Krait Documentation

Release 0.5.2

TED_996

Jun 27, 2017

Contents

CHAPTER 1

krait package

Module contents

This is the root of the Krait API. This has two purposes:

- As a package, this holds the different modules that you can use to interact with Krait, namely *krait.config*, *krait.mvc*, *krait.cookie* and *krait.websockets*.
- As a (pseudo-)module, this holds generic data related to the HTTP request, or site configuration. While this may not be the best practice, given Python's poor concurrency, the fact that any process serves at most one request at a time, this is safe. Krait emphasizes ease of use over absolute purity.

Reference

krait.**site_root** = **None**
str – The site root (lifted directly from the krait argument)

krait.**get_full_path**(*filename*)
Convert a filename relative to the site root to its full path. Note that this is not necessarily absolute, but is derived from the krait argument.

Parameters **filename** (*str*) – a filename relative to the site root (the krait argument)

Returns `os.path.join(krait.site_root, filename)`

Return type str

krait.**request** = **None**
krait.Request – The HTTP request that is being handled right now. Set by Krait.

krait.**response** = **None**
krait.Response – Set this to completely change the HTTP response, leave None for usual behaviour. Useful when you want, for example, to return a *400 Bad Request* or *302 Found*. If you only want to add or change headers, use *krait.extra_headers*.

class krait.**Request**(*http_method*, *url*, *query_string*, *http_version*, *headers*, *body*)
Bases: object

Represents an HTTP request. Objects of this class are created by Krait before passing control to your Python code. The variable *krait.request* is an instance of this class.

Parameters

- **http_method** (*str*) – The HTTP method in the request. Values: 'GET', 'POST', etc.
- **url** (*str*) – The URL of the requests, without the query
- **query_string** (*str*) – The URL query
- **http_version** (*str*) – the HTTP version; only 'HTTP/1.1' is supported
- **headers** (*dict of str str*) – The HTTP headers sent by the client
- **body** (*str*) – The body of the request

http_method
str – The HTTP method in the request. Values: 'GET', 'POST', etc.

url
str – The URL of the requests, without the query

query
dict of str str – The query extracted from the URL, parsed in a dict.

class **MultipartFormData**(*data*, *name*, *filename*, *content_type*)
Bases: tuple

content_type
Alias for field number 3

data
Alias for field number 0

filename
Alias for field number 2

name
Alias for field number 1

Request.**get_multipart_form**()
Extract an HTTP multipart form from the request body.

Returns the HTTP form parts.

Return type list of *Request.MultipartFormData*

Request.**get_post_form**()
Extract the HTTP form from a POST request. This is done on demand.

Returns the HTTP form data.

Return type dict of str str

class krait.**Response**(*http_version*, *status_code*, *headers*, *body*)
Bases: object

Represents an HTTP response. Set *krait.response* with an instance of this variable (or its subclasses) to override the HTTP response.

Parameters

- **http_version** (*str*) – 'HTTP/1.1', no other values are supported.

- **status_code** (*int*) – The HTTP status code (for example, 200 or 404).
- **headers** (*list of (str, str)*) – The response headers.
- **body** – The response body.

http_version
str – The HTTP version of the response.

status_code
int – The HTTP status code.

headers
list of (str, str) – The response headers.

body
The response body.

class krait.**ResponseNotFound**(*headers=None*)
Bases: krait._http_response.Response

Response returning a 404 Not Found

Parameters **headers** (*list of (str, str), optional*) – Extra headers to send with the response.

class krait.**ResponseBadRequest**(*headers=None*)
Bases: krait._http_response.Response

Response returning 400 Bad Request

Parameters **headers** (*list of (str, str), optional*) – Extra headers to send with the response.

class krait.**ResponseRedirect**(*destination*, *headers=None*)
Bases: krait._http_response.Response

Response returning a 302 Found redirect.

Parameters

- **destination** (*str*) – The URL on which to redirect the client.
- **headers** (*list of (str, str), optional*) – Extra headers to send with the response.

krait.**extra_headers = None**
list of (str, str) – Extra response headers to set without overriding the entire response.

krait.**set_content_type**(*raw=None*, *ext=None*)
Set the HTTP Content-Type to a custom value. Used when the original route target's extension is not relevant to the extension of the final content.

Parameters

- **raw** (*str, optional*) – full MIME type (e.g. `'application/json'`)
- **ext** (*str, optional*) – file extension from which to derive the MIME type (e.g. `'json'`)

Submodules

krait.config module

This module is used to configure the behaviour of Krait when a request arrives and when its response is sent. Currently, this module is used to configure routing and the client-side cache.

