XORCERY USA JETTY & JERSEY E IMPLEMENTACIÓN DE MTLS Y SOPORTE A JWT

on **octubre 31, 2023** in **xorcery** with **No hay comentarios.**

Para poder entender mejor a **Xorcery** vamos a explicar los bloques de construcción sobre los cuales reside el framework. En Xorcery hacemos uso de Jetty y Jersey.

JETTY

Jetty, es un servidor web y contenedor de servlets ligero preparado para ser embebido en nuestras aplicaciones. La distribución que usamos es la de eclipse y posee las siguientes características:

- Es un servidor HTTP asíncrono
- Es un Contenedor de Servlet estándar
- Es un servidor de Web Sockets
- Posee un cliente HTTP asíncrono
- Soporta OSGI, JNDI, JMX, JASPI, AJP.

Jersey

Desarrollar servicios web RESTful que admitan sin problemas la exposición de sus datos en una variedad de tipos de medios de representación y abstraigan los detalles de bajo nivel de la comunicación cliente-servidor no es una tarea fácil sin un buen conjunto de herramientas. Para simplificar el desarrollo de servicios web RESTful y sus clientes en Java, se ha diseñado una API JAX-RS estándar y portátil.

El framework Jersey RESTful Web Services 2.x es un marco de código abierto y de calidad de producción para desarrollar servicios web RESTful en Java que brinda soporte para las API JAX-RS y sirve como implementación de referencia JAX-RS (JSR 311, JSR 339 y JSR 370).

El framework Jersey RESTful Web Services 3.x brinda soporte para Jakarta RESTful Web Services 3.0.

El framework Jersey es más que la implementación de referencia JAX-RS. Jersey proporciona su propia API que amplía el kit de herramientas JAX-RS con funciones y utilidades adicionales para simplificar aún más el servicio RESTful y el desarrollo de clientes. Jersey también expone numerosos SPI de extensión para que los desarrolladores puedan ampliar Jersey para satisfacer mejor sus necesidades.

Módulos en Java

Todo el código en Xorcery está desarrollado usando Java Modules. Si tienes problemas para entender esta organización disponible desde Java 9, te recomiendo ver este link (puedes usar los subtítulos en español o inglés).

En español encontré esta buena charla para que puedan entender el trabajo de un proyecto en módulos: Modularidad en Java, creación y migración de aplicaciones.

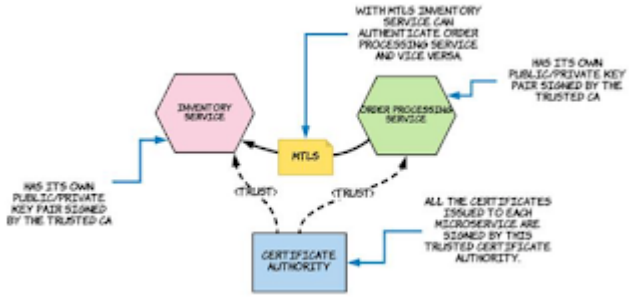
GlassFish HK2 - Inyección de Dependencias

GlassFish HK2 3.0 API, es un framework declarativo para servicios usando anotaciones como Contract y Service. Está basado en Jakarta Dependency Injection (DI) standard annotation. Lo estamos usando como framework de inyección de dependencias en Jersey y habilitando el auto-scanning para auto-descubrir los componentes marcados como @Contract y @Service.

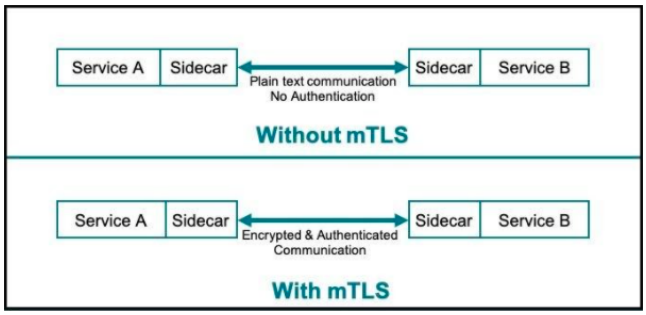
Puede encontrar un ejemplo al respecto en este link: https://mkyong.com/webservices/jax-rs/jersey-and-hk2-dependency-injection-auto-scanning/

Uso en Xorcery para tener mTLS

En la búsqueda de conseguir comunicación encriptada mTLS de servidor a servidor, se ha activado el TLS para HTTP 1.1 y HTTP/2 y el soporte para certificados cliente y así obtener soporte mTLS.



Este tipo de comunicación lo debe ud. Haber leído o conocido al usar un Service Mesh. Pero Xorcery, lo que desea es evitar tal complejidad y ofrecerle el mismo tipo de comunicación segura entre servicios.



