Package 'rmq'

October 26, 2015

Title R Messaging and Queuing

Version 1.0.1

Maintainer Jason E. Aten <j.e.aten@gmail.com>

Author Jason E. Aten <j.e.aten@gmail.com>

Description Package rmq provides messaging based on msgpack and websockets. It demonstrates calling from R into Golang (Go) libraries to extend R with functionality available in Go. We use the Go library https://github.com/ugorji/go/codec for msgpack encoding and decoding. This is a high performance implementation, and supports the updated msgpack 2 (current) spec only. For websockets, we use the terrific https://github.com/gorilla/websocket library.

License Apache 2.0. Individual vendored components include their own licenses, which are Apache2, MIT, or BSD style. See the src/vendor subdirectories for details.

SystemRequirements golang 1.5.1 or later must be installed. On ubuntu, 'apt-get install golang'. On fedora, 'yum install golang'.

URL https://github.com/glycerine/rmq

BugReports https://github.com/glycerine/rmq/issues

Suggests testthat

R topics documented:

from.msgpack *create an R object from raw msgpack bytes*

Description

Given a vector of raw bytes written in msgpack format, `from.msgpack` converts these into an R object.

Usage

```
from.msgpack(x)
```

Arguments

x	A raw byte vector of msgpack formatted bytes.

Details

Lists, numeric vectors, integer vectors, string vectors, and raw byte vectors are supported.

Value

The R object represented by x.

See Also

http://msgpack.org

Other rmq.functions: `r2r.call`; `r2r.server`; `rmq.call`; `rmq.default.addr`; `rmq.server`; `to.msgpack`

r2r.call *Send an R object to a listening RMQ server.*

Description

`r2r.call()` is the client counter-part to `r2r.server()`

Usage

```
r2r.call(msg, addr = rmq.default.addr)
```

Arguments

msg	An R object. Can be a list. Internally this will be serialized using `serialize`, then converted into a msgpack binary array and sent to the server.
addr	A string of "IP:port" format. The server will bind `addr`, and it must be available. Defaults to `rmq.default.addr`, which is "127.0.0.1:9090".

See Also

Other rmq.functions: `from.msgpack`; `r2r.server`; `rmq.call`; `rmq.default.addr`; `rmq.server`; `to.msgpack`

Examples

```
## Not run:
  x=list()
  x$arg=c(1,2,3)
  x$f = function(y) { sum(y) }
  r2r.call(x)

## End(Not run)
```

`r2r.server` *Start a server expecting serialized then msgpacked R objects.*

Description

`r2r.call` calls on R's native `serialize()` function, the encodes those bytes in msgpack and sends them over to a waiting `r2r.server`, which turns them back into R objects before passing them to the handler.

Usage

```
r2r.server(handler, addr = rmq.default.addr)
```

Arguments

`handler`	A handler R function taking a single argument
`addr`	A string of "IP:port" format. The server will bind `addr`, and it must be available. Defaults to `rmq.default.addr`, which is "127.0.0.1:9090".

Details

This is an example of how to use `rmq.server` to good effect. While `rmq.server` is designed to allow cross-language messaging, it may also be the case that only R sessions wish to communicate. If both client and server speak R's XDR based serialization protocol (e.g. if both ends are R sessions), then we can `serialize()` arbitrary R objects into msgpack RAW bytes, transmit those RAW bytes, and then `unserialize()` the XDR back into full R objects. Although not-interoperable with most other languages, this does mean that we can exchange *any* R object. The msgpack support for language interop is limited to numeric arrays, string arrays, RAW arrays, integer arrays, lists, and recursively nested lists. While this level of msgpack support does cover most of the inter-language use cases, sometimes we want to serialize full R objects without restriction. For such purposes, the approach demonstrated in the r2r.server() call and the r2r.call() come in handy.

Caveat: you client-server protocol can no longer be evolved by adding new fields to the msgpack. If you want to be able to evolve your cluster gracefully over time, you may be better sticking to msgpack.

See Also

Other rmq.functions: `from.msgpack`; `r2r.call`; `rmq.call`; `rmq.default.addr`; `rmq.server`; `to.msgpack`

Examples

```
## Not run:

 ## R session 1 - start the server, giving it
 ##  a handler to call on arrival of each new message.

 handler = function(x) {
   print("handler called back with argument x = ")
   print(x)
   print("computing and returning x$f(x$arg)")
   x$f(x$arg)
 }
 r = r2r.server(handler)

 ## lastly the client call - in R session #2
  x=list()
  x$arg=c(1,2,3)
  x$f = function(y) { sum(y) }
  r2r.call(x)

## End(Not run)
```

`rmq` *R Messaging and Queuing: msgpack2 serialization and RPC over websockets*

Description

RMQ lets you do msgpack2 encoding and decoding, and provides a websocket based remote procedure call (RPC) mechanism.

Details

The basic server and client functions are `rmq.server` and `rmq.call`. The client and server communicate internally by encoding and decoding to msgpack2 bytes on the wire. Msgpack2 is the upgraded msgpack spec that distinguishes between blobs and utf8 strings.

