MoonPalace - Moonshot AI Dark Side of the Moon Kimi API debugging tool

MoonPalace is an API debugging tool provided by Moonshot AI Dark Side of the Moon. It has the following features:

- All platform support:
  - Mac
  - Windows
  - Linux;
- Easy to use. After startup, `base_url` replace with `http://localhost:9988` to start debugging.
- Capture the entire request, including the "accident scene" in case of network errors;
- Through `request_id`, `chatcmpl_id` quick retrieval, and viewing of request information;
- One-click export of BadCase structured reporting data helped Kimi improve model capabilities;

We recommend using MoonPalace as your API "supplier" during the code writing and debugging stages so that you can quickly discover and locate various problems in the API call and code writing process. For various unexpected outputs of the Kimi big model, you can also export the request details through MoonPalace and submit them to Moonshot AI to improve the Kimi big model.

Installation

Install using `go` command

If you have already installed `go` the toolchain, you can execute the following command to install MoonPalace:

```
$ go install github.com/MoonshotAI/moonpalace@latest
```

`$GOPATH/bin/` The above command will install the compiled binary files in your directory. Run `moonpalace` the command to check whether it is successfully installed:

```
$ moonpalace
MoonPalace is a command-line tool for debugging the Moonshot AI HTTP API.

Usage:
  moonpalace [command]

Available Commands:
  cleanup     Cleanup Moonshot AI requests.
  completion  Generate the autocompletion script for the specified shell
  export      export a Moonshot AI request.
  help        Help about any command
  inspect     Inspect the specific content of a Moonshot AI request.
  list        Query Moonshot AI requests based on conditions.
  start       Start the MoonPalace proxy server.

Flags:
  -h, --help      help for moonpalace
  -v, --version   version for moonpalace

Use "moonpalace [command] --help" for more information about a command.
```

If you still cannot retrieve `moonpalace` the binaries, try `$GOPATH/bin/` adding the directory to your `$PATH` PATH environment variables.

Download from Releases page

You can download it from ReleasesDownload the compiled binary (executable) file from the page:

- moonpalace-linux
- moonpalace-macos-amd64 => for Intel version of Mac
- moonpalace-macos-arm64 => corresponds to the Apple Silicon version of Mac
- moonpalace-windows.exe

Please download the corresponding binary (executable) file according to your platform, place the binary (executable) file in `$PATH` the directory contained in the environment variable, rename it `moonpalace`, and finally grant it executable permission.

How to use

Start the service

Start the MoonPalace proxy server using the following command:

```
$ moonpalace start --port <PORT>
```

MoonPalace will start a local HTTP server. `--port` The parameter specifies the local port that MoonPalace listens on. The default value is `9988`. When MoonPalace starts successfully, it will output:

```
[MoonPalace] 2024/07/29 17:00:29 MoonPalace Starts {'=>'} change base_url to "http://127.0.0.1:
```

As required, we will `base_url` replace with the displayed address. If you use the default port, please set it `base_url=http://127.0.0.1:9988/v1`. If you use a custom port, please `base_url` replace with the displayed address.

Additionally, if you want to always use a debugger when debugging , you can set a default for `api_key` MoonPalace using the parameter when starting MoonPalace , so that you don't have to set it manually when requesting . MoonPalace will help you add the one you set when requesting the Kimi API . `--key` `api_key` `api_key` `--key` `api_key`

