Package: porcelain (via r-universe)

September 26, 2024

Title Turn a Package into an HTTP API

Version 0.1.15

Description Wrapper around the plumber package to turn a package into an HTTP API. This adds some conventions that we find useful, such as some testing infrastructure and automatic validation of responses against a json schema.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

URL https://github.com/reside-ic/porcelain

BugReports https://github.com/reside-ic/porcelain/issues

Depends R (>= 3.6.0)

Imports R6, V8, ids, jsonlite, jsonvalidate (>= 1.2.2), lgr, plumber

Suggests callr, fs, httr, knitr, mockery, pkgload, rmarkdown, roxygen2, testthat, withr

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

VignetteBuilder knitr

Remotes ropensci/jsonvalidate

Config/testthat/edition 3

Repository https://vimc.r-universe.dev

RemoteUrl https://github.com/reside-ic/porcelain

RemoteRef master

RemoteSha 655da3517c1a795b4498b1b3e0b557ad8c9b3464

Contents

`porcelain` *A* `porcelain` *object*

Description

A `porcelain` object. This extends (via inheritance) a plumber object, and so only changes to the plumber API are documented here.

Super classes

`plumber::Hookable` -> `plumber::Plumber` -> `porcelain`

Methods

Public methods:

- `porcelain$new()`
- `porcelain$include_package_endpoints()`
- `porcelain$handle()`
- `porcelain$request()`
- `porcelain$clone()`

Method `new()`**:** Create a porcelain object

Usage:

```
porcelain$new(..., validate = FALSE, logger = NULL)
```

Arguments:

`...` Parameters passed to `plumber`

`validate` Logical, indicating if any validation (implemented by the `validate_response` argument) should be enabled. This should be set to `FALSE` in production environments. By default (if `validate` is `NULL`), we look at the value of the environment `PORCELAIN_VALIDATE` - if `true` (case insensitive) then we will validate. This is intended to support easy use of validation on continuous integration systems.

`logger` Optional logger, from the `lgr` package, perhaps created with porcelain_logger. If given, then we will log at the beginning and end of the request.

Method `include_package_endpoints()`**:** Include package endpoints

Usage:

```
porcelain$include_package_endpoints(state = NULL, package = NULL)
```

Arguments:

`state` A named list of state, if your package requires any state-binding endpoints. Typically these will be mutable state (database connections, job queues, or similar). You must provide all states as required by the combination of all endpoints.

`package` Either a package name or environment (optional, usually we'll find the right thing)

Method `handle()`**:** Handle an endpoint

Usage:

```
porcelain$handle(...)
```

Arguments:

`...` Either a single argument, being a `porcelain_endpoint` object representing an endpoint, or arguments to pass through to `plumber`.

Method `request()`**:** Send a request to plumber for debugging

Sends a request to plumber so that the API can be easily tested without running the whole API. The interface here will probably change, and may end up using the interface of `httr`.

Usage:

```
porcelain$request(
  method,
  path,
  query = NULL,
  body = NULL,
  content_type = NULL,
  request_id = NULL
)
```

Arguments:

`method` Name of HTTP method to use (e.g., `GET`)

`path` Path to send the request to

`query` Optional query parameters as a named list or character vector.

`body` Optional body (only valid with `PUT`, `POST`, etc).

`content_type` Optional content type (mime) which can be provided alongside body. If not provided it is set to `application/octet-stream` if `body` is raw, or `application/json` otherwise.

`request_id` Optional request ID. An ID which is attached to every log raised by this request. Used for tracing purposes.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
porcelain$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`porcelain_add_headers` *Add headers to endpoint output data*

Description

Intended to be used from endpoint target function. Note `Content-Type` headers are handled by returning arg to endpoint.

Usage

```
porcelain_add_headers(data, headers)
```

Arguments

`data`	Response data
`headers`	Named list of headers to add.

