Testing, Maven and JAX

Jaroslav Dytrych

Faculty of Information Technology Brno University of Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole
dytrych@fit.vutbr.cz

26 September 2023

Testing

- The Four Levels of Software Testing
  - Unit Testing – one component
  - Integration Testing – more components, focused to interfaces
  - System Testing – whole system
  - Acceptance Testing – whole system, customer stories
- Regression Testing – verify that a fixed bug or new feature has not resulted in another functionality failure or business rule violation
- Fixture – all input data for the test such that it is possible to repeatedly run the test regardless of the current context (testing data, configuration of test, . . .)
- Test case – the set of tests with one fixture which are testing one unit
- Test suite – the set of test cases (e.g. all units in the given system)

- JUnit
- Arquillian (Integration tests)
- Selenium (Web Browser Automation)
- Hudson/Jenkins (Continuous integration)

- JUnit
  - xUnit family, based on SUnit (smalltalk)
  - de facto standard
  - JUnit 4, older versions don't use Java Annotations

```
import org.junit.Test;
import static org.junit.Assert.*;
import com.example.foo.Foo;
public class FooTest {
    @Test
    public void testAnswer() {
        Foo foo = new Foo();
        assertEquals(42, foo.answer());
    }
}
java -cp junit-4.jar:. org.junit.runner.JUnitCore FooTest
```

- Methods which are called before and after each test
  - `@org.junit.Before`
    `void setUp()`
  - `@org.junit.After`
    `void tearDown()`
- Expected Exceptions (JUnit 4)

```
@Test(expected=IndexOutOfBoundsException.class)
public void empty() {
    new ArrayList<Object>().get(0);
}
```

- Assert Exception (JUnit 5)

```
@Test
public void empty() {
  ArithmeticException thrown =
    Assertions.assertThrows(
      ArithmeticException.class, () -> {
        calculator.divide(10, 0);
    });
  Assertions.assertEquals("/ by zero",
                          thrown.getMessage());
```

- Timeout

```
@Test(timeout=100)
public void empty() {
    while (true);
}
```

JUnit assertions

- `assertEquals(expected, actual)`
- `assertEquals(String message, expected, actual)`
- `assertNull(object)`
- `assertNotNull(object)`
- `assertNull(String message, object)`
- `assertNotNull(String message, object)`
- `assertSame(expected, actual)`
- `assertSame(String message, expected, actual)`
- `assertTrue(boolean actual)`
- `assertTrue(String message, boolean actual)`

JUnit – Fail

- Test failure can be indicated explicitly
- `fail()`
- `fail(String message)`

Example NumberGenerator

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses({
    MyClassTest.class,
    MyOtherClassTest.class
})
public class AllTests {
    // Empty class
}
```

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class TestRunner {
  public static void main(String[] args) {
    Result result = JUnitCore.runClasses(
      AllTests.class);

    for (Failure failure : result.getFailures()) {
     System.out.println(failure.toString());
    }

    System.out.print("Overall success: ");
    System.out.println(result.wasSuccessful());
  }
}
```

- Test multiple components together
- In-container testing
- Arquillian `http://arquillian.org/`
  - Special suite for running integration tests
  - An Arquillian test case must have three things:
    - a `@RunWith(Arquillian.class)` annotation on the class,
    - a public static method annotated with `@Deployment` that returns a ShrinkWrap archive,
    - at least one method annotated with `@Test`.
  - The purpose of the test archive is to isolate the classes and resources which are needed by the test from the remainder of the classpath

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

- ShrinkWrap

  `https://arquillian.org/modules/shrinkwrap-shrinkwrap/`
  - can be viewed as a build tool
  - Shrinkwrap provides a simple API to assemble archives like JARs, WARs, and EARs in Java.

- Contexts – the ability to bind the lifecycle and interactions of stateful components to well-defined but extensible lifecycle contexts
- Dependency injection – the ability to inject components into an application in a typesafe way, including the ability to choose at deployment time which implementation of a particular interface to inject
- Dependency injection always occurs when the bean instance is first instantiated by the container. Simplifying just a little, things happen in this order:
  1. The container calls the bean constructor (the default constructor or the one annotated `@Inject`), to obtain an instance of the bean.
  2. The container initializes the values of all injected fields of the bean.
  3. The container calls all initializer methods of bean (the call order is not portable, don't rely on it).
  4. Finally, the `@PostConstruct` method, if any, is called.

