GlassFish v3

Kohsuke Kawaguchi
Sun Microsystems, Inc.

hk2.dev.java.net,
glassfish.dev.java.net

What's GlassFish v3?

- JavaEE 6
  - API for REST (JAX-RS)
  - Better web framework support (Servlet 3.0)
  - WebBeans, JSF 2.0, JPA 2.0, JAXB/WS 2.2, ...
- Great development experience
  - Continue great IDE integrations
  - Fast start-up
  - Embed into Maven/Ant
  - Automation support for testing

What's GlassFish v3?

- Scripting Language support
  - Run your RoR apps, Phobos apps, PHP apps, etc.
  - Take advantage of clustering, management, monitoring features built into GlassFish
  - Take advantage of script language support in NetBeans and develop/deploy smoothly into GlassFish
- Role-based access control in administration
- A lot more features planned

What's GlassFish v3?

- Modularization
  - Start small, and add modules as you need
  - We offer several tested-together configurations, but you can deviate and add more
  - Single code base to scale from simple servlet container to highly-available clustered application server
  - Foundation for middleware like JBI, SIP service, etc.

What's GlassFish v3?

- Extensibility
  - Repeat what happened to IDEs to application servers
  - Invest in mechanisms to define extensibility
- Significant infrastructure improvements
  - Make it easier for us to deliver better quality, quickly

Demo!

Module System

Module Subsystem : HK2

- 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
- Runs on Java SE 5.

What is a Module?

- Much like NetBeans module system
- Identified by a name and version
- Classloaders set up according to dependencies
- Exports a subset of its content (SPI).
- Imports other modules
  - identified by name&version
- Manifest file to represent these information

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.

Repository

- Repositories hold modules
- Can be added and removed at run time
- Different types supported
  - directory based
  - maven
  - OSGi ?
- Allows GFv3 to boot in many ways
  - From single uber-jar
  - From modules in your maven repository

Bootstrapping

- No more classpath at invocation
- HK2 bootstraps the “module world” and hand-over execution into GFv3

- to run : java -jar glassfish.jar

Build system : maven 2

- Each module is built from a maven project (pom.xml)
- pom.xml describes the module's
  - name
  - version
  - dependencies
- Module metadata created automatically from POM
  - No need to repeat yourself

Module Example

- Declare your module like:

```
<groupId>com.sun.enterprise.glassfish</groupId>
<artifactId>gf-web-connector</artifactId>
<packaging>hk2-jar</packaging>
```

- Module dependency = Maven dependency

```
<dependency>
  <groupId>com.sun.enterprise.glassfish</groupId>
  <artifactId>webtier</artifactId>
  <version>...</version>
</dependency>
```

Resulting definition

- Jar file manifest file

```
Built-By : hudson
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
...
```

Taking more advantages of Maven 2

- People no longer need to build the whole thing
  - Just build what you are changing
  - Fetch the rest from Maven repositories
- Sophisticated build-time processing is easier
  - Code generation
  - Byte-code post processing
  - Annotation processing
  - Metadata generation
- Running tools like findbugs is easier

Better OEM/Customization Possibility

- Create different distributions of GlassFish easily
  - “I want basic servlet container + Web Service”
  - “I want such and such version of JRuby/Spring/Hibernate with GFv3”
  - Custom branding
- Useful for specialized use, or large scale deployment

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.

Component Model and IoC

When you get IoC, why can’t we?

Services, services

- Service loader pattern on steroid
- Used extensively to identify extension points like:
  - Application containers (like servlets, Phobos, JRuby...)
  - Administrative commands
  - Admin console

Services in V3

- Interfaces are declared with @Contract
- Implementations are declared with @Service
- Build system generates metadata for runtime

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

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

Services can be named

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

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

Dependency Injection

- Declaratively inject components to fields/methods

```
@Inject ConfigService config;
```

- Programmatic retrieval

```
AdminCommand cmd =
  habitat.get(AdminCommand.class, "list-domains");
```

Configuration Injection

- Parse configuration XML file as component wiring description (somewhat like XBean or Ant tasks)
  - But JavaEE based
- Validate XML as we parse it
- Handle referential integrity
- JMX integration for management
- Extensibility built-in
  - Other modules can extend syntax
- Careful lazy loading

@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.

