HttpLayer Tutorial

Yue Wang, 30/6/2023

This Node implements a simple HTTP request layer for sending HTTP requests and processing the results of the requests. It uses the HTTPRequest class for HTTP communication and passes the request results to other modules for processing through a signal connection mechanism.

(**<u>WARNING</u>**: The code is still under refinement and will be adjusted and improved at any time based on subsequent development.)

1. Brief description of the functions

(1) **Signal http_completed**:

```
7
8       signal http_completed(res, response_code, headers, route)
9
```

Triggered after an HTTP request completes, passing the request result, response code, response headers and routing information. Used to pass the result returned by the backend to the specific business Node.

(2) **_on_HTTPRequest_request_completed** function:

```
10      # Callback function after completion of HTTP request
11      func _on_HTTPRequest_request_completed(result, response_code, headers, body, route, httpObject, redirectTo = null):
12              # Converts the response body to a JSON object and stores the result
13              var json = JSON.new()
14              json.parse(body.get_string_from_utf8())
15              var res = json.get_data()
16              print(res)
17
18              emit_signal("http_completed", res, response_code, headers, route)
19
20              if res == null:
21                      print("API is not running")
22                      return
23
24              if res.has("status"):
25
26                      match res.status:
27                              "error":
28                                      var msg = res.message
29
30                                      if TYPE_DICTIONARY == typeof(msg):
31                                              msg = ""
32                                              for k in res.message.keys():
33                                                      msg = str(msg, "\n - ", res.message[k][0])
```

<u>• Callback function after the HTTP request is completed.</u>

- Converts the response body to a JSON object and stores the result.
- Processes and operates accordingly to the request result.

• Emits the http_completed signal, passing the request result, response code, response headers and routing information.

• If redirection to another scenario is required, a scenario switch is performed.

(3) **_destroyHttpObject** function:

```
    # Used to destroy the HTTP request object, ensuring that the memory and resources used by the request are freed.
    func _destroyHttpObject(_object):
            if weakref(_object).get_ref():
                    _object.queue_free()
```

Destroys the HTTP request object, ensuring that memory and resources used by the request are freed.

(4) **_apiCore** function:

```
# The core operations used to make HTTP requests:
# _apiCore(Endpoint of the request (router), Data to be sent (request body), Whether authorisation is required, Methods of request (GET, POST, PUT, DELETE, etc.), Rout
func _apiCore(_endpoint, _data, _authorize = false, _method="GET", _route = "", _redirectTo = null):
        # Create a new instance of HTTPRequest and connect it to the signal request_completed to trigger the specified callback function _on_HTTPRequest_request_comple
        var http = HTTPRequest.new()
        add_child(http)
        print("success api1")
        http.use_threads = use_threads
        http.request_completed.connect(_on_HTTPRequest_request_completed.bind(_route, http, _redirectTo))

        # Create a list containing request headers
        var headers = [
                "Content-Type: application/json",
                "Accept: application/json",
                ]

        # If _authorize is true, add the "Authorization" header to the headers list and use Game.user_token for authorization
        if _authorize:
                headers.append(str("Authorization: Bearer ", GameManager.user_token))

        print("success api2")
```

• Used to perform the core operations of an HTTP request.

• Creates a new instance of HTTPRequest and connects it to the request_completed signal to trigger the _on_HTTPRequest_request_completed callback function when the request completes.

• Create a list of request headers and add authorization headers as needed.

• Load the progress scenario (Loader).

• Send an HTTP request using HTTPRequest and pass the request URL, request header, request method and request body.

• Handle request errors.

• Scenario switching if redirection is required.