If you set it up correctly `base_url` and successfully call the Kimi API, MoonPalace will output the following information:

```
$ moonpalace start --port <PORT>
[MoonPalace] 2024/07/29 17:00:29 MoonPalace Starts {'=>'} change base_url to "http://127.0.0.1:
[MoonPalace] 2024/07/29 21:30:53 POST   /v1/chat/completions 200 OK
[MoonPalace] 2024/07/29 21:30:53   - Request Headers:
[MoonPalace] 2024/07/29 21:30:53     - Content-Type:   application/json
[MoonPalace] 2024/07/29 21:30:53   - Response Headers:
[MoonPalace] 2024/07/29 21:30:53     - Content-Type:   application/json
[MoonPalace] 2024/07/29 21:30:53     - Msh-Request-Id: c34f3421-4dae-11ef-b237-9620e33511ee
[MoonPalace] 2024/07/29 21:30:53     - Server-Timing:  7134
[MoonPalace] 2024/07/29 21:30:53     - Msh-Uid:        cn0psmmcp7fclnphkcpg
[MoonPalace] 2024/07/29 21:30:53     - Msh-Gid:        enterprise-tier-5
[MoonPalace] 2024/07/29 21:30:53   - Response:
[MoonPalace] 2024/07/29 21:30:53     - id:               cmpl-12be8428ebe74a9e8466a37bee7a9b11
[MoonPalace] 2024/07/29 21:30:53     - prompt_tokens:    1449
[MoonPalace] 2024/07/29 21:30:53     - completion_tokens: 158
[MoonPalace] 2024/07/29 21:30:53     - total_tokens:     1607
[MoonPalace] 2024/07/29 21:30:53   New Row Inserted: last_insert_id=15
```

MoonPalace will output the request details in the form of a log on the command line (if you want to store the log content persistently, you can `stderr` redirect it to a file).

Note: In the log, `Msh-Request-Id` the value of the field in the Response Headers corresponds to the value of the parameter in **the retrieval request** and **export request described** `--requestid` below , and in the Response `id` corresponds to `--chatcmpl` the value of the parameter and `last_insert_id` corresponds to `--id` the value of the parameter.

Content truncation detection

MoonPalace can detect whether the output content of the current Kimi large model is truncated or incomplete (this function is enabled by default). When MoonPalace detects that the output content is truncated or incomplete, it will output in the log:

```
[MoonPalace] 2024/08/05 19:06:19   it seems that your max_tokens value is too small, please set
```

If you are currently using non-streaming output mode (stream=False), MoonPalace will give you suggested `max_tokens` values.

Enable duplicate content output detection

MoonPalace provides a detection function for repeated content output of Kimi Big Model. Repeated content output means: **Kimi Big Model will repeatedly output a specific word, sentence, and blank character, and `max_tokens` will not stop until the limit is reached. **When using `moonshot-v1-128k` a model with higher fees such as , this repeated output will result in additional Tokens cost consumption, so MoonPalace provides `--detect-repeat` an option to enable repeated content output detection, as shown below:

```
$ moonpalace start --port <PORT> --detect-repeat --repeat-threshold 0.3 --repeat-min-length 20
```

After enabling `--detect-repeat` the option, MoonPalace will interrupt the output of Kimi Big Model when it detects duplicate content output of Kimi Big Model and output in the log:

```
[MoonPalace] 2024/08/05 18:20:37   it appears that there is an issue with content repeating in
```

Note: After enabling `--detect-repeat`, MoonPalace will interrupt the output of Kimi large model only in streaming output (stream=True), which is not applicable in non-streaming output.

You can adjust MoonPalace's blocking behavior using the `--repeat-threshold` / parameters: `--repeat-min-length`

- `--repeat-threshold` The parameter is used to set MoonPalace's tolerance for duplicate content. A higher threshold means a lower tolerance, and duplicate content will be blocked faster. 0 <= threshold <= 1
- `--repeat-min-length` The parameter is used to set the number of characters that MoonPalace starts to detect when duplicate content is output. For example, --repeat-min-length=100 means that duplicate detection is enabled when the number of UTF-8 characters output

exceeds 100, and duplicate content output detection is disabled when the number of characters output is less than 100.