Container Lifecycle

Application container life-cycle

- 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

- 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

- 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
Startup[] startups;
```

GlassFish shutdown

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

  @Inject
  Startup[] startupSvcs;

  @Inject
  Habitat habitat;

  public void postConstruct() {
    habitat.removeComponents(startupSvcs);
  }
}
```

Conclusion

Summary : GlassFish V3

- Decomposition of the Java EE application server
- Fast edit/build/debug cycle
- Extensible: host all types of containers on JVM
- Embeddable
- Based on module subsystem (HK2)
- Use innovative and reusable components technology
- Available today in preview

For More Information

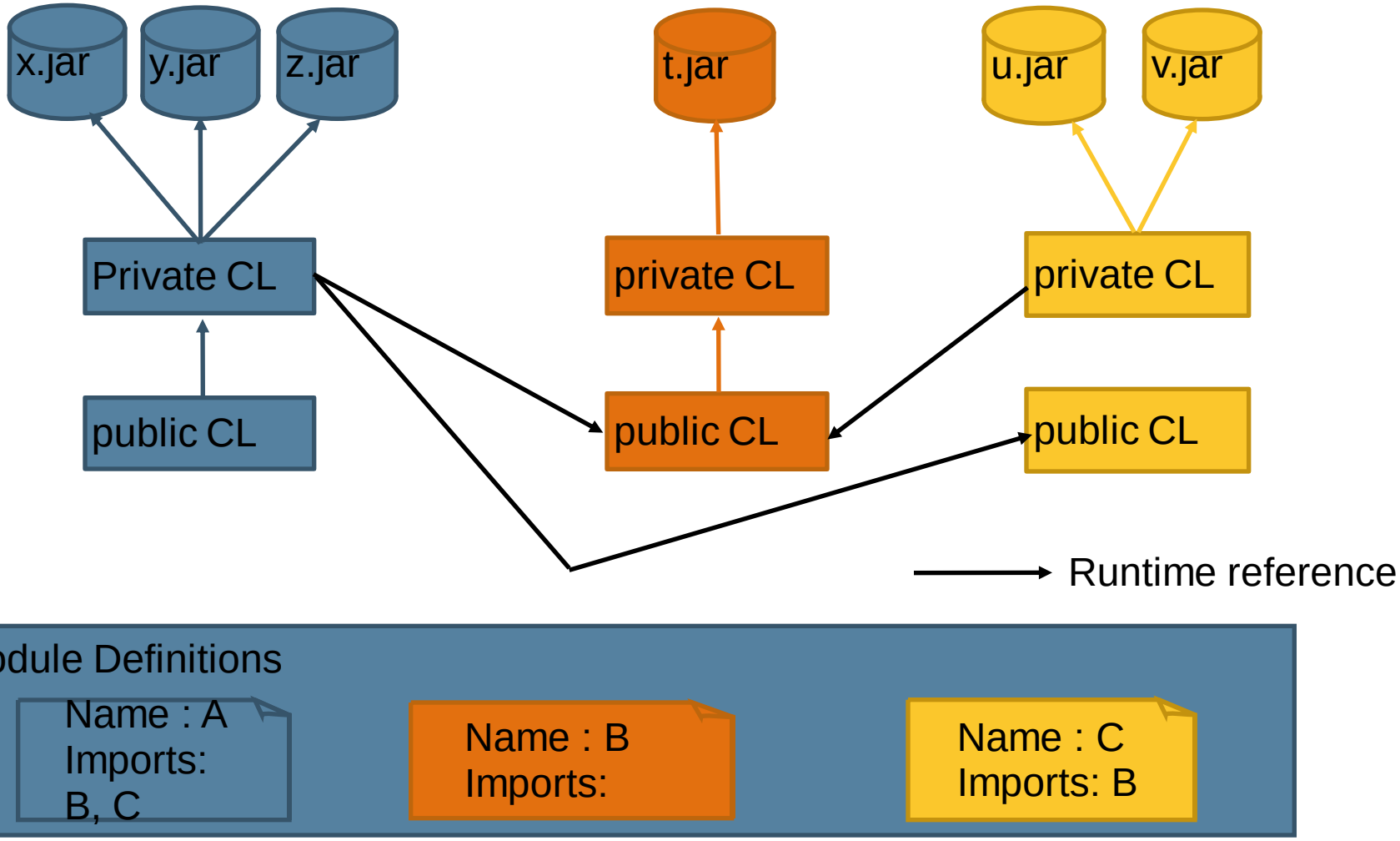
- Links
  - http://hk2.dev.java.net/
  - http://glassfish.dev.java.net/
  - http://wiki.glassfish.java.net/
- Emails
  - jerome.dochez@sun.com
  - kohsuke.kawaguchi@sun.com

Q&A

Nothing to see here

Back up slides

Runtime network of class loaders



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() {..}
```