`https://docs.jboss.org/weld/reference/2.0.4.Final/en-US/html_single/#d0e1126`

- Tests dependency injection
- CDI events, interceptors
  - `@PostConstruct`
  - `@PreDestroy`
  - `...`
- ShrinkWrap example

```
@Deployment
public static JavaArchive createDeployment() {
    JavaArchive jar = ShrinkWrap.create(JavaArchive.class)
        .addClasses(Greeter.class, PhraseBuilder.class)
        .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
    // System.out.println(jar.toString(true));
    return jar;
}
```

Example calculator-junit_arquillian

- Functional Testing
  - tests the features/functionality
  - black box testing, usually whole application
- Selenium
  - Web browser remote control
  - Selenium Java API
  - Firefox, Chrome, Edge, Safari, Opera, ...
  - `http://seleniumhq.org/`
  - Web page crawling
  - Reading autocomplete values
  - Forms
    - Find form inputs
    - Fill them and submit form
    - Check outputs
  - Check elements enabled

- WebDriver
  - Do not require selenium server for running test.
  - Using native automation from each and every supported language for running automation scripts on browsers.
  - Supports web as well as mobile application testing so you can test mobile applications (iPhone or Android).
  - Supporting latest versions of almost all browsers.
  - Controls the browser itself.
- Remote Control (Selenium 1)
  - Requires selenium server for running test.
  - Using JavaScript to drive automation with browser.
  - Supports only web application testing.
  - Supporting all browsers but not supporting latest versions.
  - Selenium RC is using JavaScript to interact and operate on web page.

Example SeleniumGoogleSuggest

- Principles
  - Maintain a code repository
  - Automate the build
  - Make the build self-testing
  - Support for team development
  - Every commit should be built and tested
  - Everyone can see test results
- Initial setup time required
  - Hudson `https://www.eclipse.org/hudson/`
    - Hudson project was transferred from Oracle to the Eclipse Foundation.
    - Development on the Eclipse Hudson project has ceased. `https://projects.eclipse.org/projects/technology.hudson`
  - Jenkins `https://jenkins.io/`
    - The project was forked from Hudson after Oracle bought Sun.
    - Jenkins has most of the original Hudson core developers.

- Easy installation (Java 11 required)
  - java -jar jenkins.war
- Configuration
  - Web GUI
  - Setting build scripts, testing, reporting
- Repository change support
- RSS/Email/IM integration
- Junit/TestNG
- Distributed builds
- Plugin support

References

- JUnit
  - http://www.tutorialspoint.com/junit/
- Selenium
  - http://www.tutorialspoint.com/selenium/
- Arquillian
  - http://arquillian.org/guides/getting_started/
- ShrinkWrap
  - http://arquillian.org/guides/shrinkwrap_introduction/
- Jenkins
  - https://jenkins.io/doc/tutorials/
- Java EE
  - http://docs.oracle.com/javaee/6/tutorial/doc/
- Migration to Java 11
  - https://winterbe.com/posts/2018/08/29/migrate-maven-projects-to-java-11-jigsaw/

Maven

Content

- Introduction to Maven
- Project object model
- Lifecycle
- Repositories and dependency resolution
- Creation of maven project
- Plugins

Build tools

- Scripting
  - Ant
  - Gradle
- Artifact oriented
  - Maven
  - Debian packaging (debhelper)

- Project management tool
  - more than just a build tool
  - platform
- Dependency management
- Building
- Documentation
- Project lifecycle
- Convention over configuration
  - reasonable defaults
- Common interface
  - `mvn install`

Maven project structure

```
foo/                        // Project root
  pom.xml                   // Project object model
  src/
    main/                   // Project sources
      java/                 // Project Java sources
      resources/
    test/                   // Test sources
  target/                   // Build output
    classes/
    foo.jar
```

Maven project structure – web

```
foo/                          // Project root
  pom.xml                     // Project object model
  src/
    main/                     // Project sources
      java/                   // Project Java sources
      webapp/
        META-INF/
        WEB-INF/
    test/                     // Test sources
  target/                     // Build output
    classes/
    foo/
      META-INF/
      WEB-INF/
    foo.war
```

- Basic information (groupId, artifactId, version, . . .)
- Dependencies
- Build
- Reporting (documentation)

- Default lifecycle – deployment (build and deploy into shared repository)
- Site lifecycle – create and deploy documentation
- Clean lifecycle – clean project (remove build outputs)

- Lifecycle phases
  - `mvn <phase>`
  - When you invoke a phase, Maven will go through all phases until specified one
  - `validate` – validate the project is correct and all necessary information is available
  - `compile` – compile the source code of the project
  - `test` – test the compiled source code using a unit testing framework
  - `package` – take the compiled code and package it
  - `verify` – run checks on results of integration tests to ensure quality
  - `install` – install the package into the local repository
  - `deploy` – copies the final package to the remote repository for sharing
  - `clean` – remove all files generated by the previous build

All lifecycle phases of `default` lifecycle (handles project deployment):

- `validate`
- `initialize`
- `generate-sources`
- `process-sources`
- `generate-resources`
- `process-resources`
- `compile`
- `process-classes`
- `generate-test-sources`
- `process-test-sources`
- `generate-test-resources`
- `process-test-resources`
- `test-compile`
- `process-test-classes`
- `test`
- `prepare-package`
- `package`
- `pre-integration-test`
- `integration-test`
- `post-integration-test`
- `verify`
- `install`
- `deploy`

- Site Lifecycle (handles the creation of project's site documentation)
  - `pre-site` – execute processes needed prior to the actual project site generation
  - `site` – generate the project's site documentation
  - `post-site` – execute processes needed to finalize the site generation, and to prepare for site deployment
  - `site-deploy` – deploy the generated site documentation to the specified web server using Maven Wagon (transport abstraction)
- Clean Lifecycle (project cleaning)
  - `pre-clean` – execute processes needed prior to the actual project cleaning
  - `clean` – remove all files generated by the previous build
  - `post-clean` – execute processes needed to finalize the project cleaning

