Airhacks 2016/06/13

Alexander Sparkowsky

JCP / JavaEE

- Download all JavaEE specs
- When problems occur: Perform full-text search
- Don’t have to read the whole spec

Annotation in JavaEE

Basics

- Use regular Java annotation
- In JavaEE the only retention that matters is `RetentionPolicy.RUNTIME`
- Using `value()` allows to use annotations like `@NoMagic("the value")`

```
package com.airhacks;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

@Target({ElementType.FIELD, ElementType.METHOD})
@Retention(RetentionPolicy.RUNTIME)
public @interface NoMagic {

    String value();

    int priority() default 42;

}
```

UnitTest for annotations

```
@Test
public void findAnnotation() {
    Field[] declaredFields = CargoCult.class.getDeclaredFields();
    for (Field declaredField : declaredFields) {
        NoMagic annotation = declaredField.getAnnotation(NoMagic.class);
        if (annotation != null) {
            System.out.println("annotation = " + annotation + " on field " +
declaredField + " with value: " + annotation.value() + " prio " + annotation.priority
());
        } else {
            System.out.println("declaredField = " + declaredField + " is not
annotated");
        }
    }
}
```

Convention over configuration

- Minimal projects possible
- Today applications can be written without XML

CDI

- DI is no magic

Listing 1. Simple DI implementation

```
package com.airhacks;

import java.lang.reflect.Field;
import org.junit.Test;

/**
 *
 * @author airhacks.com
 */
public class WizardTest {

    @Test
    public void cargoCultInjection() throws ClassNotFoundException,
InstantiationException, IllegalAccessException {
        Class<?> clazz = Class.forName("com.airhacks.Wizard");
        Field[] allFields = clazz.getDeclaredFields();
        for (Field field : allFields) {
            if (field.isAnnotationPresent(Magic.class)) {
                Class<?> typeOfField = field.getType();
                System.out.println("typeOfField = " + typeOfField);
                Object service = typeOfField.newInstance();
                Object client = clazz.newInstance();
                field.set(client, service);
                System.out.println("---- " + client);
            }
        }

    }

}
```

- Default constructor allows dynamic instantiation
- Don't perfom initialization in constructors
  - Injected fields not set / Object nor ready yet
  - Constructor may not be called always
  - Use `@PostConstruct`
- `@Produces` can be applied to a variable

```
public class SimpleProducer {
    @Produces
    private String producedString = "the value";
}
```

- Qualifier annotations are used to distinguish multiple injected values of same type

> Only use qualifiers in your project if you really need them

> In usual projects you don't need `@Produces` or qualifiers

- `InjectionPoint` can be used in `@Produces` methods
  - Class of injection point
  - Member name (name of field) where the field is injected
  - **Add own annotation to the member where the value is injection without the need to be a `@Qualifier`**
    - Get the annotation from the injection point and use it
- Interfaces can also be injected
  - In this case qualifiers will be most likely be necessary to avoid ambiguity

Listing 2. Sample annotation to distinguish interface implementation

```
package com.airhacks;

import javax.inject.Qualifier;
import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 *
 * @author airhacks.com
 */
@Qualifier
@Target({ElementType.TYPE, ElementType.PARAMETER})
@Retention(RetentionPolicy.RUNTIME)
public @interface Magic {

    Kind value();

    // tag::embeddedenum[]
    enum Kind {
        WHITE, BLACK
    }
    // end::embeddedenum[]
}
```

Note that the `Kind` is implemented as inner enumeration.

```
enum Kind {
    WHITE, BLACK
}
```

JAX-RS

- `@ApplicationPath("myApp")` specifies the path *below* the context path of the application
- A JavaEE application can only become a root context via proxy (apache, NGINX)

Aspects / Interception

- if a class is annotated with `@Interceptors(MyAuditorClass.class)` all public methods will be intercepted
- A class can only be intercepted when it is created using CDI (it will not be intercepted if created in the consturctor or elsewhere using `new`)

Listing 3. Intercepting Auditor

```
package com.airhacks;

import javax.interceptor.AroundInvoke;
import javax.interceptor.InvocationContext;

/**
 *
 * @author airhacks.com
 */
public class Audit {

    @AroundInvoke
    public Object onCall(InvocationContext ic) throws Exception {
        System.out.println("Before call: " + ic.getMethod());
        return ic.proceed();
    }

}
```

- Most JavaEE services are build using aspects
- Not the actual bean is injected but a proxy