Por tal motivo, en el código fuente encontrará:

1. **xorcery-certificates**: el código que inicia el aprovisionamiento, configura el programador de renovación de certificados y almacena el certificado en el KeyStore, etc. Esto puede

ejecutarse tanto en el cliente como en el servidor, y define un SPI

2. **xorcery-certificates-client**: implementa SPI y delega al servidor remoto a través de HTTP. Opcionalmente, puede admitir el reto HTTP01 ACME
3. **xorcery-certificates-server**: proporciona API JAXRS que los clientes (2.) pueden llamar y luego delega en implementaciones SPI
4. **xorcery-certificates-letsencrypt**: Implementación Let's Encrypt de SPI, se puede ejecutar en "cliente" o "servidor", ambos funcionarían.
5. **xorcery-certificates-ca**: nuestra propia implementación CA del SPI

Se ha buscado diseñar el SPI cuidadosamente, para tener una configuración más limpia y que haga posible el uso de Let's Encrypt y nuestro propio CA al mismo tiempo (hacen cosas diferentes y tienen ventajas y desventajas cada uno).
Se pueden ejecutar en modo independiente (donde cada servidor realiza el aprovisionamiento por sí mismo) o en modo cliente-servidor donde el aprovisionamiento/renovación está centralizado.

Los servidores ahora pueden recibir certificados de nuestra propia CA o Let's Encrypt, o ambos (lo que hace posible el SNI).

La validación de la Solicitud de firma de certificado ahora admite opcionalmente la autofirma, por lo que no es necesario distribuir la clave de aprovisionamiento si sabe que todas las CSR provienen de su propia red.

JWT Autenticación y Autorización

También tenemos un módulo **xorcery-jwt** que:

1. Expone un login endpoint (username/password). De esta manera, los clientes podrán realizar la autenticación.
2. Tiene un SPI para permitir que el plugin provea JWT Claims.

En **xorcerty-jetty-server** también se ha agregado soporte para Jetty contraints mappings de manera que se pueda agregar requerimientos de autorización. Básicamente, se verificará los "roles" de los claims JWT (como una serie de nombres de roles) con los requisitos de roles de una ruta mapeada. Con esto último ya tenemos un nivel de autorización.

Uso en clientes

Con todo lo anteriormente explicado, ahora podrá configurar en sus clientes la información de certificados, recursos REST API, información del jetty server, JWT issuer, etc.

La siguiente configuración la tomo de xorcery-examples:

```
application.name: "forum"
instance.home: "{{ SYSTEM.jpackage_app-path ? jpackage.app | SYSTEM.user_dir}}"
jpackage.app: "{{ SYSTEM.jpackage_app-path }}/../../lib/app"

# So that we can generate a SSL certificate for the local hostname. Replace with whatever domain name you actually use
instance.domain: local

# Add local convenience names for your own computer into the SSL cert
certificates:
  dnsNames:
    - localhost
    - "{{ instance.host }}"
  ipAddresses:
    - 127.0.0.1
    - "{{ instance.ip }}"

# REST API resources
jersey.server.register:
  - com.exoreaction.xorcery.examples.forum.resources.api.CommentResource
  - com.exoreaction.xorcery.examples.forum.resources.api.ForumResource
  - com.exoreaction.xorcery.examples.forum.resources.api.PostCommentsResource
  - com.exoreaction.xorcery.examples.forum.resources.api.PostResource
```

```
  -
com.exoreaction.xorcery.examples.forum.resources.api.PostsReso
urce


dns.client.search:
  - xorcery.test
dns.client.hosts:
    _certificates._sub._https._tcp : "https://127.0.0.1"
dns.client.nameServers:
  - 127.0.0.1:8853

jetty:
  server:
    http:
      port: 8080
    ssl:
      port: 8443
    security:
      jwt:
        issuers:
          server.xorcery.test: "MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ....."

# These features can be extracted into separate services
jwt.server.keys:
  - keyId: "2d3f1d1f-4038-4c01-beb7-97b260462ada"
    alg: "ES256"
    publicKey: "secret:MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ...."
    privateKey:
"secret:MEECAQAwEwYHKoZIzj0CAQYIKoZIzj0DAQ...."

dns.server.port: 8853

# Log configuration
log4j2.Configuration:
  name: Xorcery Example Forum
  status: warn
  thresholdFilter:
    level: trace
  appenders:
    Console:
      name: STDOUT
      target: SYSTEM_OUT
      PatternLayout:
        Pattern: "%d [%t] %-5level %marker %c{1.}:
%msg%n%throwable"
```

```
#   Log4jPublisher:
#     name: Log4jPublisher
#     PatternLayout:
#       Pattern: "%d [%t] %-5level %marker %c{1.}:
%msg%n%throwable"

  Loggers:
    logger:
      - name: org.apache.logging.log4j
        level: debug
        additivity: false
        AppenderRef:
          ref: STDOUT

      - name: com.exoreaction.xorcery.core.Xorcery
        level: debug

      - name: com.exoreaction.xorcery.service
        level: debug

      - name: com.exoreaction.xorcery.dns
        level: trace

    Root:
      level: info
      AppenderRef:
        - ref: STDOUT
#       - ref: Log4jPublisher
```

En los siguientes artículos vamos a explicar que otros bloques de construcción posee Xorcery y el porqué de su uso para finalmente terminar en las demos de micro servicios.