Value

Data from endpoint target with headers

Examples

```
porcelain_add_headers("output",
                  list("Content-Dispotition" = "output_file.txt"))
```

`porcelain_background` *While porcelain makes it easy to test endpoints individually, you may still want some integration or end-to-end tests where you bring the entire API up and interact with it from your tests. This class provides a helper for doing this in a way that is reasonably tidy.*

Description

While porcelain makes it easy to test endpoints individually, you may still want some integration or end-to-end tests where you bring the entire API up and interact with it from your tests. This class provides a helper for doing this in a way that is reasonably tidy.

While porcelain makes it easy to test endpoints individually, you may still want some integration or end-to-end tests where you bring the entire API up and interact with it from your tests. This class provides a helper for doing this in a way that is reasonably tidy.

Public fields

`log` The path to the log file (read-only)

`port` The port used by the background server (read-only)

Methods

Public methods:

- `porcelain_background$new()`
- `porcelain_background$start()`
- `porcelain_background$status()`
- `porcelain_background$stop()`
- `porcelain_background$url()`
- `porcelain_background$request()`

Method `new()`**:** Create a background server object

Usage:

```
porcelain_background$new(
  create,
  args = NULL,
  port = NULL,
  log = NULL,
  verbose = FALSE,
  timeout = 60,
  env = NULL
)
```

Arguments:

`create` A function that will create an api object

`args` Arguments that will be passed to `create` when creating the api object in the background process

`port` The port to use for the background server. If not given then a random free port will be used in the range 8000 to 10000 - you can find the created port using the `port` field in the resulting object, or use the `$url()` or `$request()` methods.

`log` The path to a log file to use

`verbose` Logical, indicating if we should print informational messages to the console on start/stop etc.

`timeout` The number of seconds to wait for the server to become available. This needs to cover the time taken to spawn the R process, and create your API object (loading all packages needed) up to the point where the server is responsive. In most cases this will take 1-2s but if you use packages that use many S4 methods or run this on a slow computer (e.g., a continuous integration server) it may take longer than you expect. The default is one minute which should be sufficient in almost all cases.

`env` A named character vector of environment variables (e.g., `c(VARIABLE = "value")`) to set in the background process before launching the server. You can use this to control the behaviour of the background server using variables your api recognises. In addition, we export `callr::rcmd_safe_env()` and the value of `PORCELAIN_VALIDATE`.

Method `start()`**:** Start the server. It is an error to try and start a server that is already running.

Usage:

```
porcelain_background$start()
```

Method `status()`**:** Return the background server status. This will be one of:

- `running`: The server is running
- `stopped`: The server is stopped
- `blocked`: The server is stopped, but something else is running on the port that we would use
- `starting`: The server is starting up (not visible in normal usage)

Usage:

```
porcelain_background$status()
```

Method `stop()`**:** Stop a running server. If the server is not running, this has no effect.

Usage:

```
porcelain_background$stop()
```

Method `url()`**:** Create a url string for the server, interpolating the (possibly random) port number. You can use this in your tests like `bg$url("/path")`

Usage:

```
porcelain_background$url(path)
```

Arguments:

`path` String representing the absolute path

Method `request()`**:** Run a request to the server, using `httr`. This presents a similar inteface to the `request` method on the porcelain object.

Usage:

```
porcelain_background$request(method, path, ...)
```

Arguments:

`method` The http method as a string (e.g., `"GET"`), passed to httr::VERB as the `verb` argument

`path` String representing the absolute path, passed to `$url()`

`...` Additional arguments passed to `httr::VERB`, such as `query`, or the body for a `POST` request.

`porcelain_endpoint` *Basic endpoint object*

Description

Create a `porcelain_endpoint` object that collects together an HTTP method (e.g., `GET`), a path (e.g., `/path`) and a target R function. Unlike plumber endpoints, porcelain endpoints are meant to be used in testing.

Public fields

`method` HTTP method

`path` HTTP path

`target` R function used for the endpoint

`validate` Logical, indicating if response validation is used

`inputs` Input control

`state` Possibly mutable state

`returning` An `porcelain_returning` object controlling the return type (content type, status code, serialisation and validation information).