- `MAVEN_OPTS` environment variable
  - contains parameters used to start up the JVM running Maven and can be used to supply additional options to globally to Maven (e.g.: `-Xms256m -Xmx512m`).
- settings.xml
  - located in `${maven.home}/conf/` or `${user.home}/.m2` the settings files is designed to contain any configuration for Maven usage across projects (`localRepository`, `servers`, `mirrors`, `proxies`, `profiles` – e.g. Windows/Linux, ...).
  - `/etc/maven/settings.xml`
- `.mvn` folder:
  - located within the projects top level folder, the files `maven.config` and `extensions.xml` contains project specific configuration for running Maven.

- Plugin is a Java class with Maven specific metadata
- Plugin can define goals and reports
- Goals can be called directly

  `mvn <plugin>:<goal>`
- Goals can be bound to lifecycle phase
- Phases and goals may be executed in sequence

  `mvn clean compile assembly:single`

  `mvn clean dependency:copy-dependencies package`
- Reports are used during site generation

  `mvn surefire-report:report` – parses the generated `TEST-*.xml` files and renders them using DOXIA, which creates the web interface version of the test results

Maven repository

- Local repository
  - contains all downloaded dependencies
  - by default in `~/.m2`
  - can be changed in `/etc/maven/settings.xml`
  - can be changed by command line argument
    `-Dmaven.repo.local=../repo/`
- Central repository
  - `http://repo1.maven.org/maven2`
  - contains common software, always available
- Remote repository
  - for more specific software
  - must be declared in `pom.xml`

```
<repositories>
  <repository>
    <id>java.net</id>
    <url>https://maven.java.net/content/repositories/public/</url>
  </repository>
</repositories>
```

Maven dependency resolution

- Priorities of dependency searching
  - Local repository
  - Central repository
  - Remote repository

Maven dependency resolution (2)

- Transitive dependencies are solved

```
<dependencies>
  <dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.20.0</version>
  </dependency>
</dependencies>
```

- List transitive dependencies
  - `mvn dependency:tree`

Maven dependency resolution (3)

- Dependency scopes
  - `compile` – default scope, compile dependencies are available in all classpaths of a project. Furthermore, those dependencies are propagated to dependent projects.
  - `provided` – like `compile`, but indicates you expect the JDK or a container to provide the dependency at runtime. This scope is only available on the compilation and test classpath, and is not transitive.
  - `runtime` – This scope indicates that the dependency is not required for compilation, but is for execution. It is in the runtime and test classpaths, but not the compile classpath.
  - `test` – This scope indicates that the dependency is not required for normal use of the application, and is only available for the test compilation and execution phases. This scope is not transitive.
  - `system` – This scope is similar to `provided` except that you have to provide the JAR which contains it explicitly. The artifact is always available and is not looked up in a repository.
  - `import` – It indicates the dependency to be replaced with the effective list of dependencies in the specified POM's.

Including custom library

- When required library is not in any repository

```
mvn install:install-file
-Dfile=/path/to/jar/<library>-<version>.jar
-DgroupId=<groupId>
-DartifactId=<library> -Dversion=<version> -Dpackaging=jar

mvn install:install-file -Dfile=./MyComponent-1.0-SNAPSHOT.jar
-DgroupId=cz.vutbr.fit -DartifactId=MyComponent
-Dversion=1.0-SNAPSHOT -Dpackaging=jar
```

- After this, dependency is available through local repository

```
<dependencies>
  <dependency>
    <groupId>cz.vutbr.fit</groupId>
    <artifactId>MyComponent</artifactId>
    <version>1.0-SNAPSHOT</version>
  </dependency>
</dependencies>
```

- Project templates
- `mvn archetype:generate`
  - Lists hundreds of possibilities of preconfigured projects

Creating maven project

- Desktop application
  - `mvn archetype:generate -DgroupId=project-packaging -DartifactId=project-name -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false`
- Web application
  - `mvn archetype:generate -DgroupId=project-packaging -DartifactId=project-name -DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false`
- In NetBeans simply open it
- Convert project to eclipse (retired – use `https://www.eclipse.org/m2e/` instead)
  - `mvn eclipse:eclipse`

- Custom functionality

```
<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.11.0</version>
    <configuration>
      <source>${jdk.version}</source>
      <target>${jdk.version}</target>
    </configuration>
  </plugin>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>3.4.0</version>
  </plugin>
  <plugin>
    <groupId>org.mortbay.jetty</groupId>
    <artifactId>jetty-maven-plugin</artifactId>
    <version>${jetty.version}</version>
  </plugin>
</plugins>
```

- Ability to compile project, deploy it on runtime, . . .

References

- http://www.tutorialspoint.com/maven/
- https://maven.apache.org

JAX

- JAX-RS
  - Java™ API for (XML) RESTful Web Services
- JAX-WS
  - Java™ API for XML-Based Web Services

JAX-RS

- What is REST
  - Representational State Transfer
    - Resource is representing the application state.
    - Operation leads to state transition.
  - Architectural style based on HTTP
  - Everything is a resource
  - Stateless operations
  - Typically uses client/server model
  - Handles different resource representations
    - HTML, XML, JSON, plain text, . . .