Client and server use the websocket protocol which means the server can be accessed from the broswer-based javascript, and the calls will go through firewalls without issue. The gorilla websocket implementation supports TLS certificates for security. A user supplied R function is invoked by the server to handle each incoming client connection.

You can also make use of `to.msgpack` and `from.msgpack` diretly for situations that do not require remote procedure call or websockets.

In summary, see `to.msgpack` and `from.msgpack` for stand-alone fast msgpack2 serialization within an R session. See `rmq.server` and `rmq.call` for the basic RPC over websocket functionality that allows inter-operations with other languages using msgpack2. The default listening address is defined in `rmq.default.addr`. This overview of RMQ is found under the title `rmq`.

With a little elaboration on these basics, we can exchange arbitrary R data and functions, `r2r.server` and `r2r.call` for R-to-R object transfer. The rmq package comes with two simple example scripts that illustrate using these r2r functions. See https://github.com/glycerine/rmq/blob/master/example-client.R and https://github.com/glycerine/rmq/blob/master/example-server.R in the source package for a simple working system to get started from.

References

https://github.com/glycerine/rmq, http://msgpack.org

`rmq.call` *Send a message to a listening RMQ server.*

Description

Send a message to a listening RMQ server.

Usage

```
rmq.call(msg, addr = rmq.default.addr, timeout.msec = 5000)
```

Arguments

`msg`	An R object. Can be a list. Internally this will be converted into msgpack and sent to the server.
`addr`	A string of "IP:port" format. The server will bind `addr`, and it must be available. Defaults to `rmq.default.addr`, which is "127.0.0.1:9090".
`timeout.msec`	A timeout value in milliseconds. A value of 0 means wait forever for a reply. It is recommended to use a small finite timeout such as the 5 second default, because there is no other way to interrupt the `rmq.call()` while it is waiting on the network. Issuing ctrl-c (SIGINT) in particular will not interrupt the `rmq.call()` in progress.

Value

The return value is the response from the rmq server to the given msg.

See Also

Other rmq.functions: `from.msgpack`; `r2r.call`; `r2r.server`; `rmq.default.addr`; `rmq.server`; `to.msgpack`

Examples

```
## Not run:
  rmq.call(msg, addr="10.0.0.1:7777", timeout.msec = 1000)

## End(Not run)
```

rmq.default.addr *The default address bound by* rmq.server.

Description

The default address bound by rmq.server.

Usage

```
rmq.default.addr
```

Format

```
chr "127.0.0.1:9090"
```

See Also

Other rmq.functions: from.msgpack; r2r.call; r2r.server; rmq.call; rmq.server; to.msgpack

rmq.server *Start an RMQ server, listening on specified IP and port.*

Description

Start an RMQ server, listening on specified IP and port.

Usage

```
rmq.server(handler, addr = rmq.default.addr)
```

Arguments

handler	A handler R function taking a single argument
addr	A string of "IP:port" format. The server will bind addr, and it must be available. Defaults to rmq.default.addr, which is "127.0.0.1:9090".

Value

No return value. Blocks forever listening and calling the handler function when a request arrives. Ctrl-c will interrupt the server and shut it down. Call rms.server() again to re-start the server.

See Also

Other rmq.functions: from.msgpack; r2r.call; r2r.server; rmq.call; rmq.default.addr; to.msgpack

Examples

```
## Not run:
   ## a) the simplest echo server - run this in R session #1. Assumes port
   ##    9090 (defined in package variable rmq.default.addr) is available locally.
   rmq.server(handler=function(msg) {msg})

  ## b) This second example is a simple handler
  ## that echos the input it receives, and adds a few other things.
  ## This would also be in R session #1, as an alternative
  ## to a) above.
 handler = function(x) {
   print("handler called back with argument x = ")
   print(x)
   reply=list()
   reply$hi = "there!"
   reply$yum = c(1.1, 2.3)
   reply$input = x
   reply
 }
 r = rmq.server(handler)

 ## c) lastly the client call. In R session #2. You'll
 ## always need to run c) after first starting the
 ## the server using a) or b) above in a separate
 ## R session.
 rmq.call("hello rmq!")

 ## d) illustrate how the client call can pass complex
 ## nested list structured data.
 monster=list()
 eyes=list()
 eyes$description = c("red","glowing")
 monster$eyes = eyes
 monster$measurements = c(34, 22, 33)

 ## finally, send the monster to the server.
 rmq.call(monster)


## End(Not run)
```

`to.msgpack` *serialize an R object to raw msgpack bytes*

Description

Given an R object, `to.msgpack` will convert that object to a vector of raw bytes written in msgpack format.

Usage

```
to.msgpack(x)
```

Arguments

x	An R object to be serialized. Lists, numeric vectors, raw vectors, and string vectors are supported.

Details

Lists, numeric vectors, integer vectors, string vectors, and raw byte vectors are supported.

Value

A raw byte vector containing the msgpack serialized object.

See Also

http://msgpack.org

Other rmq.functions: from.msgpack; r2r.call; r2r.server; rmq.call; rmq.default.addr; rmq.server

Examples

```
x=list()
   x$hello = "rmq"
   raw=to.msgpack(x)
   y=from.msgpack(raw)
   ## y and x should be equal
```

Index