Aviso legal

Serviços para Aplicações Web e Móveis

Logging

Fábio Marques (fabio@ua.pt)
Mestrado em Informática Aplicada

Escola Superior de Tecnologia e Gestão de Águeda
Universidade de Aveiro

- Debugging: permitem que os programadores identifiquem, diagnostiquem e corrijam erros no código. Ao registar informações importantes como valores de variáveis, chamadas de funções e erros, é possível seguir o fluxo de execução da aplicação e identificar a causa dos erros.
- Monitorização: fornecem informações sobre o desempenho da aplicação em tempo real. Permite detetar padrões de utilização e anomalias.
- Auditoria: servem como um registo histórico de eventos e ações realizadas no sistema. São úteis para identificar quem fez o quê e quando e para garantir a conformidade com as políticas de segurança e privacidade.

A forma mais simples de fazer este registo de dados é recorrer a uma biblioteca.

Biblioteca	**comando npm**
Morgan	npm install morgan
Winston	npm install winston
Bunyan	npm install bunyan
Pino	npm install pino
loglovel	npm install loglevel
npmlog	npm install npmlog

Introdução

Biblioteca Morgan

- Biblioteca relativamente simples de utilizar
- Mensagens personalizáveis recorrendo a tokens
- É utilizada pelo express
- Pode ser utilizada em projetos de pequena, média ou grande dimensão
- Permite definir a localização do log
- Permite definir múltiplos destinos

```
var express = require('express');
var logger = require('morgan');

var app = express();

app.use(logger('dev'));

app.get('/', function (req, res) {
  res.send('Hello, world!')
});
```

- Predefinidos
  - combined: define os registos como o formato combinado padrão do Apache
  - common: define os registos como o formato comum padrão do Apache
  - dev: formato de registo com cor, é possível identificar o comando HTTP, o recurso pedido, o código de estado e o tempo de resposta ao pedido
  - short: entre outros, identifica o endereço IP do cliente, o comando HTTP, o caminho, o recurso pedido, o código de estado e o tempo de resposta ao pedido
  - tiny: resume-se essencialmente ao comando http, caminho, código de estado e tempo de resposta
- Definidos manualmente recorrendo a tokens

combined

```
::ffff:127.0.0.1 - - [23/Mar/2023:14:20:49 +0000] "GET / HTTP/1.1" 200
  170 "-" "Mozilla/5.0 (X11; Linux x86_64; rv:109.0) Gecko/20100101
  Firefox/111.0"
```

common

```
::ffff:127.0.0.1 - - [23/Mar/2023:14:24:30 +0000] "GET / HTTP/1.1" 200 170
```

dev

```
GET / 200 182.522 ms - 170
```

short

```
::ffff:127.0.0.1 - GET / HTTP/1.1 200 170 - 182.370 ms
```

tiny

```
GET / 200 170 - 178.268 ms
```

String de registo com tokens predefinidos

```
app.use(logger(':method :url :status :res[content-length] -
  :response-time ms'));
```

Função com formato personalizado

```
app.use(logger((tokens, req, res) => {
  return [
      tokens.method(req, res),
      tokens.url(req, res),
      tokens.status(req, res),
      tokens['response-time'](req, res), 'ms'
  ].join(' ');
}))
```

https://expressjs.com/en/resources/middleware/morgan.html

- A biblioteca tem acesso aos objetos Request e Response
- Mesmo a headers HTTP personalizados
- Para criar tokens utilizamos o método token()

```
logger.token('host', function(req, res) {
  return req.headers['host'];
})
app.use(logger(':method :url :status :host'));
```

- Por omissão a biblioteca envia os registos para o standard output
- A biblioteca permite definir um destino alternativo (por exemplo um ficheiro)

```
var fs = require('fs');

var accessLogStream = fs.createWriteStream(path.join(__dirname,
  'access.log'), { flags: 'a' });

app.use(logger('combined', { stream: accessLogStream }));
```

- A opção skip é um atributo opcional do segundo argumento
- Podemos utilizar este atributo para identificar quais os eventos que devem ser ignorados pelo registo

```
app.use(logger('dev', {
  skip: function (req, res) { return res.statusCode < 400 }
}))

app.use(logger('common', {
  stream: fs.createWriteStream(path.join(__dirname, 'access.log'),
    { flags: 'a' })
}))
```