The members of this module should only be changed from `.py/init.py`, later changes are ignored.

Reference

class `krait.config.`**Route**(*verb=None*, *url=None*, *regex=None*, *target=None*)
Bases: `object`

Signifies a route that incoming requests can take.

Parameters

- **verb** (*str, optional*) – The routing verb (most HTTP verbs, plus `ANY` and `WEBSOCKET`). See *Route.route_verbs* for options; GET by default
- **url** (*str, optional*) – The URL to match, or None to skip
- **regex** (*str, optional*) – The regex to match, or None to skip
- **target** (*str, optional*) – The target of the route, or None to keep default target (extracted from URL)

verb
str – The routing verb

url
str, optional – The URL to match, or None to skip

regex
str, optional – The regex to match, or None to skip

target
str, optional – The target of the route. None keeps default target (extracted from the URL)

route_verbs = ['GET', 'POST', 'PUT', 'DELETE', 'CONNECT', 'OPTIONS', 'TRACE', 'ANY', 'WEBSOCKET']
The route verb options.

`krait.config.`**routes = None**
list of *Route* – The list of routes to be read by Krait. The default is all GETs to default targets, deny anything else. If you override this, the last route *should* be a default one (`Route()`) to allow GET requests to reach resources that aren't explicitly routed (like CSS or Javascript files)

`krait.config.`**cache_no_store = []**
list of str – The list of filename regexes that the clients shouldn't cache.

`krait.config.`**cache_private = []**
list of str – The list or filename regexes that only private (client) caches should keep. This usually means that browser caches can keep the resource, but not shared caches.

`krait.config.`**cache_public = []**
list of str – The list of filename regexes that can be kept in any caches.

`krait.config.`**cache_long_term = []**
list of str – The list of filename regexes that can be kept for a longer time. This duration is configured with

krait.config.cache_max_age_long_term This is a modifier, so it can be applied to both private or public cache directives.

krait.config.**cache_max_age_default = 300**
int – The number of seconds that clients should not re-request the resource. This corresponds to the Max-Age of the HTTP response, for public or private (and not long-term) cached resources.

krait.config.**cache_max_age_long_term = 864000**
int – The number of seconds that clients should not re-request the resource, for long-term cached resources. This corresponds to the Max-Age of the HTTP responses, for long-term, public or private, cached resources.

krait.mvc module

This module interfaces Krait's MVC support with your Python backend. MVC support is implemented by having a Python script act as a route target, that creates and specifies a controller object that holds the properties used in the finished page. This page's code is provided by a template file, that the controller requests. Templates are files with Pyml syntax embedded inside that accesses properties on the controller. The rest of the code is usually HTML, although it can be JSON, JavaScript, or anything else.

Usage

See the tutorial (TODO) for a more detailed explanation.

1. **Create a template (a view):** Create a file with the `.html` or `.pyml` extension in a hidden directory in your site root (typically `.view`) with your template code.
2. **Create a controller:** Create a subclass of *CtrlBase* and override its *CtrlBase.get_view* method to return the **relative** path of a template. To add properties to the controller, simply set them in its `__init__` method.
3. **Reference properties on the controller in templates:** In template files the `ctrl` variable refers to the active controller. Use Pyml syntax to access its members and use them on the page.
4. **Set a controller to handle a specific URL** Create a Python script with the appropriate location and name to be reached by the URL. Import `krait.mvc`, then call *set_init_ctrl*, passing as an argument an object of your controller type. Krait will then call its overridden *CtrlBase.get_view* method and render the template.

Reference

class krait.mvc.**CtrlBase**
Bases: `object`

The base of a MVC controller. Abstract, implementations have to override the *get_view* method.

get_view()
Get the view that renders the response.

Returns a **relative** path

Return type str

class krait.mvc.**SimpleCtrl**(*view*, *members*)
Bases: *krait.mvc.CtrlBase*

A simple controller wrapper. Best not to use, controllers should set their views themselves.

Parameters

- **view** (*str*) – The view that this controller renders with.
- **members** (*dict*) – The attributes of the controller. These will be set dynamically as instance attributes.

view
str – The view that this controller renders with.

Apart from the view, other attributes exist, specified in the `members` dictionary.

get_view()
Get the selected view.

Returns *view*

Return type str

krait.mvc.**init_ctrl** = **None**
CtrlBase – The controller to be used as a master controller. Do not use directly, use *set_init_ctrl*. Set from route targets that want to invoke a controller (and render a response using MVC).

krait.mvc.**set_init_ctrl**(*ctrl*)
Invoke a controller after the route target has finished executing.

