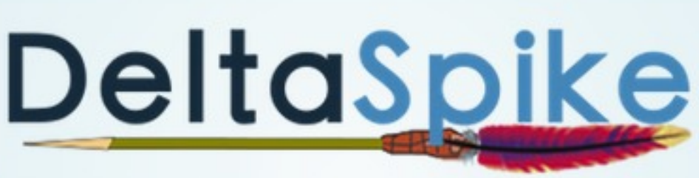
JAVAEE 6 => "APACHEE"

MAKE MODERN BUT STANDARD APPLICATIONS

Romain Manni-Bucau

Suivre @rmannibucau Follow @rmannibucau

AGENDA



WHO AM I?

Java/JavaEE developer at Atos Worldline (mid 2010)

Apache contributor and commiter (2011)

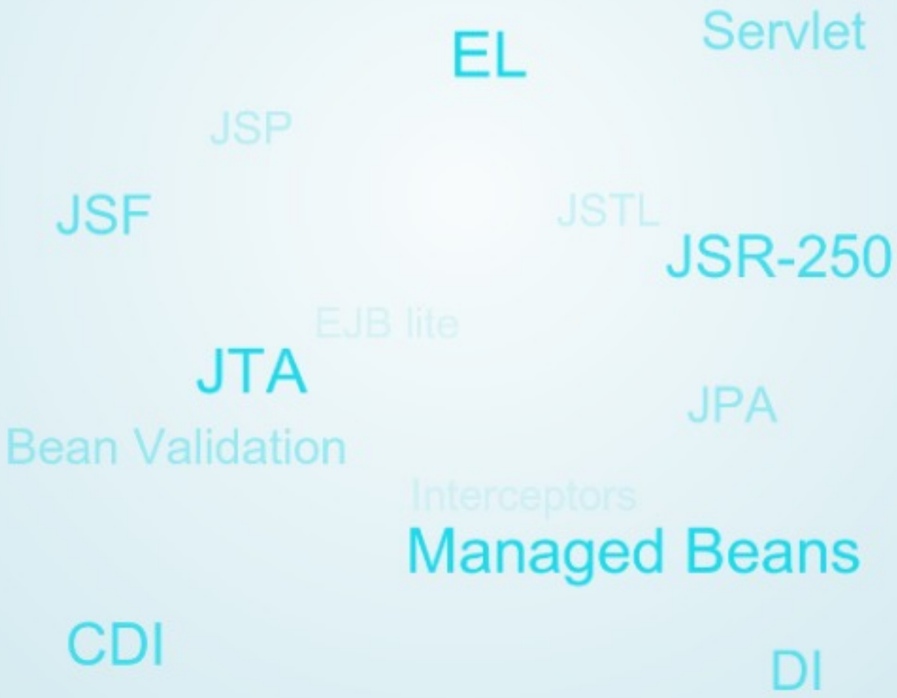
Suivre @rmannibucau Follow @rmannibucau

JAVAEE 6 (IN 2 WORDS)

JSR 316 -> ~2010, ~30 specs...and profiles

THE WEB PROFILE

Take the best of standards de facto!



CDI => IOC MADE STANDARD!

```
public class AnotherBean {
    @Inject // IoC
    private ABean bean;

    @Inject @SomeQualifier
    private ThirdBean third;

    @Inject // small but efficient synch Bus
    private Event<Ping> pingEvent;

    @PostConstruct // lifecycle
    public void init() {
        pingEvent.fire(new Ping("init done"));
    }

    public void calledOnEvent(@Observes Ping event) {
        // ...pong?
    }

    @Produces
    @ApplicationScoped // scopes
    public Bar produces() {
        return new Bar();
    }
}
```

And so on...

JAVAEE 6 :: OTHER GREAT ADDITIONS

- Bean validation
- JAX-RS
- EJBContainer API
- @Asynchronous
- @WebXXX + fragments
- ...

APACHE CODI

APACHE CODI

- Subproject of Apache MyFaces (JSF implementation)
- Adds some CDI features to JSF
- Doesn't depend on MyFaces -> portable!