- GET retrieves a representation of a resource. Defines a reading access of the resource without side-effects. The resource is never changed via a GET request.
- HEAD identical to a GET except that no message body is returned in the response.
- POST creates a new resource to an existing URL. Can be used also for update.
- PUT creates a new resource to a new URL, or modify an existing resource to an existing URL.
- DELETE removes the existing resource.

- Java defines REST via JSR (Java Specification Request) 311
  - which is called JAX-RS (Java API for RESTful Services)
  - updated by JSR 339 – JAX-RS 2.0
- Jersey is the reference implementation
  - contains REST server and REST client
- Server side
  - Jersey servlet scans predefined classes
  - Registration in `web.xml`
  - Base URL

    `http://hostname:`
    `port/context-root/url-pattern/path-of-rest-class`
- Paths specified via annotations

- Jersey has to be registered as the servlet dispatcher for the REST services.
- Where to look for server classes
  - `jersey.config.server.provider.packages` defines package with server definitions in Jersey 2.x
  - `com.sun.jersey.config.property.package` in Jersey 1.x
- Root path specification
  - `context-root` (`glassfish-web.xml`)
- Everything is configured in `web.xml`

web.xml example Jersey 2.x

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <display-name>server</display-name>
  <servlet>
    <servlet-name>myrest</servlet-name>
    <servlet-class>
      org.glassfish.jersey.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>jersey.config.server.provider.packages</param-name>
      <param-value>cz.vutbr.fit.knot.gja.JM.server</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>myrest</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

web.xml example Jersey 1.x

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
  <display-name>server</display-name>
  <servlet>
    <servlet-name>myrest</servlet-name>
    <servlet-class>
      com.sun.jersey.spi.container.servlet.ServletContainer
    </servlet-class>
    <init-param>
      <param-name>com.sun.jersey.config.property.packages</param-name>
      <param-value>cz.vutbr.fit.knot.gja.JM.server</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>myrest</servlet-name>
    <url-pattern>/rest/*</url-pattern>
  </servlet-mapping>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
</web-app>
```

JAX-RS annotations

Annotation	Description
@Path(your_path)	Sets the path to base URL + /your_path.
@POST	Method will answer to POST request
@GET	Method will answer to GET request
@PUT	Method will answer to PUT request
@DELETE	Method will answer to DELETE request
@Produces(MediaType.ABC)	Response of method is of ABC type
@Consumes(MediaType.ABC)	Works with ABC type
@MatrixParam(variable)	Associative parameters delimited by ;
@PathParam(variable)	Variables separated by /
@FormParam(variable)	Form variables
@Context	Application, HttpHeaders, Request, . . . ServletConfig, UriInfo, ServletContext, HttpServletRequest and HttpServletResponse
@FormDataParam	Uploaded file

Jersey example

```
/**
 * Root resource (exposed at "myresource" path)
 */
@Path("myresource")
public class MyResource {

    /**
     * Method handling HTTP GET requests. The returned object will
     * be sent to the client as "text/plain" media type.
     *
     * @return String that will be returned as a text/plain response.
     */
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String getIt() {
        return "Got it!";
    }
}
```

Examples JerseyHelloWorld, JerseyPath

Form parameters

- `@FormParam` annotation
- Form submit needed

```
@POST
@Produces(MediaType.TEXT_HTML)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public void newTodo(@FormParam("id") String id,
    @FormParam("summary") String summary,
    @FormParam("description") String description,
    @Context HttpServletResponse servletResponse)
        throws IOException {

  // method body

  servletResponse.sendRedirect("../index.html");
}
```

Example JerseyCRUD

Matrix parameters

- GET parameters separated by `;`
- Call example
  - `../mapping;year=1234;country=CZ;author=Me`

```
@GET
public Response getBooks(@MatrixParam("year") String year,
    @MatrixParam("author") String author,
    @MatrixParam("country") String country) {

    //method body

    return Response.status(200).entity("Jersey servlet
      called my method with following parametres
      - year : " + year + ", author : " + author
      + ", country : " + country).build();
}
```

Example JerseyMatrix

Path parameters

- Parameters separated by `"/"`
- Call example
  - `../mapping/2001/11/12`

```
@GET
@Path("{year}/{month}/{day}")
public Response getUserHistory(
    @PathParam("year") int year,
    @PathParam("month") int month,
    @PathParam("day") int day) {

    String date = year + "/" + month + "/" + day;
    return Response.status(200).entity("Jersey
    servlet called my method with following
    parameters, year/month/day : " + date).build();

}
```

Example JerseyPathParam

File upload

- Jersey multipart
- InputStream with file data
- `FormDataContentDisposition` is optional

```
@Context ServletContext servletContext;
@POST
@Path("/upload")
@Consumes(MediaType.MULTIPART_FORM_DATA)
@Produces(MediaType.TEXT_PLAIN)
public Response uploadFile(
  @FormDataParam("file") InputStream uploadedInputStream,
  @FormDataParam("file") FormDataContentDisposition fileDetail) {