(5) **_login function** (and other specific interface functions):

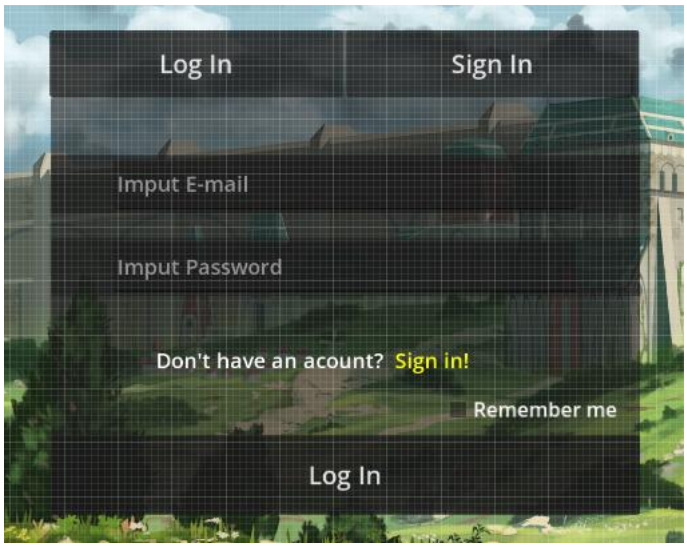
```
    # Call the _apiCore function to send the corresponding API request
    func _login(_credentials, _redirectTo):
            _apiCore("auth/login", _credentials, false, "POST", "login", _redirectTo)
```

Call the _apiCore function to send the corresponding API request .

2. How to use

Example of using the "Login" scenario:

At login, when the user account and password are entered, the 'Login' button will eventually be clicked on:



What we need to handle at this point is to pass the account and password information to the backend, so that the backend can verify that the account and password match, etc., and return the results to the frontend.

Therefore, the front-end needs to pass to the back-end is the currently entered email and password.

How to pass info to the back end?

Add the corresponding business interface in the HttpLayer, like **_login(_credentials, _redirectTo):** (_credentials: info passed from the front end; _redirectTo: pages redirected after processing)

```
100     # Call the _apiCore function to send the corresponding API request
101     func _login(_credentials, _redirectTo):
102             _apiCore("auth/login", _credentials, false, "POST", "login", _redirectTo)
103
104     func _register(_credentials, _redirectTo):
105             _apiCore("register", _credentials, false, "POST", "register", _redirectTo)
```

And pass data to the method in the Node of the specific business process:

```
func _on_login_button_pressed():
        HttpLayer._login({
                                "email": get_node("log_in/email_input_panel/LineEdit").text,
                                "password": get_node("log_in/pw_input_panel/LineEdit").text
                        }, "res://main_scene.tscn")
```

The **_apiCore** function will be called for each interface. In the _apiCore function, we initiate an HTTPRequest and complete the communication with the backend.

NOTE: _endpoint, _method and _route in _apiCore must be filled in with specifications and consistent with the backend (a separate api doc will be written afterwards to ensure consistency between the front and backend).

How to handle the info returned from the back-end?

The _on_HTTPRequest_request_completed method will handle the result after the HTTPRequest is finished. It converts the returned response body into a JSON object and stores it as result:

```
# Callback function after completion of HTTP request
func _on_HTTPRequest_request_completed(result, response_code, headers, body, route, httpObject, redirectTo = null):
        # Converts the response body to a JSON object and stores the result
        var json = JSON.new()
        json.parse(body.get_string_from_utf8())
        var res = json.get_data()
        print(res)

        emit_signal("http_completed", res, response_code, headers, route)
```

Then we can process it differently depending on the result (e.g. 'error'/'null'/'success'):

```
if res == null:
        print("API is not running")
        return

if res.has("status"):

        match res.status:
                "error":
                        var msg = res.message

                        if TYPE_DICTIONARY == typeof(msg):
                                msg = ""
                                for k in res.message.keys():
                                        msg = str(msg, "\n - ", res.message[k][0])

                        print(msg)
                        _destroyHttpObject(httpObject)
                        return

                "success":
```

It is also possible to pass the result to the specific business Node via the http_completed signal and process the result in the specific business: