eBook Gratuit

APPRENEZ cxf

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#cxf

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: cxf

It is an unofficial and free cxf ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official cxf.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Démarrer avec cxf

Remarques

Cette section fournit une vue d'ensemble de ce que cxf est et pourquoi un développeur peut vouloir l'utiliser.

Il devrait également mentionner tous les grands sujets dans cxf, et établir un lien avec les sujets connexes. La documentation de cxf étant nouvelle, vous devrez peut-être créer des versions initiales de ces rubriques connexes.

Examples

Webclient de base avec le fournisseur

Pour commencer, nous avons besoin d'une usine qui produit WebClients.

```
public class ClientFactory {
    private Map<String, WebClient> cache = new HashMap<>();

    public enum RESTClient {
        PORTAL;
    }

    public WebClient fetchRestClient(RESTClient restClient) {

        if (this.cache.containsKey(restClient)) {
            return WebClient.fromClient(this.cache.get(rc));
        }

        if (RESTClient.enum.equals(rc)) {

            List<Object> providers = new ArrayList<Object>();
            providers.add(new GsonMessageBodyProvider());

            WebClient webClient = WebClient.create("https://blah.com", providers);

            HTTPConduit conduit = WebClient.getConfig(webClient).getHttpConduit();
            conduit.getClient().setReceiveTimeout(recieveTimeout);
            conduit.getClient().setConnectionTimeout(connectionTimout);

            this.cache.put(RESTClient.CAT_DEVELOPER_PORTAL.name(), webClient);
            return WebClient.fromClient(webClient);// thread safe
        }
    }
}
```

- Nous commençons par créer une liste de fournisseurs (les consulterons plus tard)
- Nous créons ensuite un nouveau webclient en utilisant la fabrique statique "create ()". Ici, nous pouvons ajouter quelques autres choses comme les creds Auth Basic et la sécurité des threads. Pour l'instant, utilisez celui-ci.

- Ensuite, nous retirons HTTPConduit et définissons les délais d'attente. CXF est fourni avec les clients de classe de base Java, mais vous pouvez utiliser Glassfish ou d'autres.
- Enfin, nous mettons en cache le WebClient. Ceci est important car le code à ce jour est coûteux à créer. La ligne suivante montre comment le rendre threadsafe. Notez que c'est aussi la manière dont nous retirons le code du cache. Essentiellement, nous élaborons un modèle d'appel REST et le clonons chaque fois que nous en avons besoin.
- Notez ce qui n'est pas ici: Nous n'avons ajouté aucun paramètre ou URL. Celles-ci peuvent être ajoutées ici, mais cela créerait un point de terminaison spécifique et nous en voulons un générique. En outre, aucun en-tête n'a été ajouté à la demande. Celles-ci ne font pas passer le "clone", elles doivent donc être ajoutées plus tard.

Nous avons maintenant un WebClient prêt à l'emploi. Permet de configurer le reste de l'appel.

```
public Person fetchPerson(Long id) {
    long timer t = System.currentTimeMillis();
    Person person = null;
    try {
        wc = this.factory.findWebClient(RESTClient.PORTAL);
        wc.header(AUTH_HEADER, SUBSCRIPTION_KEY);
        wc.header(HttpHeaders.ACCEPT, "application/person-v1+json");

        wc.path("person").path("base");
        wc.query("id", String.valueOf(id));

        person = wc.get(Person.class);
    }
    catch (WebApplicationException wae) {
        // we wanna skip these. They will show up in the "finally" logs.
    }
    catch (Exception e) {
        log.error(MessageFormat.format("Error fetching Person: id:{0} ", id), e);
    }
    finally {
        log.info("GET HTTP:{} - Time:[{}ms] - URL:{} - Content-Type:{}",
wc.getResponse().getStatus(), (System.currentTimeMillis() - timer), wc.getCurrentURI(),
wc.getResponse().getMetadata().get("content-type"));
        wc.close();
    }
    return p;
}
```