APACHE CODI :: JPA

Highly "JavaSE" oriented

- ConfigurableDataSource (get the config from CDI)

```
<property name="openjpa.ConnectionDriverName"
          value="org.apache.myfaces.extensions.cdi
                 .jpa.impl.datasource.ConfigurableDataSource" />
```

- @Transactional, @TransactionScoped

```
@Transactional
public void delete(final Entity entity) {
    em.remove(em.merge(entity));
}
```

- ...

APACHE CODI :: I18N

Typed resource bundles

```
public interface MyBundle { // default -> org.foo.my_bundle.properties
    public class MyKey implements BundleKey, MyBundle {} //-> my_key

    @Named("secondKey")
    public class MyValue extends BundleValue implements MyBundle {}
}

public class MyBean {
    @Inject @Bundle(MyBundle.class)
    private ResourceBundle bundle;

    @Inject
    private MyBundle.MyValue value;

    public String valueFromBundle() {
        return bundle.getValue(MyBundle.MyKey.class);
    }

    public String directValue() {
        return value.toString();
    }
}
```

APACHE CODI :: TYPE SAFETY

```
@Page(basePath = "admin")
public interface AppPages extends ViewConfig {
    @Page(navigation = Page.NavigationMode.REDIRECT)
    public class Dashboard implements AppPages {}
}

// then in controller
public Class<? extends AppPages> goToDashboard() {
    someControllerAction();
    return AppPages.Dashboard.class;
}
```

APACHE CODI :: SECURITY

```java
@Page(basePath = "secured")
public interface SecuredPages extends ViewConfig {
    @Page(navigation = Page.NavigationMode.REDIRECT)
    @Secured(AccessVoter.class) // works for "normal" methods too
    public class Dashboard implements SecuredPages {}
}

@ApplicationScoped
public class AccessVoter extends AbstractAccessDecisionVoter {
    @Inject UserController user;
    @Inject @Jsf MessageContext mc; // provided by Codi

    public void checkPermission(final InvocationContext ic, final Set<SecurityViolation> violations) {
        if (!user.isLoggedIn()) {
            violations.add(newSecurityViolation(mc.message().text("{user_violation}").toText()));
        }
    }

}
```

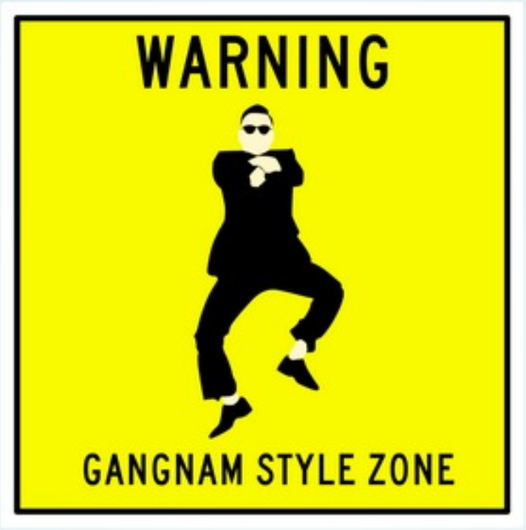
APACHE CODI :: ALREADY TOO MUCH SLIDES

- MessageResolver
- MessageInterpolator
- LocaleResolver
- ....and much more fluent API "string" oriented
- Codi Documentation

APACHE CODI :: ALREADY TOO MUCH SLIDES

- MessageResolver
- MessageInterpolator
- LocaleResolver
- ....and much more fluent API "string" oriented
- Codi Documentation

JBOSS SEAM



THIS IS DESIGNED FOR WELD!

SEAM :: MODULES

Several modules:

Config
Solder
Catch
JMS
Cron
Persistence
Remoting
REST
Security
Servlet
Spring
Social
Validation
...

