Package 'httpuv'

December 18, 2018

Type Package

Title HTTP and WebSocket Server Library

Version 1.4.5.1

Author Joe Cheng, Hector Corrada Bravo [ctb], Jeroen Ooms [ctb], Winston Chang [ctb]

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

Maintainer Joe Cheng <joe@rstudio.com>

Description 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 (>= 2) | file LICENSE

Depends R (>= 2.15.1), methods

Imports Rcpp (>= 0.11.0), utils, promises, later (>= 0.7.3)

LinkingTo Rcpp, BH, later

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

SystemRequirements GNU make

RoxygenNote 6.0.1.9000

Suggests testthat, callr

NeedsCompilation yes

Repository CRAN

Date/Publication 2018-12-18 19:40:03 UTC

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()
```

ipFamily *Check whether an address is IPv4 or IPv6*

Description

Given an IP address, this checks whether it is an IPv4 or IPv6 address.

Usage

```
ipFamily(ip)
```

Arguments

`ip`	A single string representing an IP address.

Value

For IPv4 addresses, `4`; for IPv6 addresses, `6`. If the address is neither, `-1`.

Examples

```
ipFamily("127.0.0.1")   # 4
ipFamily("500.0.0.500") # -1
ipFamily("500.0.0.500") # -1

ipFamily("::")          # 6
ipFamily("::1")         # 6
ipFamily("fe80::1ff:fe23:4567:890a") # 6
```

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 = NULL)
```

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	Deprecated (last used in httpuv 1.3.5).

Details

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

See Also

`startServer`, `service`, `stopServer`

Examples

```
## Not run:
# A very basic application
runServer("0.0.0.0", 5000,
  list(
    call = function(req) {
      list(
        status = 200L,
        headers = list(
          'Content-Type' = 'text/html'
        ),
        body = "Hello world!"
      )
    }
  )
)

## End(Not run)
```

`service` *Process requests*

Description

Process HTTP requests and WebSocket messages. If there is nothing on R's call stack – if R is sitting idle at the command prompt – it is not necessary to call this function, because requests will be handled automatically. However, if R is executing code, then requests will not be handled until either the call stack is empty, or this function is called (or alternatively, `run_now` is called).

Usage

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

Arguments

`timeoutMs`	Approximate number of milliseconds to run before returning. It will return this duration has elapsed. If 0 or Inf, then the function will continually process requests without returning unless an error occurs. If NA, performs a non-blocking run without waiting.

Details

In previous versions of httpuv (1.3.5 and below), even if a server created by `startServer` exists, no requests were serviced unless and until `service` was called.

This function simply calls `run_now()`, so if your application schedules any `later` callbacks, they will be invoked.

Examples

```
## Not run:
while (TRUE) {
  service()
}

## End(Not run)
```

---

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

Description

This function will be removed in a future release of httpuv. It is simply a wrapper for `startServer`. In previous versions of httpuv (1.3.5 and below), `startServer` ran applications in the foreground and `startDaemonizedServer` ran applications in the background, but now both of them run applications in the background.

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.

`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 and listens for connections on an thread running in the background. This background thread handles the I/O, and when it receives a HTTP request, it will schedule a call to the user-defined R functions in `app` to handle the request. This scheduling is done with `later()`. When the R call stack is empty – in other words, when an interactive R session is sitting idle at the command prompt – R will automatically run the scheduled calls. However, if the call stack is not empty – if R is evaluating other R code – then the callbacks will not execute until either the call stack is empty, or the `run_now()` function is called. This function tells R to execute any callbacks that have been scheduled by `later()`. The `service()` function is essentially a wrapper for `run_now()`.

In older versions of httpuv (1.3.5 and below), it did not use a background thread for I/O, and when this function was called, it did not accept connections immediately. It was necessary to call `service` 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.Note that httpuv augments `req` with an additional item, `req$HEADERS`, which is a named character vector of request headers.

`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

`stopServer`, `runServer`

Examples

```
## Not run:
# A very basic application
handle <- startServer("0.0.0.0", 5000,
  list(
    call = function(req) {
      list(
        status = 200L,
        headers = list(
          'Content-Type' = 'text/html'
        ),
        body = "Hello world!"
      )
    }
  )
)

stopServer(handle)

## End(Not run)
```

stopAllServers *Stop all applications*

Description

This will stop all applications which were created by `startServer` or `startPipeServer`.

Usage

```
stopAllServers()
```

See Also

`stopServer` to stop a specific server.

stopDaemonizedServer *Stop a running daemonized server in Unix environments (deprecated)*

Description

This function will be removed in a future release of httpuv. Instead, use `stopServer`.

Usage

```
stopDaemonizedServer(handle)
```

Arguments

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

stopServer *Stop a server*

Description

Given a handle that was returned from a previous invocation of `startServer` or `startPipeServer`, this closes all open connections for that server and unbinds the port.

Usage

```
stopServer(handle)
```

Arguments

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

See Also

stopAllServers to stop all servers.

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