Intercepteurs et services Web

Illustration à travers JAX-WS et CXF

Emmanuel Coquery

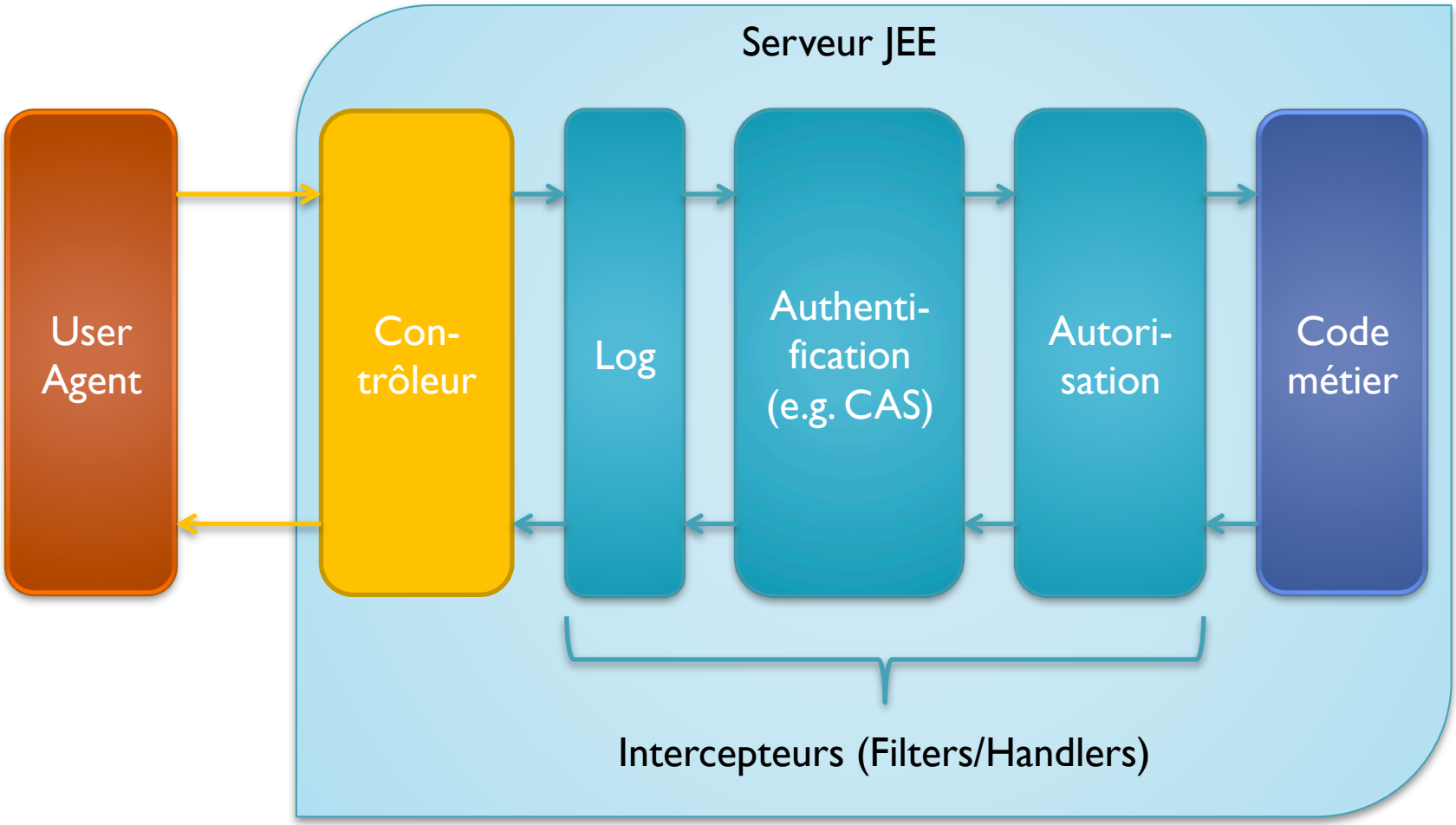
Problématique

- Séparer les aspects fonctionnels et non-fonctionnels
- Factoriser les code de traitement non-fonctionnel
  - Voir faire appel à des bibliothèques tierces pour certains traitements standards
- Configurer facilement les traitements à appliquer sur telle ou telle opération métier
  - Permettre des configurations spécifiques à tel ou tel déploiement