~ JavaEE in Weld

JBOSS SEAM :: SOLDER, BASIS

Basic CDI extensions and tools to write your own ones

```
@Requires("javax.persistence.EntityManager") // highly not portable
public class MyBeanDependingOnFoo {}
```

-

```
@Inject
private Expressions expressions;

// then
expressions.evaluateValueExpression("#{fruitBowl.fruitName}");
```

-

```
@Inject @Resource("WEB-INF/beans.xml") URL beansXml;
```

-

- AnnotatedTypeBuilder, AnnotationInstanceProvider, AnnotationInspector...
- BeanManagerAware
- ...

SOLDER :: LOGGING AND R. BUNDLES

```java
@MessageLogger
public interface CelebritySightingLog {
    @Log @Message("Spotted celebrity %s!")
    void spottedCelebrity(String name);
}

// then in beans
@Inject @Category("trains")
private TrainSpotterLog log;

// or if you just want the default logger
@Inject
private Logger logger;
```

-

```java
@MessageBundle
public interface TrainMessages {
    @Message("No trains spotted due to %s")
    String noTrainsSpotted(String cause);
}

// then in beans
@Inject @MessageBundle
private TrainMessages messages;
```

SOLDER :: SERVICE HANDLER

```
// technical handler
class CustomHandler {
   @Inject
   private MyHelper helper;

   @AroundInvoke
   Object handle(InvocationContext ctx) {
      return helper.hanlde(ctx);
   }
}

// api
@ServiceHandlerType(CustomHandler.class)
@Retention(RUNTIME)
@Target({TYPE})
public @interface CustomService {}

// usage
@CustomService
public interface MyService {
   public Foo createFoo();
}
```

SOLDER :: XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:s="urn:java:ee"
       xmlns:my="urn:java:org.jboss.solder.config.xml.test.method">

   <!-- same for fields exists -->
    <my:MethodBean>
        <my:doStuff>
            <s:Produces/>
            <my:Qualifier1/>
            <s:parameters>
                <s:array dimensions="2">
                    <my:Qualifier2/>
                    <my:MethodValueBean/>
                </s:array>
            </s:parameters>
        </my:doStuff>
    </my:MethodBean>
</beans>
```

A smell of spring....in worse

SEAM :: SECURITY

- org.jboss.seam.security.Authenticator
- Custom model (JPA oriented)

-

```
@Entity
@IdentityEntity(IDENTITY_OBJECT)
public class IdentityObject implements Serializable {
    @Id @GeneratedValue
    private Long id;

    @IdentityProperty(PropertyType.NAME)
    private String name;

    @ManyToOne @IdentityProperty(PropertyType.TYPE)
    @JoinColumn(name = "IDENTITY_OBJECT_TYPE_ID")
    private IdentityObjectType type;
}
```

-

```
// then
@Inject IdentitySession identitySession;
// offers access to model management
```

SEAM :: INTERNATIONALIZATION

```
@Inject
private MessageFactory factory;

// then
final MessageBuilder builder = factory.info("Bla {0} bla {1} bla", 2013, "green");
final String msg = builder.build().getText();
```

SEAM :: JSF

```
// phase
public void observeRenderResponse(@Observes @RenderResponse PhaseEvent e) {}

// temporal
public void observeBefore(@Observes @Before PhaseEvent e) {}
public void observeAfter(@Observes @After PhaseEvent e) {}

// both can be combined

// component
public void observePrePasswordValidation(@Observes @Component("form:password") PreValidateEvent e) {}
```

- Bunch of events
- @RenderScoped, @ViewScoped
- ...

THROW NEW TOOMUCHSLIDEEXCEPTION()

- Bunch of extensions to replace a full JavaEE server (REST, JMS...)
- BVal integration (org.j.s.validation.InjectingConstraintValidatorFactory) + method validation
- Spring integration
- ...
- Soon replaced by DeltaSpike

APACHE DELTASPIKE

WHY DELTASPIKE?

- Common code/features between Seam 3 and Codi
- Promote CDI: unique project - JBoss + CDISource + Apache (to start)
- Ensure portability (tested on a bunch of containers)

DELTASPIKE :: MODULES

Today it is composed of:

- utilities
- jpa
- jsf
- security

DELTASPIKE :: CORE :: COMMON EXTENSIONS

