EBook Gratis

APRENDIZAJE cxf

Free unaffiliated eBook created from
Stack Overflow contributors.

#cxf

Tabla de contenido

Acerca de

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

Capítulo 1: Empezando con cxf

Observaciones

Esta sección proporciona una descripción general de qué es cxf y por qué un desarrollador puede querer usarlo.

También debe mencionar cualquier tema grande dentro de cxf, y vincular a los temas relacionados. Dado que la Documentación para cxf es nueva, es posible que deba crear versiones iniciales de los temas relacionados.

Examples

Cliente web básico con proveedor

Para empezar necesitamos una fábrica que produzca clientes web.

```
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
        }
    }
}
```

- Primero creamos una lista de proveedores (llegaremos a aquellos más adelante)
- A continuación, creamos un nuevo cliente web utilizando la fábrica estática "create ()". Aquí podemos agregar algunas otras cosas como credenciales de autenticación básica y seguridad de subprocesos. Por ahora solo usa este.

- A continuación sacamos el HTTPConduit y establecemos los tiempos de espera. CXF viene con clientes de clase base Java pero puede usar Glassfish u otros.
- Finalmente cacheamos el WebClient. Esto es importante ya que el código hasta ahora es caro de crear. La siguiente línea muestra cómo hacerlo seguro para hilos. Tenga en cuenta que esta es también la forma en que extraemos el código de la memoria caché. Esencialmente, estamos haciendo un modelo de la llamada REST y luego la clonamos cada vez que la necesitamos.
- Observe lo que NO está aquí: No hemos agregado ningún parámetro o URLS. Se pueden agregar aquí, pero eso sería un punto final específico y queremos uno genérico. También no hay encabezados añadidos a la solicitud. Estos no lo hacen más allá del "clon", por lo que deben agregarse más tarde.

Ahora tenemos un WebClient que está listo para funcionar. Permite configurar el resto de la llamada.

```
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;
}
```

- Dentro del "intento" tomamos un WebClient de la fábrica. Este es un nuevo "helado".
- A continuación ponemos unos encabezados. Aquí agregamos algún tipo de encabezado Auth y luego un encabezado de aceptación. Observe que tenemos un encabezado de aceptación personalizado.
- Añadir la ruta y las cadenas de consulta viene a continuación. Tenga en cuenta que no hay ningún orden en estos pasos.
- Finalmente hacemos el "get". Hay varias maneras de hacer esto, por supuesto. Aquí tenemos a CXF haciendo el mapeo JSON por nosotros. Cuando se hace de esta manera, tenemos que lidiar con las WebApplicationExceptions. Así que si obtenemos un 404, CXF lanzará una excepción. Tenga en cuenta que aquí me comen esas excepciones porque

simplemente registro la respuesta en el final. Sin embargo, quiero obtener OTRA excepción, ya que pueden ser importantes.

- Si no le gusta este cambio de manejo de Excepciones, puede recuperar el objeto Respuesta de la "obtención". Este objeto contiene la entidad y el código de estado HTTP. NUNCA lanzará una WebApplicationException. El único inconveniente es que no realiza la asignación JSON por usted.
- Finalmente, en la cláusula "finalmente" tenemos un "wc.close ()". Si obtiene el objeto de la cláusula get, realmente no tiene que hacer esto. Algo podría salir mal, por lo que es una buena prueba de fallos.

Entonces, ¿qué pasa con ese encabezado de "aceptar": application / person-v1 + json ¿Cómo sabrá CXF cómo analizarlo? CXF viene con algunos analizadores JSON integrados pero quería usar Gson. Muchas de las otras implementaciones de analizadores Json necesitan algún tipo de anotación pero no Gson. Es estúpido fácil de usar.

```
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);
        }
    }
}
```

El código es sencillo. Implementas dos interfaces: MessageBodyReader y MessageBodyWriter (o solo una) y luego lo agregas a los "proveedores" al crear el WebClient. CXF lo descubre desde allí. Una opción es devolver "verdadero" en los métodos "isReadable ()" "isWriteable ()". Esto asegurará que CXF use esta clase en lugar de todas las integradas. Los proveedores agregados serán revisados primero.

Configurando CXF para JAX-RS

Los frascos para CXF JAX-RS se encuentran en 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>
```

Estos tarros son todo lo que necesitas para ponerlo en funcionamiento:

```
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
```

Filtros de clientes

Una buena razón para usar filtros es para el registro. Usando esta técnica, una llamada REST se puede registrar y cronometrar fácilmente.

```
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();
    }
}
```

Arriba puede ver que la solicitud se intercepta antes de que se envíe la respuesta y se establezca un ThreadLocal Long. Cuando se devuelve la respuesta, podemos registrar la solicitud y la respuesta y todo tipo de datos pertinentes. Por supuesto, esto solo funciona para las respuestas de Gson y similares, pero se puede modificar fácilmente. Esto se configura de esta manera:

```
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);
```

El registro provisto debe verse algo como esto:

```
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"}
```

Lea Empezando con cxf en línea: https://riptutorial.com/es/cxf/topic/7387/empezando-con-cxf

Creditos

S. No	Capítulos	Contributors
1	Empezando con cxf	Community, markthegrea