Interception: principe

- Appeler du code avant/après le traitement métier
- Pouvant
  - Modifier la requête/réponse
  - Empêcher l'exécution du code métier
- Similaire à:
  - Programmation orientée aspects basique
  - Routines RPC

Exemple


Serveur JEE
User Agent
Con-trôleur
Log
Authenti-fication (e.g. CAS)
Autori-sation
Code métier
Intercepteurs (Filters/Handlers)

JEE: avec les servlets

- Intercepteur = classe
  - Implémentant l'interface Filter
- Chaîne d'intercepteurs
  - Instance de FilterChain
    - Configurée dans le conteneur de servlets
- Filter: doFilter(request,response,chain)
  - Défini les traitements pré/post métier
    - Peuvent modifier request et/ou response
  - Appel explicite à la suite de la chaîne via
    - chain.doFilter(request,response)
    - Pas d'appel = non-exécution du code métier

Exemple: authentification CAS

```
package tiw5.exemple;

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class CASFilter implements Filter {

        private CASStub cas; // objet codant les interactions avec le CAS

        public void init(FilterConfig cfg) throws ServletException {
                cas = new CASStub(cfg.getInitParameter("cas.endpoint.url"));
        }

        public void destroy() {
                cas.close();
                cas = null;
        }
```

Exemple - suite

```
public void doFilter(ServletRequest req, ServletResponse resp,
                FilterChain chain) throws IOException, ServletException {
        HttpServletResponse httpResp = (HttpServletResponse)resp;
        HttpServletRequest httpReq = (HttpServletRequest) req;
        String ticket = req.getParameter("ticket");
        if (ticket == null) {
                httpResp.sendRedirect(cas.getRedirectURL
                                (httpReq.getRequestURL().toString()));
        } else {
                String user = cas.getUserId(ticket);
                if (user == null) {
                        httpResp.sendError
                                (HttpServletResponse.SC_FORBIDDEN);
                } else {
                        httpReq.setAttribute("user", user);
                        chain.doFilter(req, resp); // suite du traitement
                }
        }
}
```

Handlers en JAX-WS

- Principe similaire aux Filters
  - Fonctionnement différent
    - Pas de contrôle direct sur l'exécution de la chaîne des handlers
    - Traitement par des threads possiblement différents entre le message d'entrée et celui de sortie
  - handleMessage(MessageContext):
    - Traite un message entrant ou sortant
      - c.f. MessageContext.MESSAGE_OUTBOUND_PROPERTY
    - Renvoie false pour interrompre le traitement

Gestion des erreurs

- handleFault(MessageContext) pour le traitement des erreurs:
  - fonctionnement similaire à handleMessage
- Déclenchement d'une erreur:
  - Changer le message en une faute
  - Lever ProtocolException

Chaînes d'intercepteurs

- Similaires aux chaînes de filtrage
- Ordre le la chaîne:
  - Inversé pour les messages entrants
  - Tel quel pour les messages sortant
  - Côté serveur
    - « Le dernier handler est le plus externe, le premier est le plus proche du métier »
    - C'est l'inverse côté client
- En cas de faute déclenchée par un handler, l'ordre est renversé
  - Attention aux déclenchement de fautes sur une réponse côté serveur