```
//
// config
//

public class MyExtension implements Extension {
    public void observes(final @Observes SomeExtensionEvent event) {
        // injections not available in extensions so here the proposed solution
        final String value = ConfigResolver.getPropertyValue("my-key", "defaultValue");
    }
}

public class MyClass {
    @Inject
    @ConfigProperty(name = "key", value = "default")
    private Long maxIterations;
}

//
// include/exclude of beans
//

@Exclude(exceptIfProjectStage = ProjectStage.Production.class)
public class ProdBean {
    // some great prod business
}
```

DELTASPIKE :: CORE :: EXCEPTION

```
public class Dangerous {
    @Inject
    private Event<ExceptionToCatchEvent> exceptionEvent;

    public void methodThrowingAnException() {
        try {
            doSthgDangerous();
        } catch (final DangerousException ex) {
            exceptionEvent.fire(new ExceptionToCatchEvent(ex));
        }
    }
}

@ApplicationScoped
@ExceptionHandler
public class AbortingBreadthFirstHandler {
    public void abortHandler(final @BeforeHandles ExceptionEvent<ExceptionToCatchEvent> event) {
        event.abort(); // throwOriginal(), handled()...
    }
}
```

DELTASPIKE :: CORE :: INVOCATION HANLDER

```
@InvocationHandlerBinding
@Retention(RUNTIME)
@Target(TYPE)
public @interface PartialBeanBinding {}

@PartialBeanBinding
@RequestScoped
public class PartialBeanHandler implements InvocationHandler {
    @Inject
    private ABean bean;

    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        return this.testBean.doWhatYouWant(method, args);
    }
}

@PartialBeanBinding
@RequestScoped
public interface PartialBean {
    String getResult();
}
```

DELTASPIKE :: CORE :: JMX

```java
@ApplicationScoped
@MBean(description = "my mbean")
public class MyMBean {

    // Attributes

    @JmxManaged(description = "get counter")
    private int counter = 0;

    public int getCounter() { return counter; }
    public void setCounter(final int v) { counter = v; }

    // Operations

    @JmxManaged(description = "multiply counter")
    public int multiply(final int n) {
        return counter * n;
    }

    // Notifications

    @Inject
    private JmxBroadcaster broadcaster;

    public void broadcast() {
        broadcaster.send(new Notification(String.class.getName(), this, 10L));
    }
}
```

DELTASPIKE :: CORE :: MESSAGES

```java
// Message context
public class ABean {
    @Inject
    private MessageContext messageContext;

    public String generateMessage() {
        final Message message = messageContext
                .localeResolver(new FixedEnglishLocalResolver())
                .messageResolver(new MyMessageResolver())
                .message().template("{hello}").argument("hans");
        return message.toString();
    }
}

@MessageBundle
@MessageContextConfig(localeResolver = FixedEnglishLocalResolver.class) // defaults
public interface SimpleMessage {
    @MessageTemplate("Welcome to %s") // default = method name
    Message welcomeTo(MessageContext optionalMessageContext, String name);
}
```

DELTASPIKE :: CORE :: UTIL PART

Mainly a merge of Codi utility part + solder

```
// Project stage
@Inject
ProjectStage stage; // UnitTest, Development, SystemTest, IntegrationTest, Staging, Production, Custom

// Bean[Manager]Provider
final BeanManager bm = BeanmanagerProvider.getInstance().getBeanManager();
// doesn't work as expected for @Dependent (no clean)
final MyBean bean = BeanProvider.getContextualReference(MyBean.class);
final MyBean bean = BeanProvider.getContextualReference("myBean");
// ...
final Foo foo = new Foo();
final MyBean bean = BeanProvider.injectFields(foo);

// Literals
AnnotationLiteral<Default> defaultLiteral = new DefaultLiteral();
//... for default CDI scopes/qualifiers (DependentScopeLiteral...)

// Bean helper classes
BeanBuilder<T> beanBuilder = new BeanBuilder<T>(beanManager)
                .readFromType(annotatedType)
                .beanLifecycle(new MyLifecycle());
// ImmutableBean, WrappingBeanBuilder...

