Other languages: 繁體中文

Configuration File

You can control various behaviors of the Http server by configuring various parameters through multiple interfaces of the DrogonAppFramework instance. However, using a configuration file is a better way for the following reasons:

- Using a configuration file instead of source code can determine the behavior of the application at runtime rather than at compile time, which is undoubtedly a more convenient and flexible way;
- Using a configuration file can make the main file more concise;

Based on these additional benefits, it is recommended that application developers use configuration files to configure various parameters of the application.

The configuration file can be loaded very simply by calling the loadConfigFile() interface before calling the run() interface. The parameter of the loadConfigFile() method is the file name of the configuration file, for example:

```
int main()
{
    drogon::app().loadConfigFile("config.json");
    drogon::app().run();
}
```

The above program loads the configuration file `config.json` and then runs the application. The specific listening port, log output, database configuration and so on can be configured by the configuration file. In fact, this program can basically be the entire code of the main file of the web application.

Configuration File Details

An example of a configuration file is at the top level of the source directory, config.example.json. If you use the drogon_ctl create project command to create a project, you can also find the file config.json with the same content in the project directory. So, you basically don't need to create a new configuration file but make some changes to this file to complete the configuration of the web application.

The file is in `JSON` format and supports comments. You can comment out the unneeded configuration items with the c++ comment symbols /**/ and //.

After commenting out a configuration option, the framework initializes it with default values. The default value for each option can be found in the comments in the configuration file.

Supportted Format

- json
- yaml, should install yaml-cpp library to provide the yaml file parser.

- SSL

  The ssl option is to configure SSL files of the https service as follows:

```
  "ssl": {
    "cert": "../../trantor/trantor/tests/server.pem",
    "key": "../../trantor/trantor/tests/server.pem",
    "conf":[
      ["Options", "Compression"],
      ["min_protocol", "TLSv1.2"]
    ]
  }
```

  Where `cert` is the path of the certificate file and `key` is the path of the private key file. If a file contains both a certificate and a private key, the two paths can be made the same. The file is in PEM encoding format.

  `conf` are optional SSL options that are directly passed into SSL_CONF_cmd to allow low level configuration of the encryption. The options must be one or two element arrays.

- listeners

  As the name implies, the `listeners` option is to configure listeners for the web application. It is a JSON array type. Each JSON object represents a listener. The specific configuration is as follows:

```
"listeners": [
    {
      "address": "0.0.0.0",
      "port": 80,
      "https": false
    },
    {
      "address": "0.0.0.0",
      "port": 443,
      "https": true,
      "cert": "",
      "key": ""
    }
  ]
```

  Among them:

  - `address`: With the string type indicates the IP address to be listened to. If this option is not available, the default value "0.0.0.0" is used.
  - `port`: `port`: An integer type option indicating the port to be listened to. It must be a valid port number. There is no default value.
  - `https`: Boolean type, indicating whether to use https, the default value is false, which means using http.

  - `cert` and `key`: The string type, is valid when `https` is true, indicating the certificate and private key of https. The default value is an empty string, indicating the certificate and private key file configured by the global option `ssl`;

- db_clients

  This option is used to configure the database client. It is a JSON array type. Each JSON object represents a separate database client. The specific configuration is as follows:

```
"db_clients":[
  {
    "name":"",
    "rdbms": "postgresql",
    "host": "127.0.0.1",
    "port": 5432,
    "dbname": "test",
    "user": "",
    "passwd": "",
    "is_fast": false,
    "connection_number": 1,
    "filename": ""
  }
]
```

  Among them:

  - `name`: string type, client name, the default value is "default", name is the application developer to get the database client's markup from the framework, if there are multiple clients, the name field must be different;
  - `rdbms`：A string indicating the type of database server. Currently supports "postgresql","mysql" and "sqlite3", which is not case sensitive;
  - `host`：String, database server address, `localhost` is the default value;
  - `port`：An integer representing the port number of the database server;
  - `dbname`：String, database name;
  - `user`：String, user name;
  - `passwd`：String, password;
  - `is_fast`：bool，false by default, indicate if the client is a FastDbClient
  - `connection_number`：A integer indicating the number of connections to the database server, at least 1, the default value is also 1, affecting the concurrent performance of data read and write; If the 'is_fast' is true, the number is the number of connections per event loop, otherwise it is the total number of all connections.
  - `filename`: The filename of sqlite3 database;

- threads_num

  Suboption belonging to the `app` option, an integer, the default value is 1, indicating the number of IO threads, which has a clear impact on network concurrency. This number is not as big as possible. Users who understand the non-blocking I/O principle should know that this value should be the same as the

number of processors that he expects network IO to occupy. If the value is set to 0, the number of IO threads will be the number of all hardware cores.

```
"threads_num": 16,
```

The above example shows that network IO uses 16 threads and can run up to 16 CPU cores under high load conditions.

