

Desafío Clase 32: Loggers, Gzip y Análisis de performance

Gzip

Diferencia de cantidad de bytes:

Con compresión:

Name	Status	Type	Initiator	Size	Time
info	304	document	Other	179 B	38 ms
bootstrap.min.css	200	stylesheet	info	(disk cache)	62 ms
installHook.js	200	script	prepareInjection.js:525	427 kB	281 ms
pagejsBaidu.js	200	script	content.js:2	718 B	205 ms
extn-utils.html	200	document	jquery-3.1.1.min.js:3	1.0 kB	4 ms
extn-utils.js	200	script	chrome-extension://e...	1.3 kB	5 ms

Sin compresión:

Name	Status	Type	Initiator	Size	Time
info	304	document	Other	179 B	41 ms
bootstrap.min.css	200	stylesheet	info	(disk cache)	27 ms
installHook.js	200	script	prepareInjection.js:...	427 kB	212 ms
pagejsBaidu.js	200	script	content.js:2	718 B	138 ms
extn-utils.html	200	document	jquery-3.1.1.min.js:3	1.0 kB	5 ms
extn-utils.js	200	script	chrome-extension://...	1.3 kB	11 ms

Conclusiones: no hay diferencia en cantidad de bytes, pero si en el tiempo de respuesta que es menor cuando se usa la ruta con compresión.

Análisis de performance

Procesamiento de resultados de test --prof con --prof-process

- Ruta '/info' sin console.log:

```
[Summary]:
  ticks  total  nonlib   name
    13     0.0%  100.0%  JavaScript
     0     0.0%   0.0%    C++
    16     0.1%  114.3%    GC
 30320  100.0%                Shared libraries
```

- Ruta '/info' con console.log:

```
[Summary]:
  ticks  total  nonlib   name
    14     0.0%   92.9%  JavaScript
     0     0.0%   0.0%    C++
    20     0.1%  142.9%    GC
 38866  100.0%                Shared libraries
```

Conclusiones: si bien la diferencia no es tan grande, se ve en el nro de ticks en la prueba de la ruta con y sin console.

Resultados de test de carga con Artillery:

- Ruta '/info' sin console.log:

```
-----
Summary report @ 20:39:14(-0300)
-----

http.codes.200: ..... 1000
http.request_rate: ..... 585/sec
http.requests: ..... 1000
http.response_time:
  min: ..... 1
  max: ..... 90
  median: ..... 43.4
  p95: ..... 64.7
  p99: ..... 89.1
http.responses: ..... 1000
```

- Ruta '/info' con console.log:

```
-----  
Summary report @ 11:25:55(-0300)  
-----
```

```
http.codes.200: ..... 1000  
http.request_rate: ..... 535/sec  
http.requests: ..... 1000  
http.response_time:  
  min: ..... 2  
  max: ..... 154  
  median: ..... 46.1  
  p95: ..... 79.1  
  p99: ..... 102.5  
http.responses: ..... 1000
```

Conclusiones: la diferencia en el tiempo de respuesta en la prueba de la ruta con y sin console es notable, siendo mayor en el segundo caso, tanto el tiempo mínimo, máximo como el promedio.

Resultados de test de carga con Autocannon:

- Ruta '/info' sin console.log:

```
Running 20s test @ http://localhost:8080/info  
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	22 ms	1026 ms	1670 ms	1837 ms	987.68 ms	396.02 ms	2061 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	46	46	94	158	99.05	30.35	46
Bytes/Sec	111 kB	111 kB	228 kB	383 kB	240 kB	73.5 kB	111 kB

```
Req/Bytes counts sampled once per second.  
# of samples: 20
```

```
2k requests in 20.13s, 4.8 MB read
```

- Ruta '/info' con console.log:

```
2k requests in 20.13s, 4.8 MB read
Running 20s test @ http://localhost:8080/info-console
100 connections
```

Stat	2.5%	50%	97.5%	99%	Avg	Stdev	Max
Latency	630 ms	1135 ms	3551 ms	3775 ms	1231.36 ms	598.12 ms	3867 ms

Stat	1%	2.5%	50%	97.5%	Avg	Stdev	Min
Req/Sec	0	0	64	155	77.85	38.83	13
Bytes/Sec	0 B	0 B	154 kB	372 kB	187 kB	93.2 kB	31.2 kB

```
Req/Bytes counts sampled once per second.
# of samples: 20
```

```
2k requests in 20.5s, 3.74 MB read
```

Conclusiones: el tiempo de latencia es mayor en la ruta con console.log, la respuesta es más rápida en el caso de la ruta sin console.log.

Perfilamiento del servidor con el modo inspector de node.js

```
13     const memoria = process.memoryUsage();
14     const pathExe = process.execPath;
15     const processId = process.pid;
16     const carpeta = process.cwd();
17     const procesadores = cpu.length;
18
19     infoRouter.get("/info-console", (req, res) => {
20         console.log(`Argumentos: ${argumentos}
21                     Plataforma: ${plataforma}
22                     Version: ${version}
23                     Memoria: ${memoria}
24                     PathExe: ${pathExe}
25                     IdProcesador: ${processId}
26                     Carpeta: ${carpeta}
27                     Procesadores: ${procesadores}`);
28         res.render(path.join(process.cwd(), "/views/pages/info.ejs"), {
29             argumentos: argumentos,
30             plataforma: plataforma,
31             version: version,
32             memoria: memoria.rss,
33             pathExe: pathExe,
34             processId: processId,
35             carpeta: carpeta,
36             procesadores: procesadores,
37         });
38     });
39
40     infoRouter.get("/info", compression(), (req, res) => {
41         res.render(path.join(process.cwd(), "/views/pages/info.ejs"), {
42             argumentos: argumentos,
43             plataforma: plataforma,
44             version: version,
45             memoria: memoria.rss,
46             pathExe: pathExe,
47             processId: processId,
48             carpeta: carpeta,
49             procesadores: procesadores,
50         });
51     });
52
53     export default infoRouter;
```

Línea 40, columna 40

Conclusiones: En el caso de la ruta info con console.log, podemos ver que tiene el proceso menos performante con tiempo de carga de 12ms; en el caso de la ruta sin console.log, cuenta con un proceso máximo de 6.9ms, prácticamente la mitad que el anterior.

Diagrama de Flama:

node server.js

cold ■ ■ ■ hot
* optimized ~ unoptimized - +



Conclusiones: El lado izquierdo representa la ruta info sin console.log mientras en el derecho se realiza mayor procesamiento (por la ruta info con el console.log con los datos) y por lo tanto el gráfico es más ancho.

Conclusiones Finales

Los distintos test demuestran que la ruta info con console.log consume mayor tiempo de procesamiento que cuando se analiza la ruta sin console.log, de todos modos la diferencia es mínima al tratarse de test a nivel local y que el servidor se ejecuta en los todos los casos en modo fork.