gossr

README | 中文文档

gossr is a server-side rendering framework (SSR) for web development, implemented using golang + V8 + Vue. Similar to Nuxt or Next, but they are implemented using Nodejs.

Advantage

- The advantages of the SSR framework itself
  - Better SEO, search engine crawlers can directly crawl fully rendered pages
  - Faster content arrival time (time-to-content), users will see the fully rendered page more quickly, so as to have a better user experience
- Golang + V8 implementation, better performance than Nodejs-based solutions
  - Using golang to implement the server framework, that can schedule multiple V8 VM instances in multiple threads.
  - In actual projects, js often has a memory leak, which is fatal to the server. This framework solves the memory leak by setting the V8 VM's lifetime and deleting the instance after the lifetime expires, thus ensuring that the server can last for a long time.
- Implemented the js environment required for SSR
  - Implemented CommonJS require specification
  - Implemented XMLHttpRequest, can execute ajax request
  - Implemented console.debug, console.log, console.info, console.warn, console.error
  - The setTimeout and setInterval methods that are not recommended by SSR are not implemented to avoid potential problems
- Support better development and debugging
  - SSR and browser support unified global variables BASE_URL, API_BASE_URL, unified production environment and development environment code
  - API_BASE_URL automatically supports delegating of api calls in the development environment, thereby ensuring that cookies can be delivered correctly
- Simple server deployment
  - Golang code is compiled into a single executable file, as long as the executable file and webpack packaged js script are deployed on the server

Build and run

1. Install V8 environment
   - MacOS
     - refer to install.v8/macos.md
   - CentOS 7
     - refer to install.v8/centos7.md
   - Not supported now in other environments
2. Compilation of golang server
   - Go to the project root directory
   - Run `make` to complete the compilation
   - Run `make test` to test v8 worker

    - Run `make clean` to clean the project
3. Packaging of js project
    1. Install node environment in advance
    2. `cd client`
    3. `npm install` installs the dependent packages of the project
    4. `npm run build-dev` package for development environment, or `npm run build-prod` package for production environment
    5. `npm run watch` package for development environment, and monitors the changes of the file, and incrementally updates the package content, that is, it can be hot updated.
4. Run
    1. After the above three steps are completed, you can run the server
    2. Run by `./gossr --config conf/goblogssr-dev.toml`, goblogssr-prod.toml in the conf directory is the configuration of the production environment
    3. Visit the URL: `http://localhost:8080/` on the browser to view the output

Configuration file

- host: ip and port which server listens to. Example: "0.0.0.0:8080", indicating that all the IPs of the server, and port 8080
- env: indicates that the server is running in the development environment or in production environment. "dev" means the development environment, "prod" means the production environment, or you can customize the environment. In js defines a global variable APP_ENV, which is the value
- v8_maxcount: indicates how many V8 VM instances to run at most
- v8_lifetime: indicates the life time of each V8 VM instance, in seconds. During the lifetime of the instance, the loaded js script will remain in memory and will not be reloaded. In the development environment, it can be set to 0 to force the instance to be recreated every time when requested.
- js_project_path: directory of js project
- static_url_path: the prefix of resource url, which generated client js, img, css and other resource
- is_api_delegate: Whether the server does api delegating. If it is true, it will delegate the browser-end ajax request prefixed by /api to the server configured by internal_api_host.
- internal_api_host: the server that delegated the request
- internal_api_ip: ssr backend ajax request will be forced to change to this ip, which will improve the performance of backend ajax request. At this time, if internal_api_host is configured, the Host of the request header will be set to this value.
- internal_api_port: the API server's port
- template_name: template for outputting html pages, the template directory is client/server_dist/template
- client_cookie: if it is not empty, the gossr server will generate a cookie with a long lifetime, which can be used as the client's identification id
- redirect_onerror: If it is not empty, if an error occurs in the js script that generates the page, the request returns a 302 page response
- ssr_ctx: The ssr framework will obtain the header value corresponding to the request according to the configured header, and pass it to the js script. The script can access the corresponding header value through context.ssrCtx. And, the backend ajax request will automatically bring these header values. Cookie header will be included by default
- template_vars: template variable definition, will map the corresponding value of the state.meta object generated by the script to the template output. Type 'js' indicates that the content of the variable is a

js script; type 'html' indicates that the content of the variable is html content, and will not escape the characters <, >, etc. Other types will escape the content and do not have to be listed in the configuration item.

List of deployment files

```
├── gossr
├── conf/
└── client/
    ├── dist/
    ├── dist_server/
    └── node_modules/
        └── vue-server-renderer/
```