Events

- `@Observes` can distinguish between successful and failed transactions by using `during = TransactionPhase.AFTER_⋯` attribute
  - Behaves transitionally: Event will be delivered at the *end* of the transaction
- Rollbacks will discard "regular" CDI events

```
public void onSuccessfulBooking(@Observes(during = TransactionPhase.AFTER_SUCCESS)
String booking) {
    System.out.println("++event- " + booking);
}

public void onFailedBooking(@Observes(during = TransactionPhase.AFTER_FAILURE) String
booking) {
    System.out.println("--event- " + booking);
}
```

```
@Resource
SessionContext sc;

@Asynchronous
public Future<String> validateAddress() {
    try {
        //sc.setRollbackOnly();
        Thread.sleep(2000);
    } catch (InterruptedException ex) {
        Logger.getLogger(AddressValidator.class.getName()).log(Level.SEVERE, null, ex
);
    }
    return new AsyncResult<>("42-" + System.currentTimeMillis());
}
```

EJB

- Entrypoint to a business function (e. g. chain of injected beans) should be an EJB (handles TX etc.)
- Best performing kind of bean
- Introduces Monitoring
- EJB is a very mature technology
  - It comes with aspects, transactions and pooling
  - It's faster than regular CDI beans
    - Less injection thanks to pooling. In CDI beans injection takes place on every request.
    - `@Asynchronous` is working

When to use `@Stateful`

- Instead of using `@SessionScoped` it is possible to use `@Stateful` beans.
  - Get transactions for free
  - Useful for multi-step UIs
  - At the end of the process call a method annotated with `@Remove` which will remove the stateful bean

JNDI

- In earlier days used instead of CDI
- Lookup of components
- Now done behind the scenes by CDI

Transactions

- There will only be one instance per entity (identity) per transaction
- Transactions in the application servers are JTA transaction

NOTE	JTA transaction != database transaction

INFO: Usually JTA transactions are *not* less performant than database transactions

Data-sources

- If the data-source (`<jta-data-source>jdbc/sample</jta-data-source>`) is not present in `persistence.xml` the default data-source will be injected

- `@PersistenceContext(unit="name")` specifies the persistence unit of an `EntityManager` if multiple persistence-units are specified in *one* or multiple `persistence.xml` files using the `name` property of `<persistence-unit name="prod" transaction-type="JTA">`

Exception handling

- Checked `Exceptions` will *not* rollback a transaction
- Unchecked `Exceptions` *will* rollback a transaction

Transaction management configuration

- Usually container managed transactions will be used
  - Bean managaged transactions allow to specify a transaction timeout per method

```
@Stateless
@TransactionManagement(TransactionManagementType.BEAN)
public class MyBean {
    @Resource
    UserTransaction ut;

    public void doSomething() {
        ut.begin();
        //...
        ut.commit();
    }
}
```

- Default for all EJB methods (`@Stateless`): `@TransactionAttribute(TransactionAttributeType.REQUIRED)`
  - It's not necessary to specify transaction attribute for methods
  - Other interesting variants (rarely needed)
    - `NOT_SUPPORTED` will suspend a running transaction
    - `MANDATORY` requires an already running transaction

Bind Bean to a transaction

- `@TransactionScoped` Beans disappear after the transaction finished (independent of the outcome)

Listing 4. Alternative way to store a bean for the duration of a transaction

```
@Resource
TransactionSynchronizationRegistry tsr;

//...
// Store objects during the lifetime of a transaction
tsr.putResource(key, value)
```

Integrate custom modules/libraries

- Integrate a custom module without the need to rebuild an enterprise application
  - Use webhooks
    - Communication by REST
    - Use callback URLs
    - Something like a remote module

Asynchronous

- User `Future` to return values that will be computed later
  - A future is just a container to perform an action
  - `get()` will block/return the value computed by the future
- It's possible to create/collect multiple `Future` values in a coll

Listing 5. Usage of a `Future`

```
List<Future<String>> results = new ArrayList<>();
for (int i = 0; i < 5; i++) {
    results.add(this.av.validateAddress());
}
String vals = results.stream().map(f -> {
    try {
        return f.get();
    } catch (InterruptedException | ExecutionException ex) {
        return null;
    }
}).collect(Collectors.joining(","));

System.out.println("val = " + vals);
```

- Annotation `@Schedule` on a method can be used to invoke a method in regular intervals. Similar to cron semantic.
- `@Startup @Singleton @DependsOn` can be used to specify the order in which singletons are created

Concurrency management

- Using `@ConcurrencyManagement(ConcurrencyManagementType.BEAN)` on a bean will disable all locks on the bean

JSP

> They are coming back

> Best template langauge. Better than *FreeMarker* and *Velocity*

- Fasted and best possible template language when using multipage applications
  - SPA communicate via REST
- `@Model` stereotype for `@Named` and `@RequestScoped`
- Can be used to generate reports
  - Former Airhacks attendee Stylight is using JSP instead of JavaScript clients
    - Customized version of struts and JSP
    - Better Google scoring

JSF

> Enterprises write JavaScript clients and cures about JSF because the try to tweak components libraries like Primefaces. This produces a lot of problems. Startups use the component libraries without any change and are happy with them

