Sistemas distribuídos 2015-2016

A32-https://github.com/tecnico-distsys/A_32-project

Guilherme Pinho

Nº78819

André Vieira

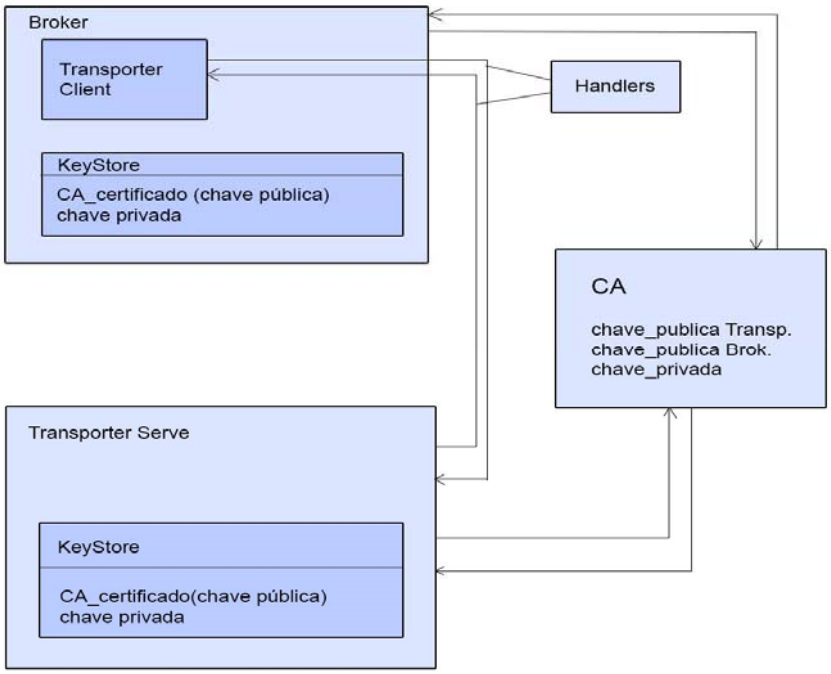
Nº79591

Miguel Pinto

Nº79060

1 SEGURANÇA

1.1 IMAGEM



1.2 DESCRIÇÃO DA IMAGEM

Nesta imagem temos 4 entidades:

→**Handlers**: biblioteca que intercepta as mensagens e adiciona conteúdo.

→**CA**: Distribuição de certificados que contêm as chaves das outras entidades; composto por um conjunto de chaves: chave pública da entidade transporter (transporter 1 e transporter 2), chave pública da entidade Broker e a sua chave privada.

→**Broker:** Uma das entidades principais do projeto. Broker é composto por uma instância de Transporter-client e por uma keyStore; Esta KeyStore guarda no seu interior a chave privada do broker e um CA_certificado (certificado público da CA).

→ **Transporter-server**: A outra entidade principal do projeto. Esta entidade tem uma keyStore, sendo esta composta pela chave privada do transporter server e um CA_certificado (certificado público da CA).

Esta imagem serve para descrever a seguinte sequência de acontecimentos:

O Broker comunica com o transporter (através da instância de transporter client), esta mensagem é interceptada pelos handlers que adicionam conteúdo (os headers), este conteúdo é cifrado com a chave privada do broker. Na recepção da mensagem o transporter pede à CA o certificado com a chave pública do broker. O CA envia o certificado e o transporter decifra a mensagem após a extração da chave pública do certificado enviado pela CA.

1.3 RACIONAL

Os handlers são uma biblioteca que permitem adicionar conteúdo às mensagens SOAP após estas terem passado pelas STUB. No caso concreto do projeto, são necessários para criar os headers que levarão uma assinatura digital, assim como um Nonce e um identificador de mensagem. Estes handlers irão estar entre o Transporter Server e o Broker Server (efetivamente no Transporter Client, visto que só é possível utilizá-los entre Client-Servidor).

A assinatura digital permite garantir a autenticidade das mensagens, o Nonce garante a frescura, e o identificador serve para não existirem *reply attacks.*

A nossa solução para o problema é a seguinte:

Primeiro temos que adicionar headers às mensagens SOAP, para isso usamos os handlers entre o transporter-server e transporter-cliente.

Estes handlers permitem adicionar à mensagem um header, isto é importante porque com este header há a possibilidade de autenticar, e garantir a integridade das mensagens, assim como todos os outros requisitos não funcionais pedidos. Para autenticar as mensagens é necessário ter a chave pública de quem enviou. Para isso pede-se ao CA a chave pública do emissor. A chave vem num certificado no formato X509 devidamente assinado pela CA, para garantir mais uma vez autenticidade. Após obter o certificado extrai-se a chave pública e decifra-se a mensagem que vinha no Header da mensagem SOAP. Se corresponder à cifra gerada no momento para garantir a igualdade, pode-se dizer que a mensagem foi enviada pelo emissor esperado.

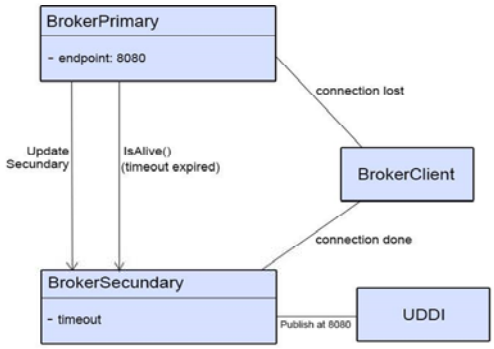
Então sempre que há uma mensagem, por exemplo no sentido broker→transporter o body é cifrado (com a chave privada do broker) e colocado no header da mensagem pelo broker. Chegando esta mensagem ao transporter, este vai ao CA pedir a chave pública correspondente à entidade que enviou a mensagem.

