/O

Questions?