// Annotations
AnnotationInstanceProvider.of(MyAnnotation.class);

// AbstractContext...
```

DELTASPIKE :: CDICTRL

CDI 1.x doesn't provide any way to control a CDI container in plain JSE mode...so CDI is not usable?...No!

```
final CdiContainer cc = CdiContainerLoader.getCdiContainer();
cc.boot();

cc.getContextControl().startContexts(); // optional

final BeanManager bm = cc.getBeanManager();
//...

cdiContainer.getContextControl().stopContexts(); // optional


cc.getContextControl().startContexts(RequestScoped.class); // batch friendly ;)
// ...

cc.shutdown();
```

DELTASPIKE :: JPA

```
// Config
@ApplicationScoped
public class LocalDatabaseConfig implements DataSourceConfig {
    // @Inject ProjectStage stage;

    public String getJndiResourceName(String connectionId) { return null; } // JSE so no jndi name
    public String getConnectionClassName(String connectionId) { return "org.foo.DummyJdbcDriver"; }
    public String getJdbcConnectionUrl(String connectionId) { return "jdbc:dummy:mem:test"; }
    public Properties getConnectionProperties(String connectionId) { reutrn new Properties(); } // username...
}

// then org.apache.deltaspike.jpa.impl.datasource.ConfigurableDataSource used as DataSource
<property name="openjpa.ConnectionDriverName"
          value="org.apache.deltaspike.jpa.impl.datasource.ConfigurableDataSource" />

// @Transactional
@ApplicationScoped
public class FailedFlushTransactionalBean {
    @Transactional
    public void executeInTransaction() {}
}

// em is produced somewhere

// TransactionStrategy
// org.apache.deltaspike.jpa.impl.transaction.ResourceLocalTransactionStrategy
// org.apache.deltaspike.jpa.impl.transaction.BeanManagedUserTransactionStrategy (alternative)
```

DELTASPIKE :: SECURITY

```
@Secured(CustomAccessDecisionVoter.class) // or at method level
public class SecuredBean {
    //...
}

public class CustomAccessDecisionVoter implements AccessDecisionVoter {
    @Override
    public Set<SecurityViolation> checkPermission(AccessDecisionVoterContext accessDecisionVoterContext) {
        Method method = accessDecisionVoterContext.<InvocationContext>getSource().getMethod();

        //...create violations if mandatory
    }
}
```

DELTASPIKE :: SECURITY (ADVANCED)

```
@Retention(value = RUNTIME) @Target({TYPE, METHOD})
@SecurityBindingType
public @interface CustomSecurityBinding {}

@ApplicationScoped
public class CustomAuthorizer {
    @Secures @CustomSecurityBinding
    public boolean doSecuredCheck(InvocationContext ic, BeanManager bm, @LoggedIn User u) throws Exception {
        return u.isLoggedIn(); // perform security check
    }
}

@ApplicationScoped
public class SecuredBean1 {
    @CustomSecurityBinding
    public void doSomething(Thing thing) { thing.doSomething(); }
}

@Retention(value = RUNTIME) @Target({ PARAMETER })
@SecurityParameterBinding
public @interface CurrentThing { }

// in the app bean
public void doSomething(@CurrentThing Thing thing) { thing.doSomething(); }

// the authorizer
public boolean check(/* same */, @CurrentThing Thing t) throws Exception {
    return t.accept(u); // perform security check
}
```

DELTASPIKE :: JSF

- PhaseEvent
- ViewScope
- Typed Config (in progress)

-

Quite the same thing as described before with some renaming (@Page -> @View) and a bit more powerful (metadata - in progress)

DELTASPIKE :: NEXT?

- cdi-query import
- more merging

JBOSS ARQUILLIAN

ARQUILLIAN :: REMINDER

- Library proposed by JBoss to ease testing
- (Mainly) Java/JavaEE friendly
- Abstract Container management ("start/deploy/undeploy/stop")
- Packaging description ("on the fly" idea)
- Very extensible (and extended)

SWHRINKWRAP :: REMINDER

- Another library proposed by JBoss to ease testing/packaging
- Mainly a fluent API to describe Archives

```
ShrinkWrap.create(JavaArchive.class, "jug.jar")
        .addClass(Greeter.class)
        .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml")
```

ARQUILLIAN :: HELLO JUG!