NOTE	If you are using a component library use the components as they are without wanting to change them

Resources

http://blog.websitesframeworks.com

- MVC framework: http://vraptor.org (combines CDI + JSP) archetype for JavaMVC
  - Also useful for invoice generation or legal stuff

Persistence

Initialize a database for development

- Generate the schema programmatically by using `javax.persistence.Persistence#generateSchema(String persistenceUnitName, Map map)`
- In project stage:
  - https://flywaydb.org (simpler, jenkins)
  - http://liquibase.org
- In production:
  - `@Startup @Singleton` together with entities

Listing 6. Generate DDL by eclipselink/hibernate → Transform to flywaydb.org

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" version="2.1" xsi:schemaLocation=
"http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="EFS2015-persistence-unit" transaction-type="JTA">
    <description>Forge Persistence Unit</description>
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
    <exclude-unlisted-classes>false</exclude-unlisted-classes>
    <properties>
      <property name="hibernate.dialect" value=
"org.hibernate.dialect.PostgreSQLDialect"/>

      <property name="javax.persistence.schema-generation.scripts.action" value="drop-
and-create"/>
      <property name="javax.persistence.schema-generation.scripts.create-target"
value="./create.sql"/>
      <property name="javax.persistence.schema-generation.scripts.drop-target" value=
"./drop.sql"/>
    </properties>
  </persistence-unit>
</persistence>
```

http://www.thoughts-on-java.org/standardized-schema-generation-data-loading-jpa-2-1

EL

- Expression language can be used in any project inside Java code not only in JSF/JSP

```
package com.airhacks.el;

import java.util.Map;
import java.util.Set;
import javax.el.ELProcessor;
import static org.hamcrest.CoreMatchers.is;
import static org.junit.Assert.assertFalse;
import static org.junit.Assert.assertNotNull;
import static org.junit.Assert.assertThat;
import org.junit.Before;
import org.junit.Test;

/**
 *
 * @author adam-bien.com
 */
public class ELTest {

    private ELProcessor cut;

    @Before
    public void init() {
        this.cut = new ELProcessor();
    }

    @Test
    public void formula() {
        Long result = (Long) this.cut.eval("2*2");
        assertThat(result, is(4l));
    }

    @Test
    public void bean() {
        Workshop workshop = new Workshop("javaee airhacks");
        this.cut.defineBean("workshop", workshop);
        String title = (String) this.cut.eval("workshop.title");
        assertThat(title, is(workshop.getTitle()));
    }

    @Test
    public void listLiteral() {
        String listLiteral = "{1,2,3}";
        Set<Integer> list = (Set<Integer>) this.cut.eval(listLiteral);
        assertFalse(list.isEmpty());
        System.out.println("List: " + list);
    }
```

```
    @Test
    public void mapLiteral() {
        String listLiteral = "{\"one\":1,\"two\":2,\"three\":3}";
        Map<String, Long> map = (Map<String, Long>) this.cut.eval(listLiteral);
        assertFalse(map.isEmpty());
        System.out.println("Map: " + map);
    }

    @Test
    public void lambda(){
        Object result = this.cut.eval("[1,2,3,4,5,6,7,8].stream().filter(i->i%2 ==
0).map(i->i*10).toList()");
        assertNotNull(result);
        System.out.println("Result: " + result);
        result = this.cut.eval("[1,5,3,7,4,2,8].stream().sorted((i,j)->j-i).toList()"
);
        System.out.println("Result: " + result);
    }

}
```

JAX-RS

- Return `JsonObject` or `Response`
- Take a brief look at the HTTP specification (for POST)
  - Return 200 or 204 (ok) or 201 (Created) with location by using `Response.created(location).build()`

Listing 7. Create and return the location of an object created by POST

```
Unresolved directive in Airhacks-20160613.adoc - include::/Users/alxs/Development
/Training/Java/Airhacks/airhacks/jax-rs/src/main/java/com/airhacks/MessagesResource
.java[tag=post]
```

Listing 8. Return an object referenced by the location header

```
Unresolved directive in Airhacks-20160613.adoc - include::/Users/alxs/Development
/Training/Java/Airhacks/airhacks/jax-rs/src/main/java/com/airhacks/MessagesResource
.java[tag=location_get]
```

JAXB vs. JsonObject

- Preferred JsonObject over JAXB because `JsonObject` gives more flexibility and most APIs are JSON anyway.
- Also you have to map them manually it's worth the effort

> I use it more and more. Even in my OSS projects.

HATEAOS

> Good to know but never be to extreme because you loose a lot of time in the last 10%

Resources

Questions

Sub-sub-sub-sub-resources Example: /cache/{code}/properties/{key} and /properties/{key} HATEOAS