Massive Enterprise Product Migration to OSGi

Raymond Auge
raymond.auge@liferay.com

LIFERAY®

bootstrapping

```
List<FrameworkFactory> frameworkFactories = ServiceLoader.load(
    FrameworkFactory.class, serviceLoaderCondition);

FrameworkFactory frameworkFactory = frameworkFactories.get(0);

Map<String, String> properties = _buildFrameworkProperties(
    frameworkFactory.getClass());

_framework = frameworkFactory.newFramework(properties);

_framework.init();
```

LIFERAY®

lifecycle of a WAR

```
@Override
public void contextInitialized(ServletContextEvent event) {
    try {
        ModuleFrameworkUtilAdapter.initFramework();
    }
    catch (Exception e) {
        ReflectionUtil.throwException(e);
    }

    super.contextInitialized(event);

    try {
        ModuleFrameworkUtilAdapter.startRuntime();
    }
    catch (Exception e) {
        ReflectionUtil.throwException(e);
    }
}
```

LIFERAY®

restricted view of the world

```
private ClassLoader moduleFrameworkClassLoader() throws Exception {
    List<ClasspathResolver> classpathResolvers = ServiceLoader.load(
        ClasspathResolver.class);

    ClasspathResolver classpathResolver = classpathResolvers.get(0);

    return new ModuleFrameworkClassLoader(
        classpathResolver.getClasspathURLs(),
        ClassLoaderUtil.getPortalClassLoader());
}
```

```
com.liferay.jaxws.osgi.bridge.jar
com.liferay.osgi.service.tracker.map.jar
com.liferay.portal.equinox.log.bridge.jar
com.liferay.registry.impl.jar
org.eclipse.osgi.jar
```

LIFERAY®

integrating spring, interleaving

```
try {
    ModuleFrameworkUtilAdapter.initFramework();

    _arrayApplicationContext = new ArrayApplicationContext(
        PropsValues.SPRING_INFRASTRUCTURE_CONFIGS);

    event.getServletContext().setAttribute(
        PortalApplicationContext.PARENT_APPLICATION_CONTEXT,
        _arrayApplicationContext);

    ModuleFrameworkUtilAdapter.registerContext(
        _arrayApplicationContext);

    ModuleFrameworkUtilAdapter.startFramework();
}
catch (Exception e) {
    ReflectionUtil.throwException(e);
}

super.contextInitialized(event);

ApplicationContext applicationContext =
    ContextLoader.getCurrentWebApplicationContext();

try {
    ModuleFrameworkUtilAdapter.registerContext(applicationContext);

    ModuleFrameworkUtilAdapter.startRuntime();
}
catch (Exception e) {
    ReflectionUtil.throwException(e);
}
```

LIFERAY®

beans as **services**

```
private void _registerApplicationContext(
    ApplicationContext applicationContext) {

    BundleContext bundleContext = _framework.getBundleContext();

    for (String beanName : applicationContext.getBeanDefinitionNames()) {
        Object bean = null;

        try {
            bean = applicationContext.getBean(beanName);
        }
        catch (BeanIsAbstractException biae) {
        }
        catch (Exception e) {
            _log.error(e, e);
        }

        if (bean != null) {
            _registerService(bundleContext, beanName, bean);
        }
    }
}
```

bean **service properties**

```
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface OSGiBeanProperties {

    public boolean portalPropertiesRemovePrefix() default true;

    public String portalPropertyPrefix() default "";

    public String[] property() default {};

    public Class<?>[] service() default {};

    public static class Convert {

        public static Map<String, Object> fromObject(Object object) {

        public static Map<String, Object> toMap(

        private static void _put(

    }

    public static class Service {

        public static Set<Class<?>> interfaces(Object object) {

        private static Set<Class<?>> _getInterfaceClasses(

    }

    public enum Type {

}
```

LIFERAY®

service registry from plain java

```
Registry registry = RegistryUtil.getRegistry();

Filter filter = registry.getFilter(
    "(&(javax.portlet.name=*)(objectClass=" +
        PollerProcessor.class.getName() + "))");

_serviceTracker = registry.trackServices(
    filter, new PollerProcessorServiceTrackerCustomizer());

_serviceTracker.open();
```

```
Registry registry = RegistryUtil.getRegistry();

Map<String, Object> properties = new HashMap<>();

properties.put("javax.portlet.name", portletId);

ServiceRegistration<PollerProcessor> serviceRegistration =
    registry.registerService(
        PollerProcessor.class, pollerProcessor, properties);

_serviceRegistrations.put(portletId, serviceRegistration);
```

LIFERAY®

legacy integration with registry dynamics

```
public static PollerProcessor getPollerProcessor(String portletId) {
    return _instance._getPollerProcessor(portletId);
}

private PollerProcessorUtil() {
    Registry registry = RegistryUtil.getRegistry();

    Filter filter = registry.getFilter(
        "(&(javax.portlet.name=*)(objectClass=" +
            PollerProcessor.class.getName() + "))");

    _serviceTracker = registry.trackServices(
        filter, new PollerProcessorServiceTrackerCustomizer());

    _serviceTracker.open();
}

private PollerProcessor _getPollerProcessor(String portletId) {
    return _pollerPorcessors.get(portletId);
}

private static final PollerProcessorUtil _instance =
    new PollerProcessorUtil();

private final StubMap<PollerProcessor> _pollerPorcessors =

private final ServiceTracker<PollerProcessor, PollerProcessor>
    _serviceTracker;

private static class PollerProcessorTargetLocator implements TargetLocator {

private class PollerProcessorServiceTrackerCustomizer
```

more to consider:

- legacy packaging, **WABs**
- legacy UI, **JSPs**
- **web resources**

Thank you!

LIFERAY®