```java
@RunWith(Arquillian.class)
public class GreeterTest {
    // which archive for the test
    @Deployment
    public static JavaArchive createDeployment() {
        return ShrinkWrap.create(JavaArchive.class)
            .addClass(Greeter.class)
            .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
    }

    // injection like in business beans thanks to enrichers
    @Inject
    Greeter greeter;

    // test as usual
    @Test
    public void sayHi() {
        assertEquals("Hello, Earthling!", greeter.greet("Earthling"));
    }
}
```

ARQUILLIAN :: GRAPHENE

```java
@RunWith(Arquillian.class)
public class SimpleServletTest {
    // deployment as usual

    @ArquillianResource
    private URL contextRoot;

    @Drone
    private WebDriver driver;

    @FindBy(id = "content") // you can create a pojo to put all @FindBy and inject it with @Page
    private WebElement content;

    @FindBy(xpath = "//div[@id='footer']") // webelements can be button and so on!
    private WebElement footer;

    @Before public void openPage() {
        driver.get(contextRoot.toExternalForm() + "simple");
    }

    @Test public void checkContent() throws IOException {
        assertEquals("the content", content.getText());
        assertTrue(content.isDisplayed());
    }

    @Test public void checkFooter() throws IOException {
        assertEquals("the footer", footer.getText());
    }
}
```

ARQUILLIAN :: WARP

```
@WarpTest
@RunWith(Arquillian.class)
public class WarpDemoTest {
    @Deployment public static WebArchive war() {
        return SWUtils.myWar();
    }

    @ArquillianResource private URL contextRoot;

    @Drone private WebDriver driver;

    @Test @RunAsClient
    public void validSessionUpdate() throws IOException {
        Warp.execute(new ClientAction() {
            @Override public void action() {
                driver.navigate().to(contextRoot.toExternalForm() + "hello");
            }
        }).verify(new ServerAssertion { // need to be static until Alpha1
            @ArquillianResource private HttpServletRequest request;

            @BeforeServlet public void beforeServlet() {
                assertNull(request.getSession().getAttribute("name"));
            }

            @AfterServlet public void afterServlet() {
                assertEquals("hello", request.getSession().getAttribute("name"));
            }
        });
    }
}
```

ARQUILLIAN :: MUCH MORE

- Run cluster of servers
- use it with Cucumber (@See CukeSpace on github)
- use it with Spock!
- ...

APACHE TOMEE

APACHE TOMEE :: CORE VALUES

APACHE TOMEE :: CORE VALUES

- Be small

APACHE TOMEE :: CORE VALUES

- Be small
  - < 30M with the WP fragance

APACHE TOMEE :: CORE VALUES

- Be small
  - < 30M with the WP fragance
- Be Tomcat

APACHE TOMEE :: CORE VALUES

- Be small
  - < 30M with the WP fragance
- Be Tomcat
- Be certified

APACHE TOMEE :: CORE VALUES

- Be small
  - < 30M with the WP fragance
- Be Tomcat
- Be certified
  - JavaEE 6 Web Profile (and JAXRS) certified

APACHE TOMEE :: CORE VALUES

- Be small
  - < 30M with the WP fragance
- Be Tomcat
- Be certified
  - JavaEE 6 Web Profile (and JAXRS) certified
  - Run on EC2 with default JVM settings

APACHE TOMEE :: WHY?

- JavaEE is not heavy!
- Web Profile! We (almost) did it for years.
- Tomcat == most used server (> 50%)
- Everybody does the same integration work

APACHE TOMEE :: MISSING IN TOMCAT

- Transactions
- Security
- JNDI (java:global, ...)
- `@Resource`
- `@PersistenceUnit, @PersistenceContext`
- `@Inject`
- `@EJB`
- `@DataSourceDefinition`
- ...

APACHE TOMEE :: STACK & FLAVORS

- Web profile
  - OpenWebBeans
  - OpenEJB
  - OpenJPA
  - MyFaces
  - Bean Validation
- JAXRS
  - CXF (jaxrs)
- Plus*
  - CXF (jaxws)
  - ActiveMQ
  - Geronimo
- Embedded*
  - Classpath ;)

APACHE TOMEE :: DATASOURCES

- Pluggable Pooling
  - commons-dbcp
  - tomcat-jdbc
  - bonecp
- Dynamic DataSource
  - Proxy DataSource able to switch between real ones by tx
  - "Router" API

APACHE TOMEE :: CONFIGURATION

- Mainly tomee.xml (or resources.xml with placeholders) for extenal stuff