Methods

Public methods:

- `porcelain_endpoint$new()`
- `porcelain_endpoint$run()`
- `porcelain_endpoint$request()`
- `porcelain_endpoint$plumber()`
- `porcelain_endpoint$create()`
- `porcelain_endpoint$clone()`

Method `new()`**:** Create an endpoint

Usage:

```
porcelain_endpoint$new(method, path, target, ..., returning, validate = NULL)
```

Arguments:

`method` The HTTP method to support

`path` The server path for the endpoint

`target` An R function to run as the endpoint

... Additional parameters, currently representing *inputs*. You can use the functions porcelain_input_query, porcelain_input_body_binary and porcelain_input_body_json to define inputs and pass them into this method. The names used must match those in target.

returning Information about what the endpoint returns, as created by porcelain_returning

validate Logical, indicating if any validation (implemented by the validate_response argument) should be enabled. This should be set to FALSE in production environments. By default (if validate is NULL), we look at the value of the environment PORCELAIN_VALIDATE - if true (case insensitive) then we will validate. This is intended to support easy use of validation on continuous integration systems.

validate_response Optional function that throws an error of the processed body is "invalid".

Method `run()`**:** Run the endpoint. This will produce a standardised response object that contains status_code, content_type, body (the serialised output as run through the process method and returned by plumber) and data (the result of running the target function)

Usage:

```
porcelain_endpoint$run(...)
```

Arguments:

... Arguments passed through to the target function

Method `request()`**:** Test the endpoint. This creates a full plumber object and serves one request to the endpoint. Argument are as passed through to porcelain's $request() method, except that method and path are automatically taken from the endpoint itself.

Usage:

```
porcelain_endpoint$request(...)
```

Arguments:

... Arguments passed through to the request method (query, body and content_type).

Method `plumber()`**:** Helper method for use with plumber - not designed for end-user use. This is what gets called by plumber when the endpoint receives a request.

Usage:

```
porcelain_endpoint$plumber(req, res, ...)
```

Arguments:

req, res Conventional plumber request/response objects

... Additional arguments passed through to run

Method `create()`**:** Create a plumber endpoint

Usage:

```
porcelain_endpoint$create(envir, validate)
```

Arguments:

envir Environment as used by plumber (currently unclear)

validate Logical, allowing override of validation at the api level. This takes precedence over the value set when creating the endpoint.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
porcelain_endpoint$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`porcelain_input_body_binary` *Control for body parameters*

Description

Control for body parameters. This might change. There are several types of HTTP bodies that we want to consider here - the primary ones are a body uploaded in binary, the other is a json object. In the latter we want to validate the body against a schema (at least if validation is used). In future we might also support a form input here too.

Usage

```
porcelain_input_body_binary(name, content_type = NULL)

porcelain_input_body_json(name, schema = NULL, root = NULL, extract = NULL)
```

Arguments

`name`	Name of the parameter
`content_type`	Content type for the input. If not given, then `application/octet-stream` is used. Provide a vector of valid types to allow any of the types to be passed.
`schema`	The name of the json schema to use
`root`	The root of the schema directory.
`extract`	Optionally, the name of an element to extract from the json. If given, then the body must be a json object (not an array, for example) and `extract` must refer to a top-level key within it. We will extract the *JSON string* corresponding to this key and forward that to the argument `name`. Deserialisation of the json is still the target function's responsibility but there will be less of it.

porcelain_input_query *Control for query parameters*

Description

Control for query parameters.

Usage

```
porcelain_input_query(..., .parameters = list(...))
```

Arguments

`...`	Named arguments representing accepted parameters. The value of each must be a type.
`.parameters`	A list of named parameters to accept, instead of using `...` - this interface is considerably easier to program against if building an API programmatically, avoiding the use of `do.call`.

Examples

```
porcelain::porcelain_input_query(number = "integer")
```

porcelain_logger *Create logger*

Description

Create a json-emitting logger, using the 'lgr' package.

Usage

```
porcelain_logger(log_level = "info", name = NULL, path = NULL)
```

Arguments

`log_level`	The level of detail to log to. See `lgr::get_log_levels()` for possible values; this could be a string ("off", "info", "all", etc) or an integer level.
`name`	The name of the logger. By default we use one derived from the package name, though this may not always be accurate.
`path`	Optionally, the path to log into. If not given then we log to the console.