Parameters **ctrl** (*CtrlBase*) – The controller object to be used.

krait.mvc.**ctrl_stack** = **[]**
list of *CtrlBase* – The stack of controllers, used with nested controllers. Semi-deprecated. Do not use directly, use *push_ctrl* and *pop_ctrl*.

krait.mvc.**curr_ctrl** = **None**
CtrlBase – The current controller, used in controller stacking. Semi-deprecated. Do not use directly, use *push_ctrl* and *pop_ctrl*.

krait.mvc.**push_ctrl**(*new_ctrl*)
Save the current controller and set a new one.

Parameters **new_ctrl** (*CtrlBase*) – The new controller.

Returns The new controller.

Return type *CtrlBase*

krait.mvc.**pop_ctrl**()
Discard the current controller and set the one before it.

Returns The old, now current, controller.

Return type *CtrlBase*

krait.cookie module

This module is used to wrap the usage of cookies in websites. This prevents the need to manipulate the HTTP headers manually.

Usage

1. **Get request cookies:** Use `cookie.get_cookie` or `cookie.get_cookies` to get all
2. **Set cookies:** Create a new `cookie.Cookie` object, optionally add attributes to it, then use `cookie.set_cookie` to make it be sent with the HTTP response.

3. **Get response cookies:** Use `cookie.get_response_cookies`.

Reference

class `krait.cookie.`**Cookie**(*name*, *value*, *attributes=None*)
Represents an HTTP cookie.

Parameters

- **name** (*str*) – The name of the cookie
- **value** (*str*) – The value of the cookie.
- **attributes** (list of *CookieAttribute*, optional) – A (possibly empty) list of attributes. Can be updated later.

name
str – The name of the cookie.

value
str – The value of the cookie.

attributes
list of *CookieAttribute* – A (possibly empty) list of attributes. Update only with instance methods.

add_attribute(*attribute*)
Add an attribute to the cookie.

:param attribute *CookieAttribute*: The attribute to add.

remove_attribute(*name*)
Remove an attribute from the cookie.

Parameters **name** (*str*) – The name of the attribute to remove (e.g. `Expires`).

set_expires(*expires_datetime*)
Set or remove the *Expires* attribute on the cookie. This makes the cookie delete itself after a certain time.

Parameters **expires_datetime** (`datetime.datetime`, optional) – The **UTC**/timezoned time of expiration, or None to remove.

set_max_age(*max_age*)
Set or remove the *Max-Age* attribute on the cookie. This makes the cookie delete itself after a certain number of seconds.

> **Warning:** This attribute is not supported by all browsers. Notably, Internet Explorer does not respect it.

Parameters **max_age** (*int, optional*) – The maximum time that a cookie can be kept, in seconds, or None to remove.

set_path(*path*)
Set or remove the *Path* attribute on the cookie. This restricts the cookie only to one URL and its descendants.

Parameters **path** (*str, optional*) – The path, or None to remove.

set_domain(*domain*)
Set or remove the *Domain* attribute on the cookie. This restricts the domain on which the cookie can be sent by the client.

Parameters **domain** (*str, optional*) – The domain, or None to remove.

set_secure(*is_secure*)
Set or remove the *Secure* attribute on the cookie. This causes the cookie to only be sent over HTTPS (not yet supported by Krait).

Parameters **is_secure** (*bool*) – True to set the attribute, False to remove it.

set_http_only(*is_http_only*)
Set or remove the *HTTPOnly* attribute on the cookie. This causes the cookie to be inaccessible from Javascript.

Parameters **is_http_only** (*bool*) – True to set the attribute, False to remove it.

class krait.cookie.**CookieAttribute**(*name*, *value*)
Bases: object

A generic cookie attribute.

Parameters

- **name** (*str*) – The name of the attribute.
- **value** (*str, optional*) – The value of the attribute.

name
str – The name of the attribute.

value
str – The value of the attribute.

class krait.cookie.**CookieExpiresAttribute**(*expire_datetime*)
Bases: *krait.cookie.CookieAttribute*

Sets the *Expires* attribute on the cookie. This makes the cookie delete itself after a certain time.

Parameters **expire_datetime** (datetime.datetime) – the moment that the cookie expires at

expire_datetime
datetime.datetime – the moment that the cookie expires at

class krait.cookie.**CookieMaxAgeAttribute**(*max_age*)
Bases: *krait.cookie.CookieAttribute*

Sets the *Max-Age* attribute on the cookie. This makes the cookie delete itself after a certain number of seconds.