  String fileName = fileDetail.getFileName();
  String path = servletContext.getRealPath(fileName);

  // save it
  writeToFile(uploadedInputStream, path);

  String output = "File uploaded to : " + path + " and have "
    + Long.toString(fileDetail.getSize()) + "bytes";
  return Response.status(200).entity(output).build();
}
```

File download

```
@Context ServletContext servletContext;

@GET
@Path("/get")
@Produces("image/png")
public Response getFile() {

  File file = new File(servletContext
      .getRealPath("image_on_server.png"));

  ResponseBuilder response = Response.ok((Object) file);
  response.header("Content-Disposition",
      "attachment; filename=image_on_server.png");
  return response.build();
}
```

Example JerseyUpload

- Java™ Architecture for XML Binding (JAXB)
  - JSR 222
- Object to XML mapping
- Root element specification
  - `@XmlRootElement`
  - Data object
- Return type specification
  - `MediaType.TEXT_XML`
  - `MediaType.APPLICATION_XML`
  - `MediaType.APPLICATION_JSON`
- Object is returned from response function.
- Object is automatically converted to return type specified by annotation.

JAXB example

```
@XmlRootElement
public class Product {

  private String name;
  private double price;

  public Product() {
  }

  public Product(String name, double price) {
    this.name = name;
    this.price = price;
  }
  ...
}

JAXBContext jc = JAXBContext.newInstance(Product.class);
Marshaller m = jc.createMarshaller();
m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, true);
OutputStream os = new FileOutputStream("product.xml");
m.marshal(new Product("Káva", 20.5d), os);

Unmarshaller u = jc.createUnmarshaller();
Object p = u.unmarshal(new File("product.xml"));
```

References

- http://www.mkyong.com/tutorials/jax-rs-tutorials/
- https://www.vogella.com/tutorials/REST/article.html
- https://jersey.java.net/documentation/latest/getting-started.html
- https://docs.oracle.com/cd/F28299_01/pt857pbr3/eng/pt/tibr/concept_UnderstandingRESTServiceOperations.html
- https://jcp.org/en/jsr/detail?id=339
- https://allegro.tech/2014/10/async-rest.html
- https://www.jesperdj.com/2018/09/30/jaxb-on-java-9-10-11-and-beyond/

JAX-WS

- Java™ API for XML-Based Web Services (JAX-WS)
  - JSR 224
  - extends the existing JAX-RPC 1.0 specification with new features
- What are Web Services
  - Services and clients communicates via XML
  - Message or RPC oriented approach
  - Web service operation invocation is represented by SOAP (Simple Object Access Protocol)
  - SOAP messages (XML files) are sent via HTTP

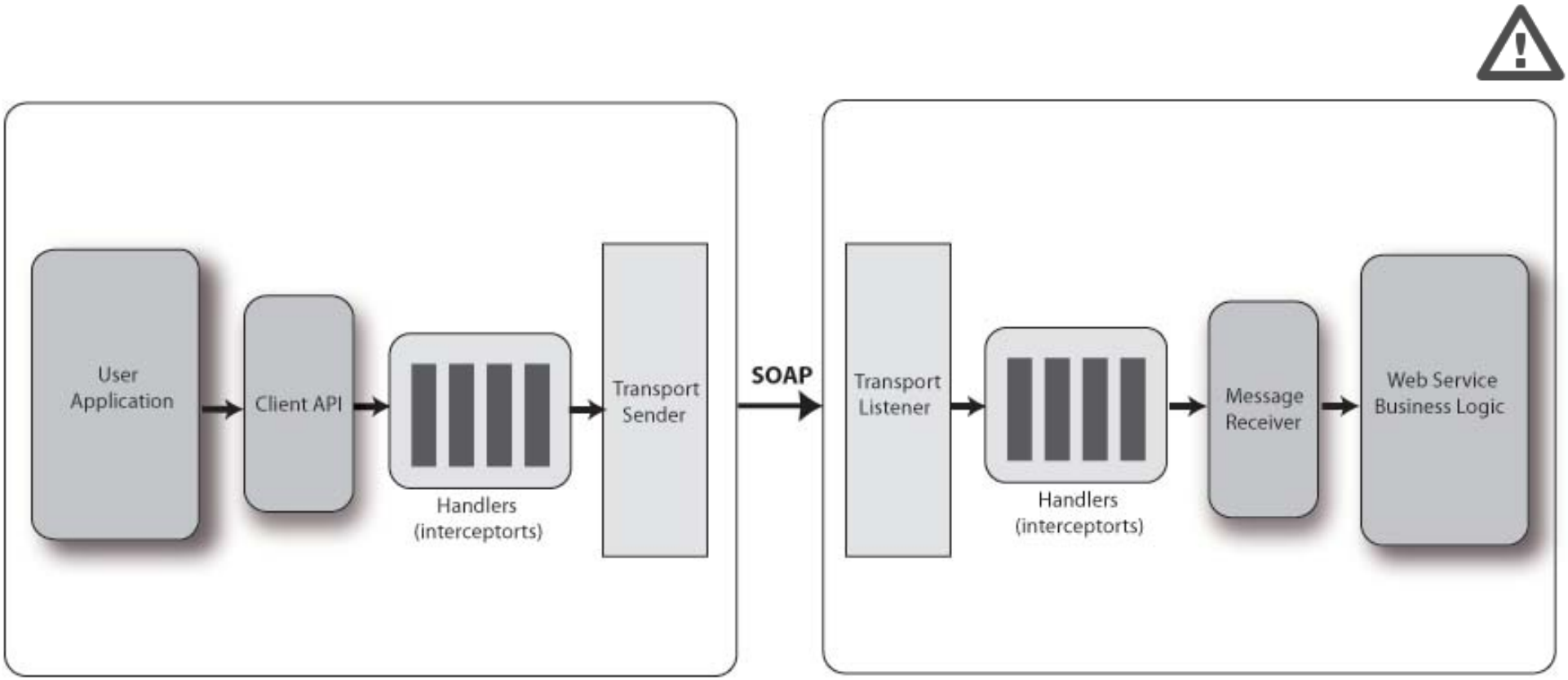
- XML-based protocol
- Defines
  - envelope structure
  - encoding rules
  - conventions for representing web service invocations and responses
- Web methods
  - GET retrieves a representation of a resource.
  - POST updates an existing resource or creates a new resource.
  - PUT creates a new resource or modify an existing resource.
  - DELETE removes the resources.
- Server side
  - Java interface (created later by client as a proxy)
  - defining methods of interface
- Client side
  - create proxy
  - call proxy methods (these are invoked on server)

- A SOAP message is an XML document containing the following elements:
  - an `Envelope` element that identifies the XML document as a SOAP message
  - a `Header` element that contains header information
  - a `Body` element that contains call and response information
  - a `Fault` element containing errors and status information

SOAP Message