```
<tomee>
    <Resource id="jdbc/my-ds">
        # Properties like format, case insensitive
        JdbDriver = com.mysql.Driver
        JdbcUrl = jdbc:mysql://localhost:3306/demo
        UserName = mysql
        Password = mysql
    </Resource>
</tomee>
```

- Advanced configuration
  - application.properties
    - configuration scoped by application of internal configuration
    - ex: quartz
  - openejb-jar.xml (application configuration)...
    - configuration of specific beans (EJBs, JAXRS endpoints...)

APACHE TOMEE :: JPA

- openejb.jpa.init-entitymanager
  - force eager initialization
- openejb.jpa.criteria.log.jpql
  - log criteria API queries as JQPL
- openejb.jpa.auto-scan, openejb.jpa.auto-scan.package
  - optimized entities scanning (!= JPA provider scanning)
- openejb.jpa.table_prefix
- PUs exposed as MBeans
  - updatable "on the fly"

-

SQL logging:

```
<Resource id="jdbc/my-ds">
    LogSql = true
    ...
</Resource>
```

APACHE TOMEE :: MAVEN PLUGIN 1/3

```
<plugin>
    <groupId>org.apache.openejb.maven</groupId>
    <artifactId>tomee-maven-plugin</artifactId>
    <version>${tomee.version}</version>
</plugin>
```

APACHE TOMEE :: MAVEN PLUGIN 2/3

```
<plugin>
    <groupId>org.apache.openejb.maven</groupId>
    <artifactId>tomee-maven-plugin</artifactId>
    <version>${tomee.version}</version>
    <configuration>
        <tomeeClassifier>jaxrs</tomeeClassifier>
        <debugPort>5005</debugPort>
        <config>${project.basedir}/src/test/tomee/conf</config>
        <args>-Xmx512m</args>
        <systemVariables>
            <foo>bar</foo>
        </systemVariables>
        <libs>
            <lib>mysql:mysql-connector-java:5.1.20</lib>
        </libs>
        <webapps>
            <webapp>org.superbiz:my-simple-webapp:1.0.0</webapp>
        </webapps>
    </configuration>
</plugin>
```

APACHE TOMEE :: MAVEN PLUGIN 3/3

```
<plugin>
    <groupId>org.apache.openejb.maven</groupId>
    <artifactId>tomee-maven-plugin</artifactId>
    <version>${tomee.version}</version>
    <configuration>
        <reloadOnUpdate>true</reloadOnUpdate>
        <synchronizations>
            <synch> <!-- synch resources -->
                <source>${project.basedir}/src/main/webapp</source>
                <target>${project.build.directory}/apache-tomee/webapps/ROOT</target>
                <extensions><extension>.xhtml</extension></extensions>
            </synch>
            <synch> <!-- synch binaries triggering reload because of reloadOnUpdate -->
                <source>${project.build.outputDirectory}</source>
                <target>${project.build.directory}/apache-tomee/webapps/ROOT/WEB-INF/classes</target>
                <updateOnlyExtensions><updateOnlyExtension>.class</updateOnlyExtension></updateOnlyExtensions>
            </synch>
        </synchronizations>