- Dans le "try", nous récupérons un WebClient de l'usine. Ceci est un nouveau "glacial".
- Ensuite, nous définissons des en-têtes. Ici, nous ajoutons une sorte d'en-tête d'authentification, puis un en-tête d'acceptation. Notez que nous avons un en-tête d'acceptation personnalisé.
- Ajouter le chemin et les chaînes de requête vient ensuite. Gardez à l'esprit qu'il n'y a pas d'ordre à ces étapes.
- Enfin, nous faisons le "get". Il y a plusieurs façons de faire cela bien sûr. Ici, nous avons CXF faire le mappage JSON pour nous. Lorsque cela est fait de cette façon, nous devons traiter les WebApplicationExceptions. Donc, si nous obtenons un 404, CXF lancera une exception. Remarquez ici que je mange ces exceptions car je me contente de consigner la réponse dans le fichier final. Je souhaite toutefois obtenir une autre exception, car elles pourraient être importantes.

- Si vous n'aimez pas ce changement de gestion des exceptions, vous pouvez récupérer l'objet Response du "get". Cet objet contient l'entité et le code d'état HTTP. Il ne jettera JAMAIS une WebApplicationException. Le seul inconvénient est qu'il ne fait pas le mappage JSON pour vous.
- Enfin, dans la clause "finally", nous avons un "wc.close ()". Si vous obtenez l'objet de la clause get, vous n'avez pas vraiment à le faire. Quelque chose pourrait aller de travers si c'est une bonne sécurité.

Alors, qu'en est-il de cet en-tête "accept": application / person-v1 + json Comment CXF saura-t-il l'analyser? CXF est livré avec des analyseurs JSON intégrés, mais je voulais utiliser Gson. Beaucoup des autres implémentations des analyseurs Json ont besoin d'une sorte d'annotation, mais pas de Gson. C'est stupide facile à utiliser.

```
public class GsonMessageBodyProvider<T> implements MessageBodyReader<T>, MessageBodyWriter<T>
{

    private Gson gson = new GsonBuilder().create();

    @Override
    public boolean isReadable(Class<?> type, Type genericType, Annotation[] annotations,
MediaType mediaType) {
        return StringUtils.endsWithIgnoreCase(mediaType.getSubtype(), "json");
    }

    @Override
    public T readFrom(Class<T> type, Type genericType, Annotation[] annotations, MediaType
mediaType, MultivaluedMap<String, String> httpHeaders, InputStream entityStream) throws
IOException, WebApplicationException {
        try {
            return gson.fromJson(new BufferedReader(new InputStreamReader(entityStream, "UTF-
8")), type);
        }
        catch (Exception e) {
            throw new IOException("Trouble reading into:" + type.getName(), e);
        }
    }

    @Override
    public boolean isWriteable(Class<?> type, Type genericType, Annotation[] annotations,
MediaType mediaType) {
        return StringUtils.containsIgnoreCase(mediaType.getSubtype(), "json");
    }

    @Override
    public long getSize(T t, Class<?> type, Type genericType, Annotation[] annotations,
MediaType mediaType) {
        return 0;
    }

    @Override
    public void writeTo(T t, Class<?> type, Type genericType, Annotation[] annotations,
MediaType mediaType, MultivaluedMap<String, Object> httpHeaders, OutputStream entityStream)
throws IOException, WebApplicationException {
        try {
            JsonWriter writer = new JsonWriter(new OutputStreamWriter(entityStream, "UTF-8"));
            writer.setIndent("  ");
            gson.toJson(t, type, writer);
```

```
            writer.close();
        }
        catch (Exception e) {
            throw new IOException("Trouble marshalling:" + type.getName(), e);
        }
    }
}
```

Le code est simple. Vous implémentez deux interfaces: MessageBodyReader et MessageBodyWriter (ou une seule), puis vous l'ajoutez aux "fournisseurs" lors de la création du WebClient. CXF en est conscient. Une option consiste à renvoyer "true" dans les méthodes "isReadable ()" "isWriteable ()". Cela assurera que CXF utilise cette classe à la place de toutes celles intégrées. Les fournisseurs ajoutés seront vérifiés en premier.