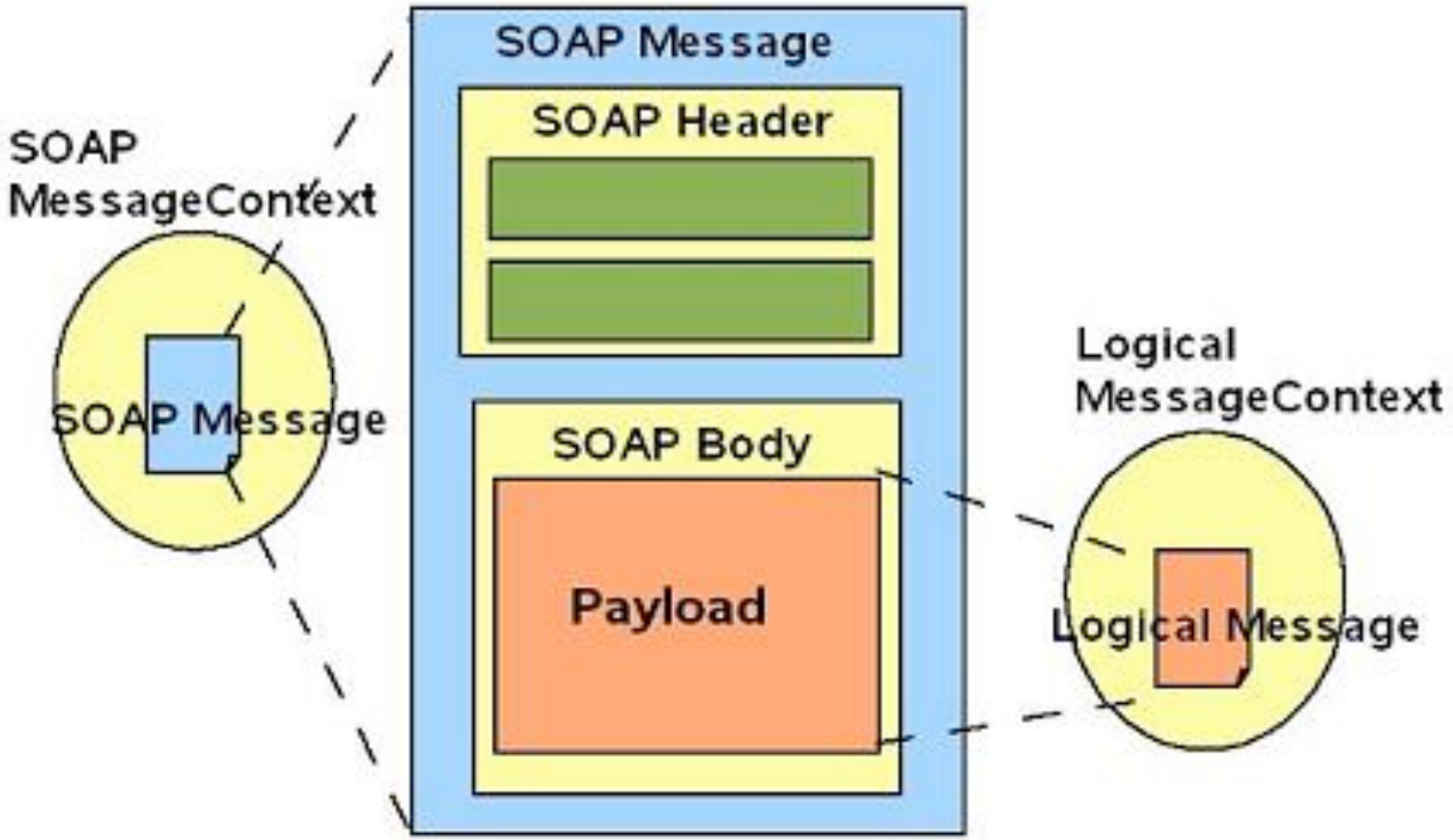
Contextes

- Objet contenant les informations du message
  - Contenu du message
  - Méta-données
    - e.g. header SOAP
    - MessageContext extends Map<String,Object>
- Un handler peut modifier le contexte
  - Ajout/suppression de méta-données
  - Modification du message
    - e.g. remplacement du message par une faute