```
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <macAddress xmlns="http://ws.fit/"
      xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
      SOAP-ENV:actor="http://schemas.xmlsoap.org/soap/actor/next">
      E4-11-5B-F2-54-AA
    </macAddress>
  </S:Header>
  <S:Body>
    <ns2:getServerName xmlns:ns2="http://ws.gja.knot.fit.vutbr.cz/"/>
  </S:Body>
</S:Envelope>
```

JAX-WS architecture



- Implementations
  - Apache AXIS
  - Glassfish Metro (reference implementation)

- Glassfish Metro
  - Endpoint specified in `sun-jaxws.xml`
  - `web.xml`
    - `WSServletContextListener` – listener class
    - `WSServlet` – web service servlet
  - `@WebService`
  - `@WebMethod`

```
<?xml version="1.0" encoding="UTF-8"?>
<endpoints
  xmlns="http://java.sun.com/xml/ns/jax-ws/ri/runtime"
  version="2.0">
  <endpoint
    name="HelloWorldWs"
    implementation="cz.vutbr.fit.knot.gja.example.HelloWorld"
    url-pattern="/hello"/>
</endpoints>
```

```
<web-app>
  <display-name>Archetype Created Web Application</display-name>
  <listener>
    <listener-class>
      com.sun.xml.ws.transport.http.servlet.WSServletContextListener
    </listener-class>
  </listener>
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>
      com.sun.xml.ws.transport.http.servlet.WSServlet
    </servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Example JAX-WS-Web

- Endpoints
  - RPC (Remote Procedure Call) style web service endpoint
  - Document style web service endpoint

- Synchronous
- Easy to implement
- SOAP engine takes care of marshalling
- Poor performance
- Interface annotation
  - `@WebService`
  - `@SOAPBinding(style = Style.RPC)`
- Implementation
  - `@WebService(endpointInterface = "package.Interface")`
- Accessible on URL commonly named by package
  - QName defines target service endpoint
  - Communication via service port

JAX-WS RPC example

```
// Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.RPC)
public interface HelloWorld{
    @WebMethod String getHelloWorldAsString(String name);
}

// Service Implementation
@WebService(endpointInterface = "cz.vutbr.fit.knot.gja.ws.HelloWorld")
public class HelloWorldImpl implements HelloWorld{
    @Override
    public String getHelloWorldAsString(String name) {
        return "Hello World JAX-WS " + name;
    }
}

// Endpoint publisher
public class HelloWorldPublisher{
    public static void main(String[] args) {
        Endpoint.publish("http://localhost:9999/ws/hello",
            new HelloWorldImpl());
    }
}
```

JAX-WS RPC example (2)

- It is possible to access deployed web service by accessing the generated WSDL (Web Service Definition Language) document via this URL
  `http://localhost:9999/ws/hello?wsdl`

```
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
...

public class HelloWorldClient {
  public static void main(String[] args) throws Exception {
    URL url = new URL("http://localhost:9999/ws/hello?wsdl");
    // 1st argument is service URI, refer to wsdl document above
    // 2nd argument is service name, refer to wsdl document above
    QName qname = new QName("http://ws.gja.knot.fit.vutbr.cz/",
        "HelloWorldImplService");

    Service service = Service.create(url, qname);
    HelloWorld hello = service.getPort(HelloWorld.class);
    System.out.println(hello.getHelloWorldAsString("fit"));
  }
}
```