Configuration de CXF pour JAX-RS

Les pots pour CXF JAX-RS se trouvent dans Maven:

```
<!-- https://mvnrepository.com/artifact/org.apache.cxf/cxf-rt-rs-client -->
<dependency>
    <groupId>org.apache.cxf</groupId>
    <artifactId>cxf-rt-rs-client</artifactId>
    <version>3.1.10</version>
</dependency>
```

Ces pots sont tout ce dont vous avez besoin pour le faire fonctionner:

```
cxf-rt-rs-client-3.1.10.jar
cxf-rt-transports-http-3.1.10.jar
cxf-core-3.1.10.jar
woodstox-core-asl-4.4.1.jar
stax2-api-3.1.4.jar
xmlschema-core-2.2.1.jar
cxf-rt-frontend-jaxrs-3.1.10.jar
javax.ws.rs-api-2.0.1.jar
javax.annotation-api-1.2.jar
```

Filtres Client

Une bonne raison d'utiliser les filtres est la journalisation. En utilisant cette technique, un appel REST peut être enregistré et programmé facilement.

```
public class RestLogger implements ClientRequestFilter, ClientResponseFilter {
    private static final Logger log = LoggerFactory.getLogger(RestLogger.class);

    // Used for timing this call.
    private static final ThreadLocal<Long> startTime = new ThreadLocal<Long>();
    private boolean logRequestEntity;
    private boolean logResponseEntity;

    private static Gson GSON = new GsonBuilder().create();

    public RestLogger(boolean logRequestEntity, boolean logResponseEntity) {
```

```
        this.logRequestEntity = logRequestEntity;
        this.logResponseEntity = logResponseEntity;
    }


    @Override
    public void filter(ClientRequestContext requestContext) throws IOException {
        startTime.set(System.currentTimeMillis());
    }

    @Override
    public void filter(ClientRequestContext requestContext, ClientResponseContext
responseContext) throws IOException {
        StringBuilder sb = new StringBuilder();
        sb.append("HTTP:").append(responseContext.getStatus());
        sb.append(" - Time:[").append(System.currentTimeMillis() -
startTime.get().longValue()).append("ms]");
        sb.append(" - Path:").append(requestContext.getUri());
        sb.append(" - Content-
type:").append(requestContext.getStringHeaders().getFirst(HttpHeaders.CONTENT_TYPE.toString()));

        sb.append(" -
Accept:").append(requestContext.getStringHeaders().getFirst(HttpHeaders.ACCEPT.toString()));
        if (logRequestEntity) {
            sb.append(" - RequestBody:").append(requestContext.getEntity() != null ?
GSON.toJson(requestContext.getEntity()) : "none");
        }
        if (logResponseEntity) {
            sb.append(" - ResponseBody:").append(this.logResponse(responseContext));
        }
        log.info(sb.toString());
    }

    private String logResponse(ClientResponseContext response) {
        StringBuilder b = new StringBuilder();
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        InputStream in = response.getEntityStream();
        try {
            ReaderWriter.writeTo(in, out);
            byte[] requestEntity = out.toByteArray();
            b.append(new String(requestEntity));
            response.setEntityStream(new ByteArrayInputStream(requestEntity));
        }
        catch (IOException ex) {
            throw new ClientHandlerException(ex);
        }
        return b.toString();
    }
}
```

Vous pouvez voir ci-dessus que la requête est interceptée avant l'envoi de la réponse et qu'un ThreadLocal Long est défini. Lorsque la réponse est renvoyée, nous pouvons enregistrer la demande et la réponse et toutes sortes de données pertinentes. Bien sûr, cela ne fonctionne que pour les réponses Gson et autres, mais peut être modifié facilement. Ceci est mis en place de cette façon:

```
List<Object> providers = new ArrayList<Object>();
providers.add(new GsonMessageBodyProvider());
providers.add(new RestLogger(true, true)); <------right here!
```

```
WebClient webClient = WebClient.create(PORTAL_URL, providers);
```

Le journal fourni devrait ressembler à ceci:

```
7278 [main] INFO  blah.RestLogger - HTTP:200 - Time:[1391ms] - User:unknown -
Path:https://blah.com/tmet/moduleDescriptions/desc?languageCode=en&moduleId=142 - Content-
type:null - Accept:application/json - RequestBody:none -
ResponseBody:{"languageCode":"EN","moduleId":142,"moduleDescription":"ECAP"}
```

Lire Démarrer avec cxf en ligne: https://riptutorial.com/fr/cxf/topic/7387/demarrer-avec-cxf

Crédits

S. No	Chapitres	Contributeurs
1	Démarrer avec cxf	Community, markthegrea