Package 'httpuv'

January 27, 2015

Type Package

Title HTTP and WebSocket server library

Version 1.3.2

Date 2014-10-22

Author RStudio, Inc.

Copyright RStudio, Inc.; Joyent, Inc.; Nginx Inc.; Igor Sysoev; Niels Provos; Internet Systems Consortium, Inc.; Alexander Chemeris

Maintainer Joe Cheng <joe@rstudio.com>

Description httpuv provides low-level socket and protocol support for handling HTTP and WebSocket requests directly from within R. It is primarily intended as a building block for other packages, rather than making it particularly easy to create complete web applications using httpuv alone. httpuv is built on top of the libuv and http-parser C libraries, both of which were developed by Joyent, Inc. (See LICENSE file for libuv and http-parser license information.)

License GPL-3 | file LICENSE

Depends R (>= 2.15.1), methods

Imports Rcpp (>= 0.11.0)

LinkingTo Rcpp

URL https://github.com/rstudio/httpuv

SystemRequirements GNU make

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-10-23 07:38:13

R topics documented:

`httpuv-package` *HTTP and WebSocket server*

Description

HTTP and WebSocket server

Details

Allows R code to listen for and interact with HTTP and WebSocket clients, so you can serve web traffic directly out of your R process. Implementation is based on libuv and http-parser.

This is a low-level library that provides little more than network I/O and implementations of the HTTP and WebSocket protocols. For an easy way to create web applications, try Shiny instead.

Author(s)

Joe Cheng <joe@rstudio.com>

See Also

startServer

Examples

```
## Not run:
demo("echo", package="httpuv")

## End(Not run)
```

encodeURI *URI encoding/decoding*

Description

Encodes/decodes strings using URI encoding/decoding in the same way that web browsers do. The precise behaviors of these functions can be found at developer.mozilla.org: encodeURI, encodeURIComponent, decodeURI, decodeURIComponent

Usage

```
encodeURI(value)

encodeURIComponent(value)

decodeURI(value)

decodeURIComponent(value)
```

Arguments

value	Character vector to be encoded or decoded.

Details

Intended as a faster replacement for URLencode and URLdecode.

encodeURI differs from encodeURIComponent in that the former will not encode reserved characters: ;,/?:@&=+$

decodeURI differs from decodeURIComponent in that it will refuse to decode encoded sequences that decode to a reserved character. (If in doubt, use decodeURIComponent.)

The only way these functions differ from web browsers is in the encoding of non-ASCII characters. All non-ASCII characters will be escaped byte-by-byte. If conformant non-ASCII behavior is important, ensure that your input vector is UTF-8 encoded before calling encodeURI or encodeURIComponent.

Value

Encoded or decoded character vector of the same length as the input value.

interrupt *Interrupt httpuv runloop*

Description

Interrupts the currently running httpuv runloop, meaning `runServer` or `service` will return control back to the caller and no further tasks will be processed until those methods are called again. Note that this may cause in-process uploads or downloads to be interrupted in mid-request.

Usage

```
interrupt()
```

rawToBase64 *Convert raw vector to Base64-encoded string*

Description

Converts a raw vector to its Base64 encoding as a single-element character vector.

Usage

```
rawToBase64(x)
```

Arguments

x	A raw vector.

Examples

```
set.seed(100)
result <- rawToBase64(as.raw(runif(19, min=0, max=256)))
stopifnot(identical(result, "TkGNDnd7z16LK5/hR2bDqzRbXA=="))
```

runServer *Run a server*

Description

This is a convenience function that provides a simple way to call `startServer`, `service`, and `stopServer` in the correct sequence. It does not return unless interrupted or an error occurs.

Usage

```
runServer(host, port, app, interruptIntervalMs = ifelse(interactive(), 100,
  1000))
```

Arguments

`host`	A string that is a valid IPv4 address that is owned by this server, or `"0.0.0.0"` to listen on all IP addresses.
`port`	A number or integer that indicates the server port that should be listened on. Note that on most Unix-like systems including Linux and Mac OS X, port numbers smaller than 1025 require root privileges.
`app`	A collection of functions that define your application. See `startServer`.
`interruptIntervalMs`	How often to check for interrupt. The default should be appropriate for most situations.

Details

If you have multiple hosts and/or ports to listen on, call the individual functions instead of `runServer`.

See Also

`startServer`, `service`, `stopServer`

service *Process requests*

Description

Process HTTP requests and WebSocket messages. Even if a server exists, no requests are serviced unless and until `service` is called.

Usage

```
service(timeoutMs = ifelse(interactive(), 100, 1000))
```

Arguments

timeoutMs	Approximate number of milliseconds to run before returning. If 0, then the function will continually process requests without returning unless an error occurs.

Details

Note that while service is waiting for a new request, the process is not interruptible using normal R means (Esc, Ctrl+C, etc.). If being interruptible is a requirement, then call service in a while loop with a very short but non-zero Sys.sleep during each iteration.

Examples

```
## Not run:
while (TRUE) {
  service()
  Sys.sleep(0.001)
}

## End(Not run)
```

---

startDaemonizedServer *Create an HTTP/WebSocket daemonized server (experimental)*

Description

Creates an HTTP/WebSocket server on the specified host and port. The server is daemonized so R interactive sessions are not blocked to handle requests.