JAX-WS RPC example (3)

- Alternative, you can use `wsimport` tool to parse the published wsdl file, and generate necessary client files (stub) to access the published web service.

```
wsimport -keep http://localhost:9999/ws/hello?wsdl
```

```
import com.mkyong.ws.HelloWorld;
import com.mkyong.ws.HelloWorldImplService;

public class HelloWorldClient{
  public static void main(String[] args) {
    HelloWorldImplService helloService = new HelloWorldImplService();
    HelloWorld hello = helloService.getHelloWorldImplPort();

    System.out.println(hello.getHelloWorldAsString("fit"));
  }
}
```

- Used for implementing asynchronous service
- More time consuming to create
- Large size documents processed without significant performance drop
- Better custom data type definition
- Interface annotation
  - `@WebService`
  - `@SOAPBinding(style = Style.DOCUMENT, use=Use.LITERAL)`
  - `SOAPBinding.Use.ENCODED` – SOAP message contains data type information
  - `SOAPBinding.Use.LITERAL` – SOAP message contains reference (namespace) to the schema that is used
- Otherwise similar as RPC

JAX-WS Document example

```
// Service Endpoint Interface
@WebService
@SOAPBinding(style = Style.DOCUMENT, use=Use.LITERAL) // optional
public interface HelloWorld {
  @WebMethod String getHelloWorldAsString(String name);
}

// Service Implementation
@WebService(endpointInterface = "cz.vutbr.fit.knot.gja.ws.HelloWorld")
public class HelloWorldImpl implements HelloWorld{
  @Override
  public String getHelloWorldAsString(String name) {
    return "Hello World JAX-WS " + name;
  }
}
// Endpoint publisher
public class HelloWorldPublisher{
  public static void main(String[] args) {
    Endpoint.publish("http://localhost:9999/ws/hello",
      new HelloWorldImpl());
  }
}
```

JAX-WS Document example (2)

- Document style requires extra classes to run. You need to use `wsgen` tool to generate necessary JAX-WS portable artifacts.

```
wsgen -keep -cp . cz.vutbr.fit.knot.gja.ws.HelloWorld
```

  - It will generate two classes, copy it to your `package.jaxws` folder.
- Publish it and test it via URL `http://localhost:9999/ws/hello?wsdl`

```
import javax.xml.namespace.QName;
import javax.xml.ws.Service;
...

public class HelloWorldClient{
  public static void main(String[] args) throws Exception {

    URL url = new URL("http://localhost:9999/ws/hello?wsdl");
    QName qname = new QName("http://ws.gja.knot.fit.vutbr.cz/",
                            "HelloWorldImplService");

    Service service = Service.create(url, qname);
    HelloWorld hello = service.getPort(HelloWorld.class);

    System.out.println(hello.getHelloWorldAsString("fit"));
  }
}
```

Example JAX-WS-Document

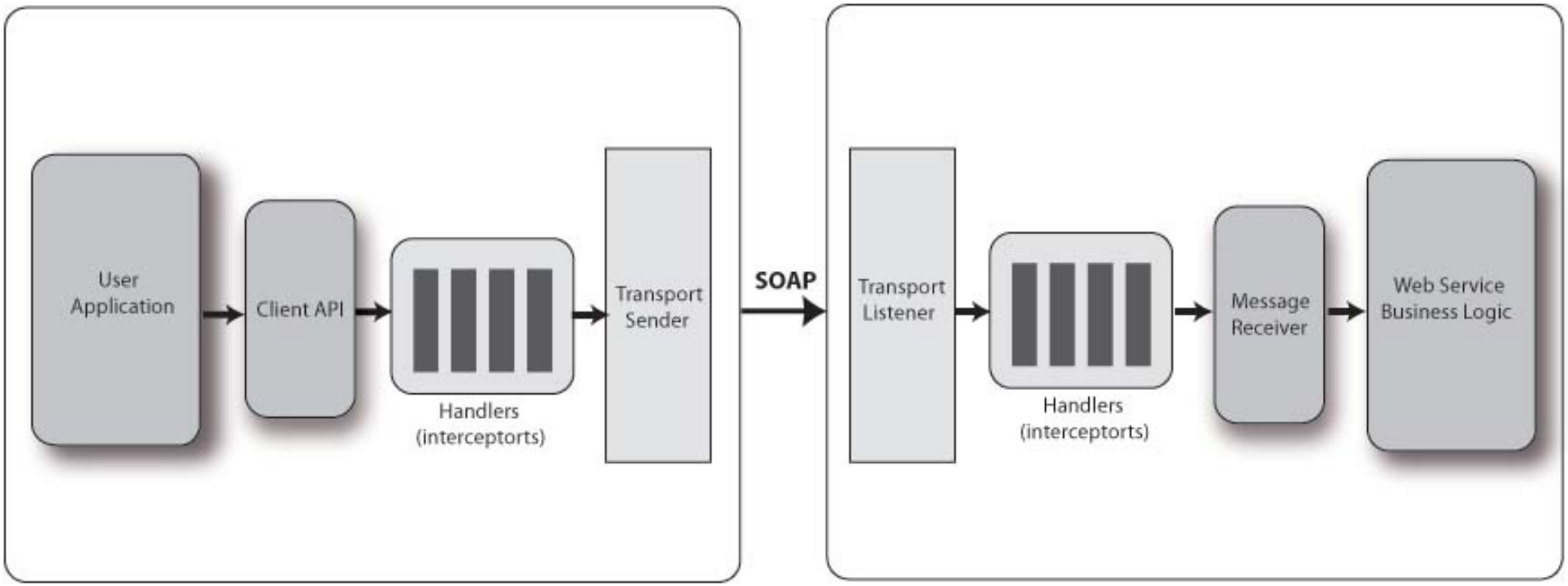
- Message Transmission Optimization Mechanism
- Sending binary attachments
- Combines Base64 with SOAP
- File upload
- File download
- MIME types

```
// Service Implementation Bean
@MTOM
@WebService(endpointInterface = "cz.vut.fit.knot.gja.ws.ImageServer")
public class ImageServerImpl implements ImageServer{
  @Override
  public Image downloadImage(String name) {
...
```