Então sempre que há uma mensagem, por exemplo no sentido broker→transporter o body é cifrado (com a chave privada do broker) e colocado no header da mensagem pelo broker. Chegando esta mensagem ao transporter, este vai ao CA pedir a chave pública correspondente à entidade que enviou a mensagem.

Com este sistema garante-se a autenticidade das mensagens e o não repúdio das mesmas.

2 REPLICAÇÃO

2.1 IMAGEM



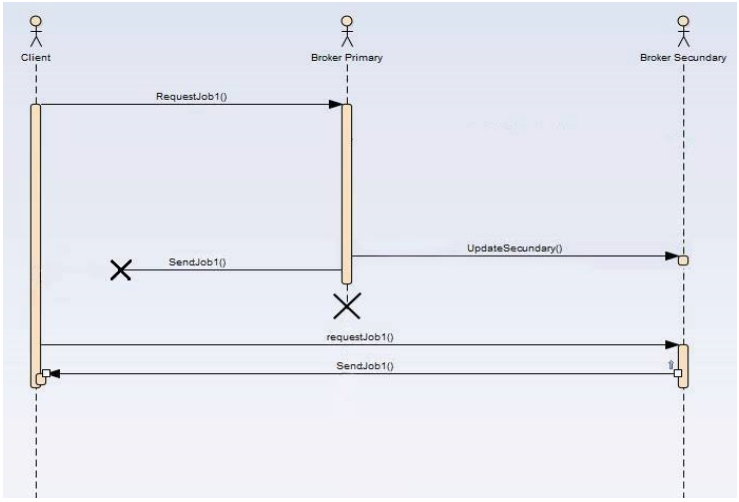


Fig.1 Interação em caso de Crash

Fig.2 Sequênica de ações em caso de falha no envioo de resposta

2.2 DESCRIÇÃO DA IMAGEM

Fig.1 – Nesta imagem representa-se o cenário seguinte:

1→A interação entre o broker primário e o secundário através dos métodos updateSecundary e IsALive explicados na secção seguinte.

2→O broker primário perde ligação com o broker client devido a um crash.

3→Com isto o timeout expira-se.

4→O broker secundário percebe que o broker primário entrou em falha e regista-se no Uddi com o mesmo endpoint do broker primário (8080).

5→A ligação com o cliente é gerada automaticamente e tudo decorre normalmente a partir deste momento.

Fig.2 – Nesta imagem quer-se representar a utilização da semântica no máximo-uma-vez:

1→O cliente faz um pedido ao broker primário.

2→O broker primário executa o pedido e envia o update para o broker secundário.

3→O broker primário tem um crash antes de conseguir enviar a resposta ao cliente.

4→O cliente reenvia o pedido após o seu timeout ter-se esgotado (envia ao broker secundário).

5→O broker secundário vê que o pedido é repetido e simplesmente envia a resposta ao pedido, não o executa de novo.

2.3 RACIONAL

Para este tipo de problemas em tolerância a faltas temos de ter em conta a seguinte fórmula:

Total = (f + 1) → sendo f o número de faltas e o total o número de servidores necessários parar tolerar essas faltas.

No nosso caso em concreto simplesmente consideramos f=1; E, por isso, só necessitamos de dois servidores: broker primário e broker secundário.

A nossa solução para este problema é:

Existe um broker primário e um broker secundário. O broker primário regista o seu Serviço como UpaBroker, num determinado IP no UDDI. Enquanto que o broker secundário espera a sua vez de se registar.

Entre os servidores (primário e secundário) existem dois métodos de comunicação:

1-IsAlive()

2-UpdateSecundary()

O objetivo do método 1 é de tempos a tempos verificar se o broker primário ainda está em cima e funcional, é como um ping para verificar a ligação. Este método é o que indica ao broker secundário quando tem de se tornar o servidor principal, isto é, se o método IsAlive não for ativado pelo Broker primário no instante em que é esperado (timeout=10 segundos), então o broker secundário assume que o broker primário teve um problema e vai ao UDDI registar-se com o endpoint anteriormente usado pelo broker primário (no nosso caso o endpoint é no porto 8080) e com o nome UpaBroker (nome anteriormente usado pelo primário). Com esta nova publicação o cliente continua a sua execução normalmente ligado ao servidor sem notar a alteração do mesmo. O Broker secundário passa a primário.

Este sistema de inserção de um timeout no servidor secundário só funciona por estarmos dentro da rede do técnico. Em caso de o projeto ser corrido na web teríamos que arranjar outra solução visto que em rede as mensagens poderiam estar a saltar de router em router (ou ficar perdidas em algum e depois voltar ao seu caminho) e não haver um tempo estimado da recepção da mensagem, sendo impossível assim usar um timeout adequado.

Para isto funcionar o Broker secundário tem de estar sempre atualizado, para isso temos o método 2. Este método é acionado pelo Broker primário sempre que há uma mudança de estado nos trabalhos, enviando para o broker secundário a informação a ser guardada (o último trabalho pedido). Com isto, o broker secundário está sempre pronto para assumir o papel de servidor primário.

Como exemplificado na figura 2, temos de ter em conta a semântica das mensagens. Temos de aplicar uma semântica no-máximo-uma-vez de modo a possibilitar que o cliente repita pedidos perdidos em caso de crash do broker primário, (pedido é efetuado no broker primário e enviado o resultado para o secundário, mas não é transmitida a resposta ao cliente), sem que o broker secundário execute mais do que uma vez esse pedido. A imagem é ilustrativa da situação.

Para isto adicionamos identificadores a cada mensagem, sempre que uma mensagem é enviada é guardado numa estrutura o identificador desta mensagem. Possibilitando assim que as mensagens repetidas possam ser ignoradas.