Logical vs SOAP

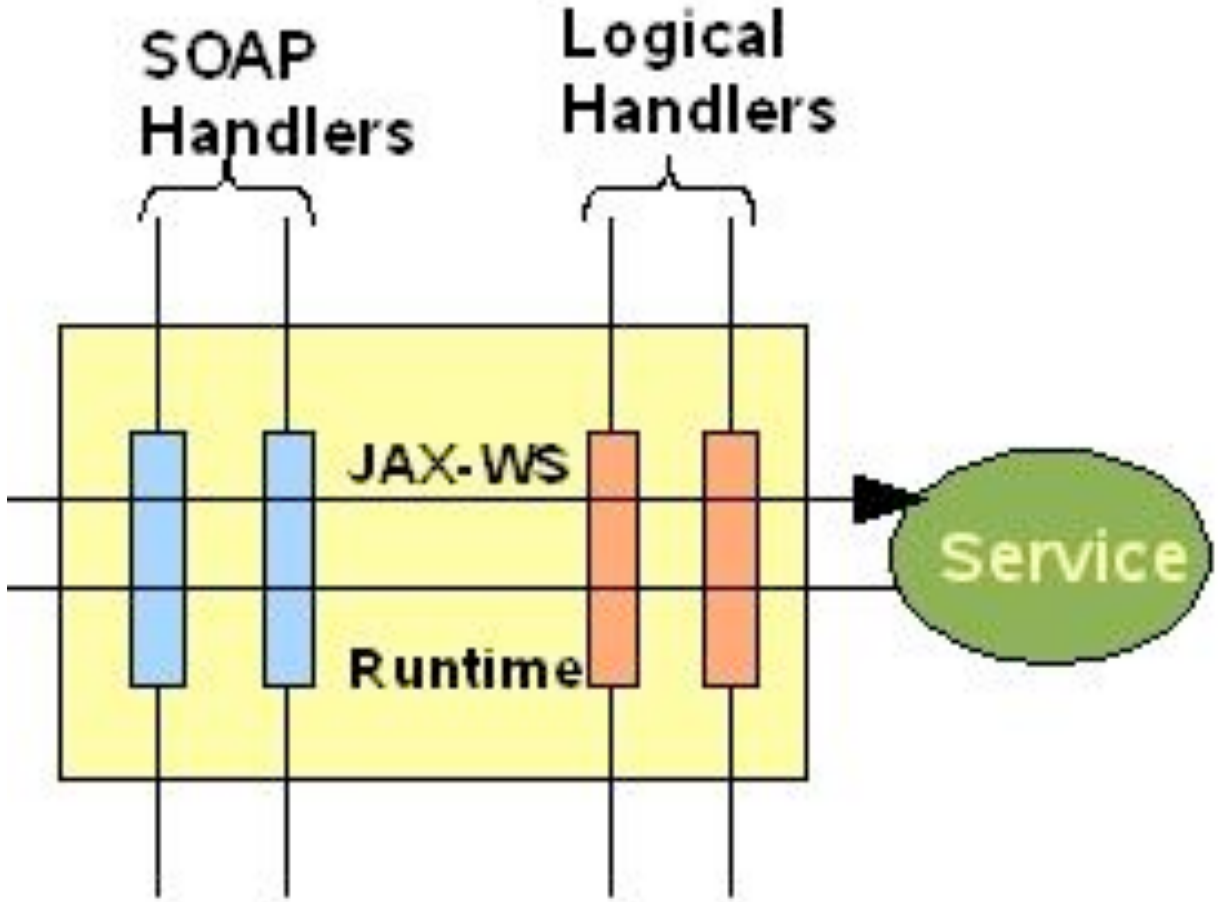
- Deux types de handlers standards
  - D'autres types possibles
- SOAP:
  - Porte sur tout le message
  - Lié au protocole SOAP
- Logical
  - Ne concerne que le payload
  - + les méta-données
    - Typiquement introduites lors de traitements précédents

Portée des différents types de contexte



Source: http://jax-ws.java.net/articles/handlers_introduction.html

Ordre d'exécution des handlers



Source: http://jax-ws.java.net/articles/handlers_introduction.html

Exemple: handler logique

```
package tiw5.exemple;
import javax.xml.ws.handler.LogicalHandler;
import javax.xml.ws.handler.LogicalMessageContext;
import javax.xml.ws.handler.MessageContext;
import ...

public class IntercepteurLogique implements LogicalHandler<LogicalMessageContext> {

        private static Logger log = Logger.getLogger("IntercepteurLogique");
        private Transformer copy;

        public IntercepteurLogique() {
                try {
                        copy = TransformerFactory.newInstance().newTransformer();
                } catch (TransformerConfigurationException e) {
                        throw new RuntimeException(e);
                }
        }
```

Handler logique - 2

```
public String getOp(LogicalMessageContext ctx) {
        Source payload = ctx.getMessage().getPayload();
        DOMResult dom = new DOMResult();
        try {
                copy.transform(payload, dom);
        } catch (TransformerException e) {
                throw new RuntimeException(e);
        }
        Node root = dom.getNode();
        if (root.getNodeType() == Node.DOCUMENT_FRAGMENT_NODE) {
                root=((DocumentFragment)root).getFirstChild();
        } else if (root.getNodeType() == Node.DOCUMENT_NODE) {
                root = ((Document)root).getDocumentElement();
        }
        return root.getNodeName();
}
```

