Documentazione per Ryno

Senni Mattia

May 7, 2024

1 Introduzione

Ryno è un semplice server HTTP in Python, ispirato al framework Express di Node.js. Questo documento fornisce una guida dettagliata su come utilizzare e comprendere Ryno, descrivendo le sue funzionalità, il funzionamento interno e le considerazioni aggiuntive.

2 Funzionalità

Di seguito sono elencate le principali funzionalità di Ryno:

- **Middleware Based**: Ryno è middleware based, prendendo ispirazione dal framework Express di Node.js. Ciò significa che le funzioni di middleware possono essere aggiunte e utilizzate per gestire le richieste HTTP in modo modulare.
- **Servizio di file statici**: Ryno è in grado di servire file statici dal dispositivo. Il server può essere configurato per servire una specifica cartella e rendere disponibili i file in essa contenuti su una determinata route.
- **Sistema di routing interno**: Ryno ha un sistema di routing interno che consente di definire facilmente le rotte per gestire le richieste HTTP.
- **Gestione degli header e dello status della risposta**: Ryno gestisce correttamente gli header e lo status della risposta HTTP, consentendo un controllo completo sulla risposta inviata al client.
- **Supporto per la compressione gzip**: Ryno supporta la compressione gzip se supportata dal client, riducendo la dimensione dei file inviati e migliorando le prestazioni della rete.
- **Esempio di sito funzionante**: Ryno include un complesso sito web funzionante come esempio, che comprende immagini, font, video, HTML, CSS e JavaScript.
- **Supporto per le API**: Ryno supporta la creazione di API RESTful, consentendo la creazione di servizi web basati su architettura REST.

3 Funzionamento

Ryno è progettato per essere semplice da utilizzare e configurare. Di seguito sono riportati i passaggi principali per utilizzare Ryno:

1. **Installazione**: Per utilizzare Ryno, è necessario installare Python 3 sul proprio sistema. Dopodiché, è possibile scaricare e utilizzare il codice sorgente di Ryno dalla seguente repository Ryno python server su github.
2. **Configurazione**: Ryno può essere configurato modificando il file di configurazione presente in src/kernel/configuration.py, specificando la cartella contenente i file statici da servire e definendo le rotte per gestire le richieste HTTP.
3. **Avvio del server**: Una volta configurato, è possibile avviare il server eseguendo il file server.py tramite python ('python server.py'). Il server si metterà in ascolto delle richieste HTTP in arrivo e risponderà di conseguenza.
4. **Gestione delle richieste**: Ryno gestirà automaticamente le richieste HTTP in arrivo, instradandole alle funzioni di middleware appropriate e restituendo le risposte corrispondenti.
5.
6. **Middleware configurabili**: Tramite il file 'src/main' è possibile aggiungere nuovi middleware su nuove route per o servire nuovi file statici o creare nuove API, seguendo gli esempi disponibili.

4 Funzionamento interno

Analisi struttura delle cartelle e dei file di Ryno:

1. server.py: il file principale che avvia il server.
2. src/main.py: il file che contiene le funzioni di middleware e le rotte per gestire le richieste HTTP.
3. src/kernel/conf.py: il file di configurazione che definisce l'host e la porta della socket.
4. src/kernel/errors.py: funzioni utili da richiamare per restituire gli errori HTTP (in particolare sono gestiti il codice 404 e 500).
5. src/kernel/handle-request.py: prende le richieste in arrivo, scompatta l'oggetto request, carica i middleware, esegue i middleware della route specificata, incapsula seguendo il protocollo http la risposta, la invia al client e chiude la socket.

6. src/kernel/kernel.py: contiene il codice per aprire la socket e gestire tramite Thread le richieste in entrata.

7. src/kernel/middleware.py: contiene funzioni che facilitano la creazione di un middleware generico + la funzione per aggiungere un middleware che consente di gestire le risposte tramite risorse statiche e la relativa compressione tramite gzip se presente l'header apposito nella richiesta.

8. src/kernel/response.py: contiene dei metodi che facilitano il set dei parametri della risposta http (tramite un pattern che ricorda il Builder pattern).

9. static/**: contiene i file statici che il server può servire (configurato con una complessa landing page per dimostrare il funzionamento del server), data la dimensione dei suoi file è consigliata la visualizzazione del sito tramite un browser che supporti la compressione gzip.

4.1 Esempio di middleware

Creazione di un middleware per gestire le risposte custom (nell file src/main.py):

```
def main(request, response):
    addMiddleware(response, "api", testMiddlewareBody)

def testMiddlewareBody(response, server):
    set_body(response, "{\"ping\":\"pong\"}")
    add_header(response, "Content-Type", "application/json")
    return True
```

Analisi di request e response: Gli oggetti request e response contengono informazioni riguardo la richiesta e la risposta da dare. Il return True o False decreta se quello deve essere l'ultimo middleware da visualizzare oppure no

```
    response = {
        "body": "",
        "body_is_in_bytes": False,
        "status": 200,
        "status_message": "OK",
        "protocol": "HTTP/1.1",
        "headers": [], # contiene tutti gli header
        "middleware": [] # contiene tutti i middleware ordinati
    }

    request = {
        "path": "index.html", # il file richiesto
        "folder": "", # la cartella richiesta
        "method": "GET", # il metodo usato per la richiesta
        "get": {"page":1}, # i parametri get
        "headers": {}, # tutti gli header della richiesta
    }
```

5 Considerazioni aggiuntive

Ryno è progettato per essere flessibile e personalizzabile, consentendo agli sviluppatori di creare facilmente servizi web robusti e scalabili (partendo da un sito statico a delle funzionanti API).