- If the server WSDL advertises that it supports MTOM, the MTOM support in the client will be automatically enabled.

Example JAX-WS-MTOM

JAX-WS architecture



Web Service handlers

- Client handlers
  - intercept client calls to server
  - injects authentication
  - adds MAC, IP address, ...
- Server interceptors
  - retrieves information from SOAP header block
  - authorizes client

Client handler

- Create the client for the web service
- Create SOAP Handler
- Handler Configuration
- Handler Mapping Tracing the outgoing and incoming messages in success and failure case
- SOAPHandler interface

```
public class MySOAPHandler
    implements SOAPHandler<SOAPMessageContext> {
  @Override
  public boolean handleMessage(SOAPMessageContext
      messagecontext) {
    Boolean outbound = (Boolean) messagecontext.get(
      MessageContext.MESSAGE_OUTBOUND_PROPERTY);
    ...
    SOAPMessage message = messagecontext.getMessage();
    SOAPHeader header = message.getSOAPHeader();
    SOAPBody body = message.getSOAPBody();
    ...
```

- handleMessage returns true to continue processing, false to block processing.

Client handler (2)

- Interceptor mapping specified in XML file
  - `@HandlerChain`
  - More mapping possible

```
@WebServiceClient(name = "ServerInfoService",
  targetNamespace = "http://ws.gja.knot.fit.vutbr.cz/",
  wsdlLocation = "http://localhost:8888/ws/server?wsdl")
@HandlerChain(file="handler-client.xml")
public class ServerInfoService extends Service
{
  // ...
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jakartaee:handler-chains
    xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <jakartaee:handler-chain>
    <jakartaee:handler>
      <jakartaee:handler-class>
      cz.vutbr.fit.knot.gja.ws.client.MacAddressInjectHandler
      </jakartaee:handler-class>
    </jakartaee:handler>
  </jakartaee:handler-chain>
</jakartaee:handler-chains>
```

Server handler

- Create Web Service
- Create SOAP Handler
- Handler Configuration
- Handler Mapping
- SOAPHandler interface
  - `SOAPMessageContext`
  - `handleMessage` returns true or false (chain may be broken)

Server handler (2)

- Intercept mapping specified in XML file
  - `@HandlerChain`
  - More mapping possible

```
@WebService
@HandlerChain(file="handler-server.xml")
public class Server{
  @WebMethod
  public String getServerName() {
    return "localhost server";
  }
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jakartaee:handler-chains
    xmlns:jakartaee="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <jakartaee:handler-chain>
    <jakartaee:handler>
      <jakartaee:handler-class>
   cz.vutbr.fit.knot.gja.ws.server.MacAddressValidatorHandler
      </jakartaee:handler-class>
    </jakartaee:handler>
  </jakartaee:handler-chain>
</jakartaee:handler-chains>
```

Example JAX-WS-Handler

- JAX-WS
  - http://www.mkyong.com/tutorials/jax-ws-tutorials/
  - https://www.mkyong.com/webservices/jax-ws/jax-ws-java-web-application-integration-example/
  - https://www.mkyong.com/webservices/jax-ws/jax-ws-soap-handler-in-client-side/
  - https://www.mkyong.com/webservices/jax-ws/jax-ws-soap-handler-in-server-side/
  - https://web.archive.org/web/20211023024040/http://blog.jdevelop.eu/?p=67
  - https://metro.java.net/nonav/1.2/docs/
  - https://docs.oracle.com/javaee/7/api/
  - https://jcp.org/en/jsr/detail?id=224
  - http://javainsimpleway.com/jax-ws-basic-example-document-style/
- SOAP
  - https://www.w3.org/TR/soap12-part0/
  - https://www.w3schools.com/xml/xml_soap.asp
  - https://www.ibm.com/support/knowledgecenter/en/SSB27H_6.2.0/fa2ws_ovw_soap_syntax_lit.html

Thank you for your attention!