Stomp for InPlace

Joop Ringelberg 03-11-20 Version: 1

Introduction

This text can be read as a companion to *A Transport Layer for the PDR*. In it we describe in some detail the technical design decisions underlying the implementation of the message layer.

Technology chosen

The implementation is built on RabbitMQ (https://www.rabbitmq.com/documentation.html) and Stompjs (http://jmesnil.net/stomp-websocket/doc/). The latter library caters for Stomp version 1.0 and 1.1, not 1.2. The Stomp web plugin for RabbitMQ handles all versions.

Exchange type

While peers may use the Perspective User Identity (PUI) to send Transactions to, only the intended receiver must be able to subscribe to the relevant queue on the AMQP server. To achieve this end, we use a Topic Exchange where

- the routing keys are PUIs;
- the binding keys are PUIs, but only the PDR for a particular PUI knows the identification of the queue that binds that PUI.

Note that not even the server administrator has to know the queue that uses a particular PUI as binding key. The subscribing PDR can keep it to itself.

A Topic Exchange matches routing keys to binding keys, where the latter may include wildcards. We have the simplest possible situation, where both keys are equal. It is the binding rule that connects the key to a particular queue, that protects the receiver from others marauding his post box!

Creating topic queues

The web client using Stomp can create a queue with a particular binding key in a Topic Exchange; we don't need the RabbitMQ administrator for that.

It turns out that when a new vhost is created using the management console of RabbitMQ, all types of Exchanges are created for it. Stomp sends a frame with a destination string that starts with "/topic" automatically to the amq.topic Exchange of that vhost.

A queue with a particular binding key and queue identification can be created from the client as follows:

```
const {id, unsubscribe} = client.subscribe(
  "/topic/" + topic,
  function(message){…},   // handle the message
  { durable: true
  , "auto-delete": false
  , id: "secred-id"       // the secret queue identification.
  });
```

Notice the fields in the object that is provided as last argument to `subscribe`. They specify that, apart from the queue identification, the queue is not to be deleted when no one subscribes to it and will be available after the server restarts, too.

This behaviour is governed partially by the semantics attributed to the *destination string* that, by default, Stomp assigns neither structure nor semantics to. For RabbitMQ this is described in https://www.rabbitmq.com/stomp.html.

Acknowledgements

We don't want Transactions to get lost. To prevent the RabbitMQ server from deleting a Transaction before it has been handled by the receiver, we make the receiver send explicit acknowledgements.

By default the server removes a message after it has been delivered. To change that behaviour we give the object supplied as third argument to `subscribe` with another key:

```
ack: "client"
```

Now the subscribing client has to acknowledge the message, using the function that is the value of the field `ack` on the message that is received.

Heartbeat

By default, the Stomp server sets up a heartbeat (RabbitMQ by default sends a beat every 10.000 milliseconds). However, as we have the client send explicit acknowledgement, it seems not necessary to have a heartbeat on the socket level. This is how to disable it:

```
const client = Stomp.client(url);
client.heartbeat = {incoming: 0, outgoing: 0};
```

User accounts

The RabbitMQ manager must create user accounts; there is no self-registration.