Enable forced streaming output

MoonPalace provides `--force-stream` an option to force all `/v1/chat/completions` requests to use streaming output mode:

```
$ moonpalace start --port <PORT> --force-stream
```

MoonPalace sets the field in the request parameters `stream` to `True`, and when getting a response, it automatically `stream` determines the format of the response based on whether the caller has set :

- If the caller has set it `stream=True`, it will be returned in the format of streaming output, and MoonPalace will not do any special processing on the response;
- If the caller does not set `stream` the value of , or sets it `stream=False`, MoonPalace will concatenate the data blocks into a complete completion structure and return it to the caller after receiving all streaming data blocks;

For the caller (developer), enabling `--force-stream` the option will not affect the Kimi API response content you get. You can still use the original code logic to debug and run your program. In other words, **enabling `--force-stream` the option will not change or destroy anything** . You can safely enable this option.

Why provide such an option?

> We preliminarily speculate that the reason for common network connection errors, timeouts and other problems (Connection Error/Timeout) is that when using non-streaming mode for requests (stream=False), the gateway or proxy server in the middle layer sets read_header_timeout or read_timeout, which causes the gateway or proxy server in the middle layer to disconnect while the Kimi API server is still assembling the response (because no response, not even the response header, is received), resulting in a Connection Error/Timeout.

> We tried adding `--force-stream` a parameter to MoonPalace. When `moonpalace start --force-stream` started with , MoonPalace will convert all non-streaming requests (stream=False or stream not set) into streaming requests, and after receiving all data blocks, assemble them into a complete completion response structure and return it to the caller.
>
> For the caller, the non-streaming API can still be used in the original way, but after the conversion by MoonPalace, the Connection Error/Timeout situation can be reduced to a certain extent, because at this time MoonPalace has established a connection with the Kimi API server and started to receive streaming data blocks.

Retrieval Request

After MoonPalace is started, all requests that pass through MoonPalace will be recorded in a sqlite database located at `$HOME/.moonpalace/moonpalace.sqlite`. You can directly connect to the MoonPalace database to query the specific content of the request, or you can query the request through the MoonPalace command line tool:

```
$ moonpalace list
+----+--------+-----------------------------------------+----------------------------------
| id | status | chatcmpl                                | request_id
+----+--------+-----------------------------------------+----------------------------------
| 15 | 200    | cmpl-12be8428ebe74a9e8466a37bee7a9b11    | c34f3421-4dae-11ef-b237-9620e33511e
| 14 | 200    | cmpl-1bf43a688a2b48eda80042583ff6fe7f    | c13280e0-4dae-11ef-9c01-debcfc72949
| 13 | 200    | chatcmpl-2e1aa823e2c94ebdad66450a0e6df088 | c07c118e-4dae-11ef-b423-62db244b927
| 12 | 200    | cmpl-e7f984b5f80149c3adae46096a6f15c2    | 50d5686c-4d98-11ef-ba65-3613954e258
| 11 | 200    | chatcmpl-08f7d482b8434a869b001821cf0ee0d9 | 4c20f0a4-4d98-11ef-999a-928b67d58fa
| 10 | 200    | chatcmpl-6f3cf14db8e044c6bfd19689f6f66eb4 | 49f30295-4d98-11ef-95d0-7a2774525b8
| 9  | 200    | cmpl-2a70a8c9c40e4bcc9564a5296a520431    | 7bd58976-4d8a-11ef-999a-928b67d58fa
| 8  | 200    | chatcmpl-59887f868fc247a9a8da13cfbb15d04f | ceb375ea-4d7d-11ef-bd64-3aeb95b9dfa
| 7  | 200    | cmpl-36e5e21b1f544a80bf9ce3f8fc1fce57    | cd7f48d6-4d7d-11ef-999a-928b67d58fa
| 6  | 200    | cmpl-737d27673327465fb4827e3797abb1b3    | cc6613ac-4d7d-11ef-95d0-7a2774525b8
+----+--------+-----------------------------------------+----------------------------------
```