Usage

```
startDaemonizedServer(host, port, app)
```

Arguments

host	A string that is a valid IPv4 address that is owned by this server, or "0.0.0.0" to listen on all IP addresses.
port	A number or integer that indicates the server port that should be listened on. Note that on most Unix-like systems including Linux and Mac OS X, port numbers smaller than 1025 require root privileges.
app	A collection of functions that define your application. See Details.

Details

In contrast to servers created by `startServer`, calls to `service` are not needed to accept and handle connections. If the port cannot be bound (most likely due to permissions or because it is already bound), an error is raised.

The app parameter is where your application logic will be provided to the server. This can be a list, environment, or reference class that contains the following named functions/methods:

`call(req)` Process the given HTTP request, and return an HTTP response. This method should be implemented in accordance with the Rook specification.

`onHeaders(req)` Optional. Similar to `call`, but occurs when headers are received. Return `NULL` to continue normal processing of the request, or a Rook response to send that response, stop processing the request, and ask the client to close the connection. (This can be used to implement upload size limits, for example.)

`onWSOpen(ws)` Called back when a WebSocket connection is established. The given object can be used to be notified when a message is received from the client, to send messages to the client, etc. See `WebSocket`.

The `startPipeServer` variant is not supported yet.

Value

A handle for this server that can be passed to `stopDaemonizedServer` to shut the server down.

See Also

`startServer`

`startServer` *Create an HTTP/WebSocket server*

Description

Creates an HTTP/WebSocket server on the specified host and port.

Usage

```
startServer(host, port, app)

startPipeServer(name, mask, app)
```

Arguments

host	A string that is a valid IPv4 address that is owned by this server, or "0.0.0.0" to listen on all IP addresses.
port	A number or integer that indicates the server port that should be listened on. Note that on most Unix-like systems including Linux and Mac OS X, port numbers smaller than 1025 require root privileges.
app	A collection of functions that define your application. See Details.
name	A string that indicates the path for the domain socket (on Unix-like systems) or the name of the named pipe (on Windows).
mask	If non-NULL and non-negative, this numeric value is used to temporarily modify the process's umask while the domain socket is being created. To ensure that only root can access the domain socket, use strtoi("777", 8); or to allow owner and group read/write access, use strtoi("117", 8). If the value is NULL then the process's umask is left unchanged. (This parameter has no effect on Windows.)

Details

startServer binds the specified port, but no connections are actually accepted. See service, which should be called repeatedly in order to actually accept and handle connections. If the port cannot be bound (most likely due to permissions or because it is already bound), an error is raised.

The app parameter is where your application logic will be provided to the server. This can be a list, environment, or reference class that contains the following named functions/methods:

call(req) Process the given HTTP request, and return an HTTP response. This method should be implemented in accordance with the Rook specification.

onHeaders(req) Optional. Similar to call, but occurs when headers are received. Return NULL to continue normal processing of the request, or a Rook response to send that response, stop processing the request, and ask the client to close the connection. (This can be used to implement upload size limits, for example.)

onWSOpen(ws) Called back when a WebSocket connection is established. The given object can be used to be notified when a message is received from the client, to send messages to the client, etc. See WebSocket.

The startPipeServer variant can be used instead of startServer to listen on a Unix domain socket or named pipe rather than a TCP socket (this is not common).

Value

A handle for this server that can be passed to stopServer to shut the server down.

See Also

runServer

stopDaemonizedServer *Stop a running daemonized server in Unix environments*

Description

Given a handle that was returned from a previous invocation of `startDaemonizedServer`, closes all open connections for that server, removes listeners in the R event loop and unbinds the port. **Be careful not to call `stopDaemonizedServer` more than once on a handle, as this will cause the R process to crash!**

Usage

```
stopDaemonizedServer(server)
```

Arguments

`server`	A handle that was previously returned from `startDaemonizedServer`.

stopServer *Stop a running server*

Description

Given a handle that was returned from a previous invocation of `startServer`, closes all open connections for that server and unbinds the port. **Be careful not to call `stopServer` more than once on a handle, as this will cause the R process to crash!**

Usage

```
stopServer(handle)
```

Arguments

`handle`	A handle that was previously returned from `startServer`.

`WebSocket-class` *WebSocket object*

Description

An object that represents a single WebSocket connection. The object can be used to send messages and close the connection, and to receive notifications when messages are received or the connection is closed.

Arguments

...	For internal use only.

Details

WebSocket objects should never be created directly. They are obtained by passing an `onWSOpen` function to `startServer`.

Fields

`request` The Rook request environment that opened the connection. This can be used to inspect HTTP headers, for example.

Methods

`onMessage(func)` Registers a callback function that will be invoked whenever a message is received on this connection. The callback function will be invoked with two arguments. The first argument is `TRUE` if the message is binary and `FALSE` if it is text. The second argument is either a raw vector (if the message is binary) or a character vector.

`onClose(func)` Registers a callback function that will be invoked when the connection is closed.

`send(message)` Begins sending the given message over the websocket. The message must be either a raw vector, or a single-element character vector that is encoded in UTF-8.

`close()` Closes the websocket connection.

Index