GlassFish V3

Jerome Dochez

Sun Microsystems, Inc.
hk2.dev.java.net, glassfish.dev.java.net

Goal of Your Talk

What Your Audience Will Gain

Learn how the GlassFish V3 groundbreaking architecture is based on IoC, modules and maven 2.

Agenda

Demo !

Modules Subsystem

Build System

Services, services

Inversion of Control

Components, scopes

DEMO

Agenda

Demo !

Modules Subsystem

Build System

Services, services

Inversion of Control

Components, scopes

Module Subsystem : HK2

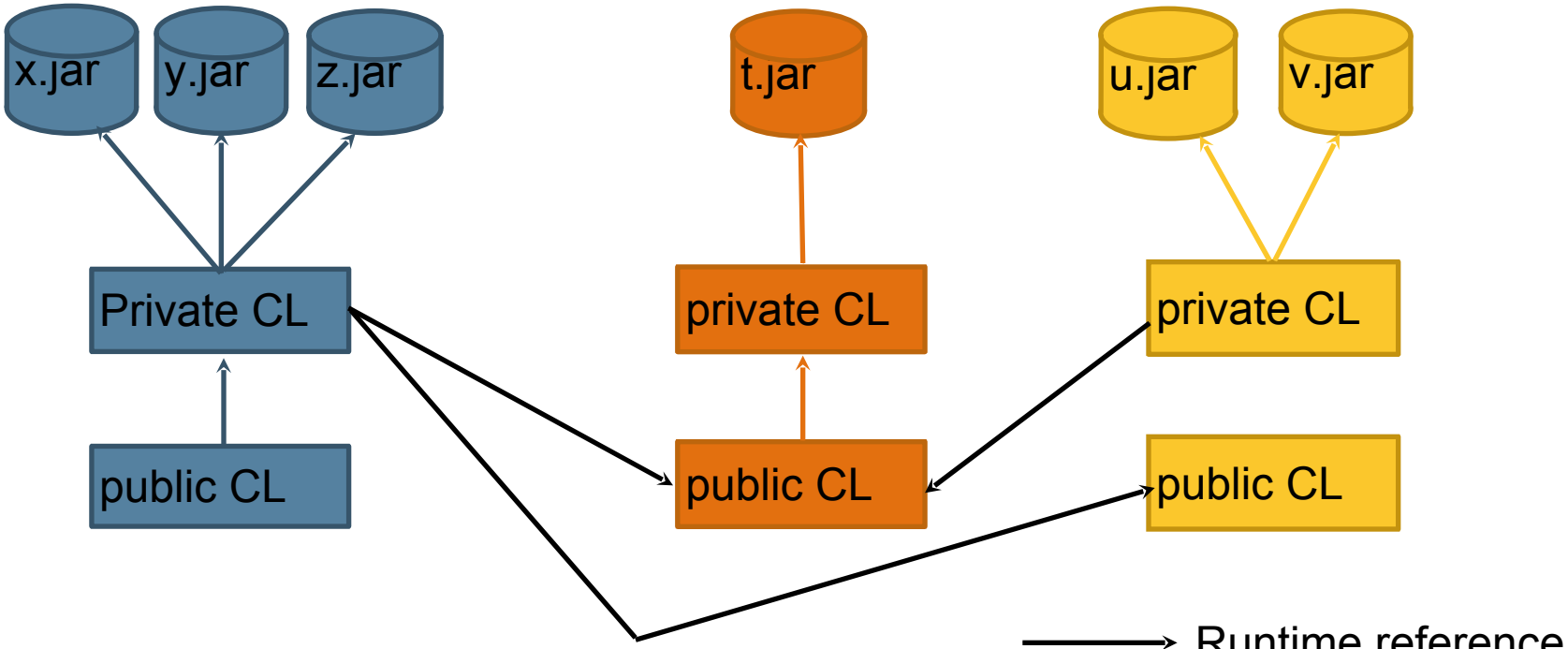
Introduction

- Loosely based on the work of JSR 277
- Due in Java SE 7
- Expert group still evolving the APIs
- Added hooks to provide extensibility points for other module types :
- maven
- OSGi
- Fits in 50 Kb : Hundred Kilobytes Kernel
- Runs on Java SE 5.