- Session

  Session-related options are also children of the `app` option, controlling whether session is used and the session timeout. Such as:

  ```
  "enable_session": true,
  "session_timeout": 1200,
  ```

  Among them:

  - `enable_session`：A Boolean value indicating whether to use a session. The default is false. If the client does not support cookies, set it to false because the framework will create a new session for each request without a session cookie, which will result in completely unnecessary resource and performance loss;
  - `session_timeout`：An integer value indicating the timeout period of the session, in seconds. The default value is 0, indicating permanent validity. Only works if enable_session is true.

- document_root

  The suboption of the `app` option, a string, indicates the document path corresponding to the Http root directory, and is the root path of the static file download. The default value is "./", which indicates the current path of the program running. such as:

  ```
  "document_root": "./",
  ```

- upload_path

  The child of the `app` option, a string, represents the default path for uploading files. The default value is "uploads". If the value is not starting with `/`, `./` or `../`, and this value is not .or..`, then this path is the relative path of the previous document_root entry, otherwise it is an absolute path or a relative path to the current directory. Such as:

  ```
  "upload_path":"uploads",
  ```

- client_max_body_size

  A child of the `app` object, a string, represents the overall maximum size of the body of a request. You can use the suffixes `k`, `m`, `g`, to specify kilobyte, megabyte or gigabye (byte * 1024), on 64bit build you can also use `t` for terabytes. The suffixes can be both lower case or upper case.

```
"client_max_body_size": "10M",
```

- client_max_memory_body_size

  A child of the `app` object, a string, represents the maximum buffer size to store a request body before caching to a file. You can use the suffixes `k`, `m`, `g`, to specify kilobyte, megabyte or gigabye (byte * 1024), on 64bit build you can also use `t` for terabytes. The suffixes can be both lower case or upper case.

```
"client_max_memory_body_size": "50K"
```

- file_types

  The suboption of the app option, an array of strings, with default values as follows, indicates the static file download type supported by the framework. If the requested static file extension is outside of these types, the framework will return a 404 error.

```
"file_types": [
      "gif",
      "png",
      "jpg",
      "js",
      "css",
      "html",
      "ico",
      "swf",
      "xap",
      "apk",
      "cur",
      "xml"
    ],
```

- mime

  The suboption of the app option, a dictionary of strings to strings or an array of strings. Declares how file extensions are mapped to new MIME types (i.e. for those not recognized by default) when sending static files. Note that this options merely registers the MIME. The framework still sends a 404 if the extension is not in `file_types` described above.

```
"mime" : {
  "text/markdown": "md",
  "text/gemini": ["gmi", "gemini"]
}
```

- Connection number control

  The children of the `app` option have two options, as follows:

  ```
  "max_connections": 100000,
  "max_connections_per_ip": 0,
  ```

  Among them:

  - `max_connections`：Integer, the default value is 100000, which means the maximum number of simultaneous concurrent connections; when the number of connections maintained by the server reaches this number, the new TCP connection request will be rejected directly.
  - `max_connections_per_ip`：Integer, the default value is 0, which means the maximum number of connections for a single client IP, and 0 means no limit.

- Log option

  The child of the `app` item, a JSON object, controls the behavior of the log output as follows:

  ```
  "log": {
        "log_path": "./",
        "logfile_base_name": "",
        "log_size_limit": 100000000,
        "log_level": "TRACE"
      },
  ```

  Among them:

  - `log_path`：String, the default value is an empty string, indicating the path where the log file is stored. If it is an empty string, all logs are output to the standard output.
  - `logfile_base_name`：A string indicating the `basename` of the log file. The default value is an empty string which means the basename will be `drogon`.
  - `log_size_limit`：A integer, in bytes. The default value is 100000000 (100M). When the size of the log file reaches this value, the log file will be switched.
  - `log_level`：A string, the default value is "DEBUG", which indicates the lowest level of log output. The optional values are from low to high: "TRACE", "DEBUG", "INFO", "WARN", where the TRACE level is only valid when compiling in DEBUG mode.