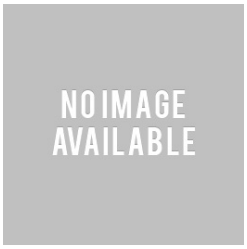
Sean bienvenidos de antemano sus Pull Requests a este framework.

Enjoy!

Jose

Share:

Related Posts:



Xorcery is born - Build high-performance microservices

Xorcery uses Jetty & Jersey and mTLS implementation and JWT support

Xorcery implements Reactive Streams - Parte 1

Xorcery usa Jetty & Jersey e implementación de mTLS y soporte a JWT

Xorcery implementa Reactive Streams - Parte 1

Entrada más reciente Página Principal Entrada antigua

0 comentarios:

Publicar un comentario

Para dejar un comentario, haz clic en el botón de abajo para acceder con Google.

ACCEDER CON GOOGLE

JoeDayz.pe

Cursos

Popular Posts

Xorcery implementa Reactive Streams - Parte 1

¿Qué es Reactive Streams? Para poder entender como nos permite Xorcery trabajar con reactive streams , tenemos que saber que es Reactive ...

Paso de la Oración

Este 10, 11 y 12 de Septiembre ha sido un fin de semana enriquecedor para Miryan y

About

Categories

#github #java (1)

#guatejug (1)

#historiasdeprogramador (1)

aaii (1)

academia web (2)

acr (1)

para mí. Despues de varios años fuimos a la convivencia...

Blog Archive

grails (5)

groovy (3)

hangouts (1)

highchart-export-server (2)

huacho (1)

hudson (1)

hyperledger-composer (1)

hyperledger-fabric (1)

i-educa (1)

iBATIS (2)

icescrum (1)

informatica (1)

Intigas (1)

ITP_JAVA (1)

jakartaee (4)

jakartaee10 (1)

JasperReports (1)

java (3)

JavaCard (1)

JavaDayUNI (1)

JavaOne (1)

jhipster (2)

jmeter (1)

joedayz (46)

JOERP (4)

jpa (1)

jquery (1)

kafka (3)

kotlin (2)

Kubernetes (3)

lombok (1)

m2eclipse (1)

mac (2)

Matt Raible (1)

Maven (3)

microprofile (6)

microprofile-jwt jakartaee (2)

microprofile-jwt jdbc-realm jakartaee (1)

microprofile-jwt jdbc-realm payara (1)

microservicios (1)

Ministerio del Interior (1)

MJN (5)

móvil (1)

mysql (1)

namespaces (1)

navidad (1)

NET (4)

Nextel (1)

Novell (1)

ocjp (1)

Opentaps (2)

Oracle (1)

oraclecloud (1)

oraclefunctions (1)

oracleopenworld (1)

OSUM (1)

OSX (1)

p6spy (1)

Payara (5)

personal (1)

perujug jconfperu joedayz (2)

php (1)

play (1)

PMP (1)

podcasts (1)

PostgreSQL (8)

programacion (1)

pubsub (1)

PUCP (4)

quadim (2)

quarkus (1)

rackspace (1)

rails (2)

redis (1)

refactoring (2)

Reniec (1)

renovatebot (2)

Rider (1)

ruby (4)

rust (1)

scala (1)

SCD2010 (1)

SCJP (1)

Scrum (3)

Scrum evaluacion (1)

seminarios (1)

Setup (1)

SourceRepo (1)

spring (13)

spring 3.1 (1)

spring android (1)

spring mobile (1)

spring social (1)

spring-boot (9)

spring-boot-admin (2)

spring-cloud (1)

spring-cloud-config (2)

SpringCommunityDay (1)

SpringRoo (2)

springsource (1)

sqlserver (2)

start-up (1)

STS (1)

Subclipse (1)

Subversion (1)

SUN (1)

SUNAT (2)

synergyj (2)

Syscom (1)

Talleres (21)

Telefonica (1)

thedevconf (1)

thymeleaf (1)

Trac (1)

try-with-resources (1)

twitter (1)

Tye (1)

ubuntu (3)

UNI (3)

UNMSM (1)

UPC (1)

videos (1)

vimeo (1)

weblogic (2)

Workspace (1)

WPF (1)

xorcery (6)

xsd (1)

YaRetail (1)

YoMeQuedoEnCasa (1)

zuul (2)