Module Instances

- At runtime, modules are identified by Module instances.
- Each Module has 2 ClassLoaders
- public that users have access to (facade)
- private that load all the module's classes
- Modules have a list of other module's class loaders to load imported classes.
- Garbage collection happens when all references to the public class loader are released.

Runtime network of class loaders



Module Definitions

Name : A
Imports: B, C

Name : B
Imports:

Name : C
Imports: B

Repository

- Repositories hold modules
- Can be added and removed at run time
- Different types supported
  - directory based
  - maven
  - OSGi ?
- Modules can be added/updated/removed from repositories

Bootstrapping

- Module subsystem can bootstrap itself
- No more classpath at invocation
- Application startup code is packaged in a jar file.
- Application code only need to implement the ApplicationStartup interface.
- Application code can declare dependencies in its manifest code.
- to run : java -jar
- For GlassFish : java -jar glassfish.jar

Build system : maven 2

- Each module is build from a maven project (pom.xml)
- pom.xml describes the module's
  - name
  - version
  - dependencies
- manifest entries are created automatically from pom.xml info
- pom.xml not used directly for performance reasons.

Module Example

Declare your module like :

```
    <groupId>com.sun.enterprise.glassfish</groupId>
    <artifactId>gf-web-connector</artifactId>
    <packaging>modsys-jar</packaging>
```

and dependencies with :

```
<dependencies>
    <dependency>
        <groupId>com.sun.enterprise.glassfish</groupId>
        <artifactId>webtier</artifactId>
        <version>${project.version}</version>
    </dependency>
....
```

Resulting definition

```
Jar File Manifest file :

Built-By : dochez
Created-By : Apache Maven
Implementation-Title : gf-web-connector
Manifest-Version : 1.0
Extension-Name : gf-web-connector
Implementation-Version : 10.0-SNAPSHOT
Import-Bundles : com.sun.enterprise.glassfish:webtier,
com.sun.enterprise.glassfish:v3-core
```

Build Repositories

- HK2 repository has been implemented using a maven repository backend.
- Build system puts modules in the maven repository.
- Running GlassFish gets the modules from the maven repository
- Once we got passed the maven bugs and quirks, build got a lot simpler than in V1/V2 leading to developer productivity.

Agenda

Demo !

Modules Subsystem

Build System

Services, services

Inversion of Control

Components, scopes

Services, services

- GlassFish V3 use extensively Services to identify extension points like :
  - Application Containers (like Web-App, Phobos, JRuby...)
  - Administrative Commands
- Services are :
  - implementing an interface
  - declared with META-INF/services file
- Can be stateless or statefull

Services in V3

- Interfaces are declared with @Contract
- Implementations are declared with @Service
- Build system will generate META-INF/services file automatically

```
@Contract
public interface Startup {...}

@Service
public class ConfigService implements Startup
{
...
}
```

@Service definition

```
public @interface Service {

    String name() default "";

    Class<? extends Scope> scope() default PerLookup.class;

    Class<? extends Factory> factory() default Factory.class;

}
```

Example :

```
@Contract
public interface AdminCommand {...}

@Service(name=”deploy”)
public class DeployCommand implements AdminCommand {
    ...
}
```

Current @Contract

- Startup : code to run at server startup
- Sniffer : code to identify deployable artifacts
- Deployer : code to deploy artifacts in a container
- AdminCommand : administrative commands
- Adapter : Grizzly adapter to receive web requests
- WebRequestHandler : adapter to service particular URL web requests.

Agenda

Demo !