> **Warning:** This attribute is not supported by all browsers. Notably, Internet Explorer does not respect it.

Parameters **max_age** (*int*) – The lifetime of the cookie, in seconds.

max_age
int – The lifetime of the cookie, in seconds.

class krait.cookie.**CookiePathAttribute**(*path*)
Bases: *krait.cookie.CookieAttribute*

Sets the *Path* attribute on the cookie. This restricts the cookie only to one URL and its descendants.

Parameters **path** (*str*) – The URL to which to restrict the cookie.

path
str – The URL to which to restrict the cookie.

class krait.cookie.**CookieDomainAttribute**(*domain*)
Bases: *krait.cookie.CookieAttribute*

Sets the *Domain* attribute on the cookie. This restricts the domain on which the cookie can be sent by the client.

Parameters **domain** (*str*) – The domain on which the cookie is restricted.

domain
str – The domain on which the cookie is restricted.

class krait.cookie.**CookieHttpOnlyAttribute**
Bases: *krait.cookie.CookieAttribute*

Sets the *HttpOnly* attribute on the cookie. This causes the cookie to be inaccessible from Javascript.

class krait.cookie.**CookieSecureAttribute**
Bases: *krait.cookie.CookieAttribute*

Sets the *Secure* attribute on the cookie. This causes the cookie to only be sent over HTTPS (not yet supported by Krait).

krait.cookie.**get_cookies**()
Get all the cookies sent by the client.

Returns list of *Cookie*

krait.cookie.**get_cookie**(*name*, *default=None*)
Get the value of a single cookie, by name.

Parameters

- **name** (*str*) – The name of the cookie to be returned.
- **default** (*str, optional*) – The value to be used if the cookie cannot be found.

Returns The value of the cookie, or the second argument, if it doesn't exist.

krait.cookie.**get_response_cookies**()
Get cookies already set with `cookie.setCookie()` or direct header manipulation

Returns the response cookies already set.

Return type list of `cookie.Cookie`

krait.cookie.**set_cookie**(*cookie*)
Set a new (or updated) cooke.

Parameters **cookie** (`cookie.Cookie`) – the cookie item.

krait.websockets module

This module interfaces Krait's WebSocket support with your Python backend. WebSocket controllers are implemented as two threads, one running the networking in C++ (the main thread), and the other running the behaviour in Python (the handler thread). Your backend implements the second thread, by subclassing *WebsocketsCtrlBase* and overriding the relevant methods.

The WebSocket protocol requires three components:

1. A page contains JavaScript code that requests a WebSocket connection to a specific URL, by requesting a WebSocket upgrade (change of protocols) through an otherwise normal GET request. This doesn't have to be Javascript running in a browser, but usually is.
2. A server-side script to handle a GET request on that URL and accept or deny the WebSocket upgrade
3. A WebSocket controller that runs in a separate thread, handles incoming messages and sends messages itself.

See the websockets tutorial (TODO) for a more detailed explanation.

Usage

1. Create a page with the relevant JavaScript that makes a WebSocket upgrade request to a separate URL, then communicates with the server on the new channel.
2. Create a WebSockets controller by subclassing *WebsocketsCtrlBase* and overriding, at least, *WebsocketsCtrlBase.on_thread_start*. This method should contain a loop that runs while *WebsocketsCtrlBase.should_stop* on `self` is false.
3. Create the route target Python script at the URL requested by the JavaScript client. Inspect *krait.websockets.request*, and if it is a valid WebSocket request (not `None` and `protocols` contains the expected subprotocol), set *krait.websockets.response* to a new *WebsocketsResponse* with a new instance of your controller and, optionally, the subprotocol that you chose.

Reference

class `krait.websockets.`**WebsocketsRequest**(*http_request*)
Bases: `object`

Represents a Websockets request. Contains additional information extracted from a `Upgrade: websocket` request.

Parameters **http_request** (*krait.Request*) – The HTTP Upgrade request.

protocols
list of str, optional – The list of protocols options, or None if no options are specified.

static extract_protocols(*http_request*)
Extract the Websocket protocol options from an HTTP request.

Parameters **http_request** (*krait.Request*) – The HTTP Upgrade request.

Returns The list of protocol options specified by the request, or None if no options are specified.

Return type list of str

class `krait.websockets.`**WebsocketsResponse**(*controller*, *protocol=None*)
Bases: `object`

Represents a Websockets response. This class is used to tell Krait what protocol has been chosen and what controller will handle the Websockets connection.