Handler logique - 3

```
public void close(MessageContext arg0) {}

public boolean handleFault(LogicalMessageContext ctx) {
        log.warning("Erreur dans "+getOp(ctx));
        return true; // Le traitement continue
}

public boolean handleMessage(LogicalMessageContext ctx) {
        boolean sortant = (Boolean) ctx.get
                        (MessageContext.MESSAGE_OUTBOUND_PROPERTY);
        log.info("Message "+(sortant?"sortant":"entrant")
                                                +": "+getOp(ctx));
        return true; // Le traitement continue
}
```

Exemple: handler SOAP - 1

```
package tiw5.exemple;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlValue;
import javax.xml.namespace.QName;

@XmlRootElement(name="user",namespace="http://exemple.tiw5.univ-lyon1.fr")
public class User {

        public static final QName QNAME =
                        new QName("http://exemple.tiw5.univ-lyon1.fr", "user");

        @XmlValue
        public String name;
}
```

Handler SOAP - 2

```
package tiw5.exemple;
import java.util.HashSet;
import java.util.Set;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.namespace.QName;
import javax.xml.ws.handler.MessageContext;
import javax.xml.ws.handler.soap.SOAPHandler;
import javax.xml.ws.handler.soap.SOAPMessageContext;

public class IntercepteurSOAP implements SOAPHandler<SOAPMessageContext> {
          private static Set<QName> handled = new HashSet<QName>();
          static {
                    handled.add(User.QNAME);
          }

          public void close(MessageContext ctx) {         }

          public boolean handleFault(SOAPMessageContext ctx) { return true; }

          public Set<QName> getHeaders() {
                    return handled;
          }
```

Handler SOAP - 3

```
public boolean handleMessage(SOAPMessageContext ctx) {
        if ((Boolean) ctx.get(MessageContext.MESSAGE_OUTBOUND_PROPERTY)) {
                return true;
        } else {
                try {
                        Object [] users = ctx.getHeaders
                       (User.QNAME, JAXBContext.newInstance(User.class), true);
                        if (users.length > 0) {
                                ctx.put("user", users[0]);
                                return true;
                        } else {
                                return false;
                        }
                } catch (JAXBException e) {
                        e.printStackTrace();
                        return false;
                }
        }
}
```

Exemple de configuration CXF web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/
javaee/web-app_2_5.xsd">
    <display-name>StoreCalService</display-name>
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:m2ti/ti3/storecalservice/services.xml</param-value>
    </context-param>

    <listener><listener-class>org.springframework.web.context.ContextLoaderListener</listener-class></
listener>

    <servlet>
        <servlet-name>CXFServlet</servlet-name>
        <servlet-class>org.apache.cxf.transport.servlet.CXFServlet</servlet-class>
    </servlet>
    <servlet-mapping><servlet-name>CXFServlet</servlet-name><url-pattern>/services/*</url-pattern></
servlet-mapping>
…
</web-app>
```

Exemple de configuration CXF: services.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:jaxws="http://cxf.apache.org/jaxws">
<import resource="classpath:META-INF/cxf/cxf.xml"/>
<import resource="classpath:META-INF/cxf/cxf-extension-soap.xml"/>
<import resource="classpath:META-INF/cxf/cxf-servlet.xml"/>

                <bean id="ws" class="m2ti.ti3.storecalservice.StoreCalServiceImpl"/>
                <bean id="logHandler" class="m2ti.ti3.storecalservice.handlers.LogHandler" />
                <bean id="userHandler" class="m2ti.ti3.storecalservice.handlers.UserHandler" />
                <bean id="checkUserHandler" class="m2ti.ti3.storecalservice.handlers.UserCheckHandler" />

   <jaxws:endpoint id="storecalservice"
            implementor="#ws"
            address="/StoreCalService">
                <jaxws:handlers>
                                <ref bean="logHandler" />
                                <ref bean="checkUserHandler"/>
                                <ref bean="userHandler" />
                </jaxws:handlers>
   </jaxws:endpoint>
</beans>
```