Value

A "Logger" object (see lgr::Logger)

Examples

```
logger <- porcelain::porcelain_logger(name = "example")
logger$log("info", "hello")
logger$log("trace", "silent")
```

`porcelain_package_endpoint` *Find roxygen-defined endpoint*

Description

Find an endpoint defined implicitly through roxygen comments (rather than explicitly via writing porcelain_endpoint.

Usage

```
porcelain_package_endpoint(
  package,
  method,
  path,
  state = NULL,
  validate = NULL
)
```

Arguments

`package`	The name of the package to look in, provided as a string or as a namespace
`method`	The HTTP method (i.e., verb), such as `GET` or `POST`, as a string
`path`	The path of the method (e.g., `/my/path`)
`state`	A list of state to bind into the method, if your endpoint requires any
`validate`	Logical, indicating if the method should be created with schema validation enabled.

Value

The endpoint, a porcelain_endpoint object

porcelain_returning *Support for endpoint return types*

Description

Support for describing and controlling expected return types. The high-level functions (`porcelain_returning_json` and `porcelain_returning_binary`) should be generally used.

Usage

```
porcelain_returning(content_type, process, validate, status_code = 200L)

porcelain_returning_json(schema = NULL, root = NULL, status_code = 200L)

porcelain_returning_binary(status_code = 200L)

porcelain_returning_text(status_code = 200L)
```

Arguments

`content_type`	The MIME content type for the endpoint, e.g. `text/plain`, `application/json`.
`process`	A processing function that will convert the output of the handler function into something of the type `content_type`. This should be independent of arguments passed to the endpoint, so practically this is the final stage of serialisation.
`validate`	A function that validates the return value and throws an error if the output is not expected. This will only be used if the endpoint is created with `validate = TRUE`.
`status_code`	The HTTP status code that the endpoint will use on a successful return. The default of 200 should be reasonable.
`schema`	The name of the json schema to use
`root`	The root of the schema directory.

porcelain_roclet *Define API using roxygen tags*

Description

A roclet for processing `@porcelain` tags within a package. This presents an automated declarative approach to defining porcelain APIs using roxygen tags. When you roxygenise your package (e.g., with `devtools::document()` or `roxygen2::roxygenise()`) this roclet will create a file `R/porcelain.R` within your package that will be included into your package API.

Usage

```
porcelain_roclet()
```

Value

A roclet, used by `roxygen2` (not typically called by users directly)

`porcelain_state` *Bind state into an endpoint*

Description

Bind state into an endpoint

Usage

```
porcelain_state(..., .state = list(...))
```

Arguments

`...`	Named arguments representing state to bind; see `Details`.
`.state`	A list of named state to bind, instead of using `...` - this interface is considerably easier to program against if building an API programmatically, avoiding the use of `do.call`.

Details

This method allows state to be bound to the target function. Each element of `...` (or `.state`) is named with the argument to the target function being bound, and the value is the value that argument will take. Once bound, the arguments to the target function may not be provided by an input.

The primary use case for this is to bind mutable state (database connections, etc) that may be shared amongst different endpoints within an API.

`porcelain_stop` *Throw an error from an endpoint*

Description

Throw an error from an endpoint. This function is intended to allow target functions to throw nice errors back through the API.

Usage

```
porcelain_stop(message, code = "ERROR", errors = NULL, status_code = 400L, ...)
```

Arguments

message	The human-readable message of the error. Ignored if errors is given.
code	Optional code for the error - if not given, then ERROR is used. Ignored if errors is given.
errors	A named list of errors - use this to signal multiple error conditions as key/value pairs.
status_code	The HTTP status code to use. The default (400) means "bad request" which should be a reasonable catch-all for bad user data.
...	Additional named args to be included as fields in the error response JSON. The values must be in format ready for serialization to JSON using jsonlite::toJSON() i.e. any unboxing using jsonlite::unbox() needs to already have been done.

Value

Nothing, as this function throws an error

Index