Modules Subsystem

Build System

Services, services

Inversion of Control

Components, scopes

Dependency Injection

- @Inject to declare a dependency
  - On any @Service annotated class
  - Field :

```
@Inject
ConfigService config;
```

  - Setter method :

```
@Inject
public void set(ConfigService svc) {..}
```

- Use ComponentManager to retrieve services instances :
  - `public <T> T getComponent(Class<T> providerClass)`
  - `public Iterable<T> getComponents(Class<T> contract)`

Extraction

- All @Service annotated classes are extracted and available using an @Inject annotation.
- @Extract to declare extra values extraction
  - On any @Service annotated class
  - Field :

```
@Extract
ConfigService config;
```

  - Getter method :

```
@Extract
public ConfigService getConfigService() {..}
```

@Service life-cycle methods

- PostContruct interface
  - one method : postConstruct()
  - called after injection is performed and before it is made publicly available
- PreDestroy interface
  - one method : preDestroy()
  - called after the service is removed from public access.
- Available to all @Service annotated class
- Handled by the HK2 Runtime.

Components Instantiation stages

- Components Creation
  - new()
  - injection of all @Inject annotated resources
  - postConstruct()
  - extraction of all @Extract annotated resources
  - extraction of the instance
- Components Destruction
  - removed from public
  - all @Extract annotated resources removed from public
  - preDestroy() called

Instantiation cascading

```
@Contract
public interface Startup {...}

Iterable<Startup> startups;
startups = componentMgr.getComponents(Startup.class);

DeploymentService:java
@Service
public class DeploymentService implements Startup {

@Inject
ConfigService config;
}

ConfigService.java:
@Service
public Class ConfigService implements ... {...}
```

Injection of that resource

will trigger instantiation of the service impl

Components Scopes

- Components have scopes.

```
@Service(Scope=Singleton.class)
public class ConfigService implements Startup {...}
```

- Scopes are components...
- therefore extensible

```
@Service
public MyScope implements Scope {...}
```

- Scopes defines the boundaries of components visibility.

Agenda

Demo !

Modules Subsystem

Build System

Services, services

Inversion of Control

Container life-cycle

Application container life-cycle startup

- Each container ship with a connector module
  - containing at least one Sniffer

```
@Contract
public interface Sniffer {

    public boolean handles(File location);
    public String getModuleType();
    public void setup(String containerHome,
             Logger logger) throws IOException;
    public void tearDown();
 }
```

  - Each sniffer gets called on deployment request
  - handles() return true when they recognize a module type

Application container life-cycle startup

- Once a Sniffer is selected :
  - Sniffer::setup() is responsible for the container's installation (eventually from the internet).
  - Sniffer::setup() is also adding HK2 Repositories to the module subsystems.
  - Deployer service is looked up from the new Repositories with the right module type (obtained from Sniffer::getModuleType()).
  - Deployer service is invoked.

Application container life-cycle shutdown

- When last application is undeployed
- Sniffer:tearDown() will be called :
  - should remove any repositories added to the module system.
  - must return in a state where setup() can be called successfully
- Glassfish v3 will release all references to the container's runtime.
- Container should be garbage collected.

Application Server startup

- GlassFish v3 startup implemented by Startup interfaces.
- AppServerStartup.java is a component itself

```
@Inject
ComponentManager cm;

...
Iterable<Startup> startupsvcs =
        cm.getComponents(Startup.class);
```

GlassFish shutdown

```
@Service(name="stop-domain")
public class StopDomainCommand
    implements AdminCommand, PostConstruct{

@Inject
Startup[] startupSvcs;

@Inject
ComponentManager cm;

public void postConstruct() {

     cm.removeComponents(startupSvcs);
}
}
```

Summary : GlassFish V3

- Decomposition of the Java EE application server implementation
- Easy to embed all types of container that run on the JVM
- Embeddable
- Based on module subsystem (HK2)
- Use innovative and reusable components technology
- Available today in preview

For More Information

Links

- http://hk2.dev.java.net/
- http://glassfish.dev.java.net/
- http://wiki.glassfish.java.net/

Emails

- jerome.dochez@sun.com
- kohsuke.kawaguchi@sun.com

Instructions:
(Delete this red box before submitting your slides)

Use this slide to mark the beginning of the Question & Answer section of your presentation.

Q&A

Optional Speaker Names Here