Parameters

- **controller** (*WebsocketsCtrlBase*) – The controller to be used to serve this Websockets connection.
- **protocol** (*str, optional*) – The chosen subprotocol, or None for no protocol.

controller
WebsocketsCtrlBase – The controller to be used to serve this Websockets connection.

protocol
str, optional – The chosen subprotocol, or None for no protocol.

`krait.websockets.`**request = None**
WebsocketsRequest – The additional Websockets information sent with the HTTP Upgrade request by the client. If this is None, this is not a HTTP `Upgrade: websocket` request.

krait.websockets.**response = None**
WebsocketsResponse – The response set by the backend, containing information to be sent back to the client, and the handler for the request. If this is None, the Websockets upgrade was denied. In this case, *krait.response* SHOULD be a 400 Bad Request or similar.

class krait.websockets.**WebsocketsCtrlBase**(*use_in_message_queue=True*)
Bases: object

Base class responsible for handling Websockets communication. Subclass it to add behavior. Passed as an argument to *WebsocketsResponse* and called by Krait to communicate with the Websockets client. Do not access its (private) attributes directly, use the provided methods. These ensure thread safety, and a stable API.

Parameters **use_in_message_queue** (*bool*) – True to use a message queue for incoming messages. Otherwise, the backend would handle these messages on an event model, by overriding *on_in_message*.

on_start()
Called by Krait when the controller is ready to start. This is running on the main thread. Do not perform expensive initialization here.

on_thread_start()
The handler thread's target. Started by the controller's on_start() method. Override this to add behavior to your controller.

on_in_message(*message*)
Called by Krait (indirectly) when a new message arrives from the client. By default adds the message to the message queue. Override this if the controller is not configured to use messages queues.

Parameters **message** (*str*) – The new message.

on_in_message_internal(*message*)
Called by Krait (directly) when a new message arrives from the client. This only acquires the lock, then calls *on_in_message*.

Parameters **message** (*str*) – The new message;

pop_in_message()
Return the first message in the input queue, popping it, if it exists. Return None otherwise. Call this to get messages from the client. Do **not** call this if the controller doesn't use input message queues.

Returns The value of the first message, or None if there are no messages.

Return type str

push_out_message(*message*)
Send a message. This adds it to the queue; Krait will watch the message queue and send it.

Parameters **message** (*str*) – The message to send.

pop_out_message()
Return the first message in the output queue, and remove it, if it exists. Return None otherwise. Called by Krait to check on messages to send.

Returns The value of the first message, or None if there are no messages.

Return type str

on_stop()
Set a flag that tells the controller thread to shut down. Called by Krait when the WebSockets connection is closing.

should_stop()
Return True if the shutdown event has been set, or False otherwise. Call this periodically from the controller thread to check if you should shut down.

wait_stopped(*timeout=None*)
Join the controller thread, with an optional timeout. Called by Krait until the thread has shut down.

Returns True if the thread has shut down, False otherwise.

Return type bool

CHAPTER 2

What is Krait?

Krait is an HTTP server with Python server-side scripting. Its main purpose is to be as simple and elegant to use as possible, while having no major drawbacks (no important missing features, no performance, stability or security issues)

It is written in C++, but you'll be coding the backend only in Python. Its Python API is built with simplicity and elegance in mind. Right now Krait only runs on Linux, but it also runs perfectly under Windows 10, in the Linux Subsystem.

CHAPTER 3

What can Krait offer?

Krait has most of the features that web apps need:

- Serving static files
- Routing requests based on URL and HTTP method
- Running Python scripts in response to HTTP requests
- Full control of responses & cookies from Python
- MVC framework with fully-featured templating language (also simple, and elegant!)
- Near-complete compatibility with HTTP/1.1
- Websockets support
- Open-source, under MIT

CHAPTER 4

What is still on the way?

The server is still in development, so a few features are still on the way. Before v1.0, the following features are planned to be complete:

- HTTPS support
- Easy to understand and pinpoint error messages
- Comprehensive documentation, tutorials, and examples
- A performance and throughput upgrade
- UTF-8 compatibility
- More flexible routing
- Hopefully, a Windows port
- Becoming even more elegant, and simple.

CHAPTER 5

Python Documentation

Krait has its own Python package, appropriately named `krait`. Access its documentation *here*.

CHAPTER 6

Indices and tables

- genindex
- modindex

Python Module Index

k

Index

O

P

Q

R

S

T

U

V

W