> **Note: Drogon's file log uses a non-blocking output structure that can achieve a log output of millions of lines per second and can be used with confidence.**

- Application control

  They are also children of the `app` option and have two options, as follows:

  ```
      "run_as_daemon": false,
      "relaunch_on_error": false,
  ```

  Among them：

  - `run_as_daemon`：Boolean value, the default value is false. When it is true, the application will be a child process of the No.1 process in the form of a daemon running in the background of the system.
  - `relaunch_on_error`：Boolean value, the default value is false. When it is true, the application will relaunches itself on error.

- use_sendfile

  The suboption of `app` option, boolean, indicates whether the linux system call sendfile is used when sending the file. The default value is true. Using sendfile can improve the sending efficiency and reduce the memory usage of large files. as follows:

  ```
  "use_sendfile": true,
  ```

  > **Note: Even if this option is true, the sendfile system call will not be used, because the use of sendfile for small files is not necessarily cost-effective, and the framework will decide whether to adopt it according to its own optimization strategy.**

- use_gzip

  The `app` suboption, boolean, default value is true, indicating whether the body of the Http response uses compressed transmission. When it is true, compression is used in the following cases:

  - The client supports gzip compression;
  - The Http body is the text type;
  - The length of the body is greater than a certain value;

  The configuration example is as follows:

  ```
  "use_gzip": true,
  ```

- static_files_cache_time

  The `app` suboption, integer value, in seconds, indicates the cache time of the static file, that is, for the repeated request for the file during this time, the framework will return the response directly from

the memory without reading the file system. The default value is 5 seconds, 0 means always cache (only read the file system once, use with caution), negative value means no cache. as follows:

```
"static_files_cache_time": 5,
```

- simple_controllers_map

  The `app` suboption, an array of JSON objects, each representing a mapping from the Http path to the HttpSimpleController, this configuration is just an alternative, not necessarily configured here, see HttpSimpleController. The specific configuration is as follows:

```
"simple_controllers_map": [
      {
        "path": "/path/name",
        "controller": "controllerClassName",
        "http_methods": ["get","post"],
        "filters": ["FilterClassName"]
      }
    ],
```

  Among them：

  - `path`：String, Http path;
  - `controller`：String, the name of the HttpSimpleController;
  - `http_methods`：An array of strings representing the supported Http methods. Requests outside this list will be filtered out, returning a 405 error.
  - `filters`：String array, list of filters on the path, see Middleware and Filter;

- Idle connection timeout control

  The `app` suboption, integer value, in seconds, the default value is 60. When a connection exceeds this time without any reading and writing, the connection will be forcibly disconnected. as follows:

```
"idle_connection_timeout":60
```

- Dynamic view loading

  The sub-options of the app, which control the enabling and the path of the dynamic view, have two options, as follows:

```
"load_dynamic_views":true,
"dynamic_views_path":["./views"],
```

Among them：

- `dynamic_views_path`：Boolean value, the default value is false. When it is true, the framework searches view files in the view path and dynamically compiles them into .so files, then loads them into the application. When any view file changes, it will also cause automatic compilation and re-loading;
- `dynamic_views_path`：An array of strings, each of which represents the search path of the dynamic view. If the path value is not starting with `/`, `./` or `../`, and the value is not `.` or `..`, then This path is the relative path of the previous document_root entry, otherwise it is an absolute path or a relative path to the current directory.

See View

- Server header field

The sub-option of the app configures the server header field of all responses sent by the framework. The default value is an empty string. When this option is empty, the framework automatically generates a header field of the form `Server: drogon/version string`. It's as follows:

```
"server_header_field": ""
```

- Keepalive requests

The `keepalive_requests` option sets the maximum number of requests that can be served through one keep-alive connection. After the maximum number of requests are made, the connection is closed. The default value of 0 means no limit. It's as follows:

```
"keepalive_requests": 0
```

- Pipelining requests

The `pipelining_requests` sets the maximum number of unhandled requests that can be cached in pipelining buffer. After the maximum number of requests are made, the connection is closed. The default value of 0 means no limit. For details about pipelining, please see the `rfc2616-8.1.1.2`. It's as follows:

```
"pipelining_requests": 0
```

Next: Aspect Oriented Programming (AOP)