        <!-- or -->
        <synchronization> <!-- default fine for webapps -->
          <extensions>
            <extension>.xhtml<extension>
          </extensions>
        </synchronization>
    </configuration>
</plugin>
```

APACHE TOMEE :: ADVANCED STUFF

- EJB multicast, failover (stateless)
- injection of remote ejbs (@EJB)
- provisioning module (url, maven...)
- skinny wars (jars.txt)
- ...

ARQUILLIAN TOMEE

ARQUILLIAN TOMEE :: TEST BEFORE

```
@BeforeClass
public static void start() throws Exception {
    final Properties p = new Properties();

    p.setProperty("jdbc/ds", "new://Resource?type=DataSource");
    // ... set file to deploy or use classpath by default

    container = EJBContainer.createEJBContainer(p);
}

@Before
public void injectAndInit() throws NamingException {
    // magic OpenEJB hook
    container.getContext().bind("inject", this);
}

@Inject
private Users users;

@Test
public void findByName() {
    // ...
}

@AfterClass
public static void close() {
    container.close();
}
```

ARQUILLIAN TOMEE :: ENRICHMENT

- Forget conventions and
  - cdi enricher
  - ejb enricher
  - resource enricher
  - persistence enricher

    That's native!

ARQUILLIAN TOMEE :: CONFIGURATION

```
<arquillian>
  <container qualifier="tomee-remote" default="true">
    <configuration> <!-- ALL IS OPTIONAL -->
      <property name="httpPort">-1</property> <!-- same for ajpPort and stopPort -->
      <property name="dir">target/tomee-remote</property>
      <property name="appWorkingDir">target/arquillian-remote-working-dir</property>
      <property name="conf">src/test/tomee/conf</property> <!-- same for bin and lib -->
      <property name="portRange">20001-30000</property>
      <property name="cleanOnStartUp">true</property>
      <property name="properties" />
      <property name="serverXml">src/test/tomee/server.xml</property>
      <property name="quickSession">true</property>
      <property name="groupId">org.apache.openejb</property>
      <property name="artifactId">apache-tomee</property>
      <property name="version">LATEST</property>
      <property name="type">zip</property>
      <property name="classifier">jaxrs</property>
      <property name="catalina_opts">-Dmy-test=config</property>
      <property name="removeUnusedWebapps">true</property>
      <property name="debug">false</property>
      <property name="debugPort">5005</property>
    </configuration>
  </container>
</arquillian>
```

ARQUILLIAN TOMEE :: SINGLE DEPLOYMENT

- Deploy an application at startup...from a ShrinkWrap archive

```
<arquillian>
  <container qualifier="tomee" default="true">
    <configuration>
      <property name="properties">
        openejb.arquillian.predeploy-archives = org.apache.openejb.arquillian.openejb.archive.Archive1
      </property>
    </configuration>
  </container>
</arquillian>
```

ARQUILLIAN TOMEE :: ADAPTERS

- TomEE Embedded
- TomEE Remote
- TomEE Webapp
- OpenEJB

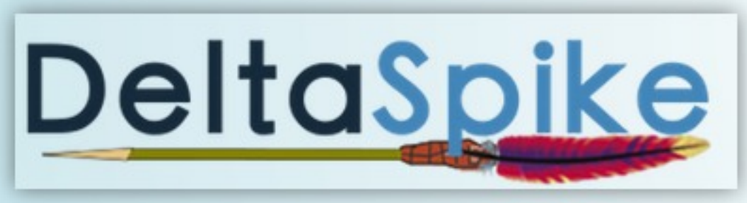
ARQUILLIAN TOMEE :: OPENEJB ADAPTER

- 100% embedded
- no archive dump (ZipExporter)
- Mockito integration

```
@RunWith(Arquillian.class)
public class ArquillianAndMockitoTest {
    @Mock @Produces private static AnInterface mock;
    @Deployment public static JavaArchive archive() { return SWUtils.jar(); }
    @Test public void test() { /* ... */ }
}
```

- single deployment feature
- working virtual resources

DEMO!



THANKS

OPEN TO ANY QUESTION (OR FOR A DRINK ;)

Slides available on Github