tclrmq

Garrett McGrath

Tcl Conference 2017

Where Is It?

- Full source code under a BSD-style license:
  - https://github.com/flightaware/tclrmq
- Contains full documentation, RabbitMQ tutorials, additional examples
- Welcome all contributions and feature suggestions

```
package require rmq
```

What Is It?

- Pure Tcl Library for RabbitMQ
  - Requires Tcl 8.6 (uses TclOO and, if TLS is needed, TclTLS)
  - No external bindings, no compilation
- Fully asynchronous
  - No blocking
  - Callback based
- Supports AMQP 0-9-1
  - Most widely supported version of the protocol
  - Primary RabbitMQ use case

AMQP?

- Advanced Message Queueing Protocol
- Programmable protocol for working with distributed queues
- Open standard developed as a cooperative effort
- Some of the earliest organizations with technical contributors
  - Red Hat
  - Cisco
  - JPMorgan Chase
- Binary, application layer protocol
  - Semantics defined in OO fashion
  - Provides several classes and methods that servers and clients must implement
  - Offers a message broker

RabbitMQ

- Particular implementation of AMQP
- Open source
- Written in Erlang
- Actively developed and maintained
- Well documented
- Supports distributed operation at client and server level
- Adds a number of protocol extensions
- Management tools and other plugins

Task 0: Channeling Connections

```
package require rmq

# Need some credentials
set login [::rmq::Login new -user tcl -pass
secret]

# Create a Connection object
set conn [::rmq::Connection new -login $login]

# Set a callback for when it connects
$conn onConnected rmq_connected

# Make the connection
$conn connect

# Enter the event loop
vwait ::die
```

```
proc rmq_connected {conn} {
    # Open a channel and do some work
    set rChan [::rmq::Channel new $conn]

}
```

More Than A FIFO

- Same idea as the queue ADT
  - Altered interface
  - AMQP server adds a new level of indirection
- Cannot put data directly on a queue
  - All messages sent to an exchange
- Exchange decides which queue to put the message
  - Uses client-supplied bindings to route messages
  - Where much of the power and programmability resides
- Several types of exchanges
  - direct
  - fanout (1-to-all) (publish / subscribe)
  - topic (filtered publish / subscribe)
  - header (programmable semantics: priority queues, consistent hashing)

Declarations: Exchanges

```
proc rmq_connected {conn} {
    set rChan [::rmq::Channel new $conn]
    $rChan onOpen declare_exchanges
}
```

```
proc declare_exchanges {rChan} {
    set eTypes [list direct topic fanout header]
    set eFlags [list $::rmq::EXCHANGE_DURABLE]

    $rChan on exchangeDeclareOk exchange_declared
    $rChan onError channel_error

    foreach eType $eTypes {
        $rChan exchangeDeclare "xname_$eType" $eType $eFlags
        vwait ::declared
    }

    declare_queues $rChan
}
```

Declarations: Queues

```
proc declare_queues {rChan} {
    # create a queue that persists after restarts and do
    # not expect any response from the server
    set qFlags [list $::rmq::QUEUE_DURABLE $::rmq::QUEUE_DECLARE_NO_WAIT]
    $rChan queueDeclare "tcl_queue" $qFlags

    # create a queue that only is accessed by the current connection
    # let the server give us a name for it
    $rChan on queueDeclareOk save_queue_name
    set qFlags [list $::rmq::QUEUE_EXCLUSIVE]
    $rChan queueDeclare "" $qFlags
}

proc save_queue_name {rChan qName msgCount consumerCount} {
    # do something useful with the queue name
    # save the exclusive queue's name, or bind it to an exchange
}
```

Bindings: Connecting Exchanges and Queues

```
proc queue_bind_after_declare {rChan qName msgCount consumerCount} {
   # binding is simple: give a queue name and an exchange name
   # provide a routing key
   $rChan queueBind $qName "xname_topic" "tcl.conference.2017"

   $rChan on queueBindOk queue_bound
}

proc queue_bound {rChan} {
   # now we know we have a binding for the xname_topic exchange
}
```

Task 1: Getting Data In (Publishing)

```
proc queue_bound {rChan} {
    # get alerted if our data cannot be publish right now
    $rChan on basicReturn returned_message

    # get an ack from the server for publishing a message
    $rChan on basicAck ack_from_server

    # now we know we have a binding for the xname_topic exchange
    set pFlags [list $::rmq::PUBLISH_IMMEDIATE]
    set props [dict create correlation-id tcl-pub content-type application/pdf]
    foreach conferencePresentation $conferencePresentations {
        # args: data exchange routing flags props
        $rChan basicPublish "xname_topic" "tcl.conference.*" $pFlags $props
    }
}
```

```
proc returned_message {rChan methodData
frameData body} {
    # figure out which message was returned and
do something
}

proc ack_from_server {rChan dTag multiple} {
    # the server received what we sent and
persisted it to disk
}
```

Task 2: Getting Data Out (Consuming)

```
proc get_some_messages {rChan} {
    # consumer flags
    set cFlags [list $::rmq::CONSUME_EXCLUSIVE]

    # args: callback proc name, queue name, consumer tag, flags, props
    $rChan basicConsume consumer_cb $qName "tcl_consumer" $cFlags

    # another way of setting up consumption
    $rChan basicQos $prefetchCount
    $rChan basicConsume consumer_cb $otherQ
}
```

Task 2: Getting Data Out (Consuming), Cont.

```
proc consumer_cb {rChan methodD frameD data} {
    # for consuming from multiple queues, can dispatch on
    # method data, which includes exchange and routing key

    # delivery tag contains a numbering of the messages in
    # this session: used for acks and nacks
    set dTag [dict get $methodD
    if {[is_good $data]} {
        if {$dTag % $someMessageMultiple == 0} {
            $rChan basicAck $dTag 1
        }
    } else {
        $rChan basicNack $dTag
    }
}
```

Future Work

- Benchmarking suite
  - For publishing and consuming under high throughput
- Test case suite
  - To start, implement all tests specified in the protocol spec
- Support for additional protocols
- New features
  - More complex consumer support
  - Connection timeouts
  - Any requests / suggestions