Use `list` the command to query the most recently generated request content. The default displayed fields are `id` // `chatcmpl` for easy retrieval and // information `request_id` for viewing

request status . If you want to view a specific request, you can use the command to retrieve the corresponding request: `status` `server_timing` `requested_at` `inspect`

```
# 以下三条命令会检索出相同的请求信息
$ moonpalace inspect --id 13
$ moonpalace inspect --chatcmpl chatcmpl-2e1aa823e2c94ebdad66450a0e6df088
$ moonpalace inspect --requestid c07c118e-4dae-11ef-b423-62db244b9277
+------------------------------------------------------------+
| metadata                                                   |
+------------------------------------------------------------+
| {                                                          |
|     "chatcmpl": "chatcmpl-2e1aa823e2c94ebdad66450a0e6df088", |
|     "content_type": "application/json",                    |
|     "group_id": "enterprise-tier-5",                       |
|     "moonpalace_id": "13",                                 |
|     "request_id": "c07c118e-4dae-11ef-b423-62db244b9277",  |
|     "requested_at": "2024-07-29 21:30:43",                 |
|     "server_timing": "1033",                               |
|     "status": "200 OK",                                    |
|     "user_id": "cn0psmmcp7fclnphkcpg"                      |
| }                                                          |
+------------------------------------------------------------+
```

By default, `inspect` the command will not print the request and response body information. If you want to print the body, you can use the following command:

```
$ moonpalace inspect --chatcmpl chatcmpl-2e1aa823e2c94ebdad66450a0e6df088 --print request_body,
# 由于 body 信息过于冗长，这里不再完整展示 body 详细内容
+--------------------------------------------------+-----------------------------------------
| request_body                                     | response_body
+--------------------------------------------------+-----------------------------------------
| ...                                              | ...
+--------------------------------------------------+-----------------------------------------
```

Export Request

When you think a request does not meet your expectations, or want to report a request to Moonshot AI (good case or bad case, we welcome both), you can use `export` the command to

export a specific request:

```
# id/chatcmpl/requestid 选项只需要任选其一即可检索出对应的请求
$ moonpalace export \
    --id 13 \
    --chatcmpl chatcmpl-2e1aa823e2c94ebdad66450a0e6df088 \
    --requestid c07c118e-4dae-11ef-b423-62db244b9277 \
    --good/--bad \
    --tag "code" --tag "python" \
    --directory $HOME/Downloads/
```

Among them, `id` // is used in the same way as the command to retrieve a specific request, and `chatcmpl` / is used to mark the current request as a Good Case or a Bad Case, and is used to label the current request accordingly. For example, in the above example, we assume that the current request content is related to the programming language Python, so we add two , namely and , to specify the path of the directory where the exported file is stored. `requestid` `inspect` `--good` `--bad` `--tag` `tag` `code` `python` `--directory`

The contents of the successfully exported file are:

```
$ cat $HOME/Downloads/chatcmpl-2e1aa823e2c94ebdad66450a0e6df088.json
{
    "metadata":
    {
        "chatcmpl": "chatcmpl-2e1aa823e2c94ebdad66450a0e6df088",
        "content_type": "application/json",
        "group_id": "enterprise-tier-5",
        "moonpalace_id": "13",
        "request_id": "c07c118e-4dae-11ef-b423-62db244b9277",
        "requested_at": "2024-07-29 21:30:43",
        "server_timing": "1033",
        "status": "200 OK",
        "user_id": "cn0psmmcp7fclnphkcpg"
    },
    "request":
    {
        "url": "https://api.moonshot.cn/v1/chat/completions",
        "header": "Accept: application/json\r\nAccept-Encoding: gzip\r\nConnection: keep-alive\
        "body":
        {}
    },
    "response":
```

```
        {
            "status": "200 OK",
            "header": "Content-Encoding: gzip\r\nContent-Type: application/json; charset=utf-8\r\nD
            "body":
            {}
        },
        "category": "goodcase",
        "tags":
        [
            "code",
            "python"
        ]
    }
```


We recommend developers to use [Github Issues](#)Submit a Good Case or Bad Case . If you do not want to disclose your request information, you can also submit your case to us via corporate WeChat, email, etc.

You can send the exported file to the following email addresses:

api-feedback@moonshot.cn

Last updated on April 2, 2025