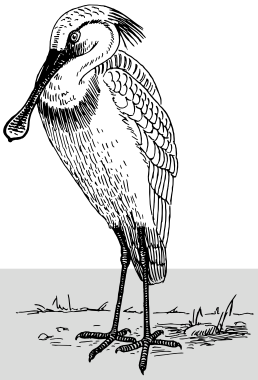


This document was generated by docboon

Caddy日志参考手册

格式、配置和使用

https://docboon.github.io/

Caddy日志参考手册: 格式、配置和使用

https://docboon.github.io/
Mr. Older Flower

出版日期 October 10th,2021

修订历史		
修订 0.1	2021年10月13日.	<olderflower @outlook.com>
文档建立		

目录

插图清单

排版约定

在正文中会有一些诸如程序代码、系统命令或是屏幕输出一类的信息，为了能清晰地展示各种元素，本手册遵守下列排版约定。

1. **字体约定**，在正文中嵌入的一些系统命令、文件名、函数或是参数等使用有别于正文的字体：

   a. 等宽字体：程序片段、正文中出现的配置选项、变量、函数名等，我们采用等宽字体，样式如下：`serverip`。

   b. 等宽微粗字体：表示由用户输入的系统命令。例如，在linux下，我们查网络接口信息的命令：`ifconfig eth0`。

   c. 等宽斜体：表示由用户输入的值或是一些需要设定的参数的值，例如，用户在浏览器里输入的一个url，用如下的样式显示：*`http://docboon.io`*。

   d. 等宽微粗斜体字体：表示文件名、数据库名及新的术语等。例如，文件的名称用如下的样式来显示：***`myfirst.xml`***。

2. **提示、告诫类信息**（admonitions），有一些内容是正文的补充或是对用户的一种提醒。为了能清晰地展示这部分内容，采用独立的“信息块”来呈现。一共有5种，分别为：

   - 

     提示

     可以使用`apt-get upgrade`来升级您的ubuntu或debian Linux系统。

   - 

     注意

     在运行`apt-get upgrade`之前，需要先运行用`apt-get update`命令。

   - 

     重要

     请及时升级系统补丁程序，这是保证系统处于安全状态的好方法。

   - 

     小心

     如果系统升级还没有完成，请不要重新启动您的电脑。

   - 

     警告

     在系统升级过程中千万不要关闭电源。

3. **屏幕内容围栏及程序代码围栏：**

   - 屏幕内容围栏，有时候我们需要整块显示在屏幕上输出的内容，或者是键入的命令。例如，在linux下，我们使用`ifconfig eth0`命令来查网络接口信息：

```
$ ifconfig eth0
eth0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 98:fa:9b:db:b9:9f  txqueuelen 1000  (以太网)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
```

```
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0xea300000-ea320000
```

- 程序代码围栏，对于一些程序代码，使用带有序号的样式来展示，例如，一段php代码：

```
 1 <?php
 2   class SimpleBook {
 3     private $title;
 4
 5     function __construct($title_in) {
 6       $this->title  = $title_in;
 7     }
 8
 9     function getTitle() {return $this->title;}
10   }
11 ?>
```

第 1 章 关于caddy

1. caddy概述

caddy目前是ZeroSSL在维护的一个开源项目。ZeroSSL[1]和Let's Encrypt[2]一样，是一个SSL证书颁发机构。Caddy属于web服务器界的新秀。用go语言开发。与其他的web服务器相比，因为有了ZeroSSL的加持，caddy最大的一个特点就是它自带TLS，能非常方便的部署HTTPS。按照官方步骤安装好了以后无需任何配置，就已经自带了https功能。同时，caddy也非常好地支持诸如代理、负载均衡等功能。在HTTP/2协议上，caddy表现也非常好。和很多的开源软件一样，caddy的文档也并不是特别的全面，有一些选项和指令需要自己去实验才能明白具体如何使用。

caddy的另外一个特点，是可以通过nginx-adapter[3]模块来使用nginx的配置文件。caddy第一个版本发布于2015年4月，截至本文完稿，caddy的最新版本是2.4.5。

关于caddy更多的内容可以参考它的官方网站：https://caddyserver.com/。

2. caddy日志概览

caddy的日志在设计上与httpd及nginx等前辈有一些不太一样的地方，httpd和nginx默认情况下使用了Common Log Format(CLF)和Combined Log Format(Combined)日志格式。而caddy使用了json文件格式来记录日志，而且字段也要多出很多。caddy的日志设计带有了许多“大数据”方面的理念：

1. 日志数据及日志条目中的字段越多越好
2. 对日志进行过滤清洗后保存比直接丢弃好
3. 合适的编码格式以提高日志数据的灵活性和交互能力

在caddy的官网有一篇小文章，简单介绍了其日志系统的理念：https://caddyserver.com/docs/logging，包括caddy在日志系统中的角色、caddy日志的结构、处理流程及日志的使用等。

2.1. caddy在日志系统中的角色

通常情况下，在日志系统包含两方面主要的内容：emission（产生日志）和 consumption（使用）

1. Emission：意味着产生日志信息，它通常包含三个步骤：

[1]https://zerossl.com/

[2]https://letsencrypt.org/

[3]https://github.com/caddyserver/nginx-adapter

a. 收集相关的信息；

b. 按照一定的格式对信息进行封装；

c. 将封装好的信息发送到适当的地方。

这三个部分的功能已经包含在caddy的核心代码中，任何的模块和插件都可以调用这部分功能。

2. **consumption**：日志的使用和处理。没有被使用的日志是没有任何价值的。只有正确的使用日志，才能使其产生应有的价值。

在这两个概念里，caddy属于前者，是一个日志生产者。

2.2. caddy结构化的日志

caddy采用了更为结构化的日志系统，JSON格式的日志是caddy比较建议使用的格式。如果不想使用JSON格式，也可以在配置文件中使用encoder指令来指定其他的格式。如下的内容是caddy官方网站所展示的一段JSON格式的日志内容：

```
{
 "level": "info",
 "ts": 1585597114.7687502,
 "logger": "http.log.access",
 "msg": "handled request",
 "request": {
  "method": "GET",
  "uri": "/",
  "proto": "HTTP/2.0",
  "remote_addr": "127.0.0.1:50876",
  "host": "example.com",
  "headers": {
   "User-Agent": [ "curl/7.64.1"],
   "Accept": ["*/*"]
  },
  "tls": {
   "resumed": false,
   "version": 771,
   "ciphersuite": 49196,
   "proto": "h2",
   "proto_mutual": true,
   "server_name": "example.com"
  }
 },
 "user_id": "",
 "duration": 0.000014711,
 "size": 2326,
 "status": 200,
 "resp_headers": {
  "Server": ["Caddy"],
  "Content-Type": ["text/html"]
 }
}
```

2.3. 处理流程

caddy中处理log的程序称为**logger**，日志信息被**logger**收集并发送到**log**进行处理，caddy允许配置多个**log**s。一个**log**包含如下几个部分：

1. encoder

   日志采用的编码格式，包括console、json、filter及single_field等。

2. writer

   配置日志输出的地方。可以是一个文件也可以是一个网络地址。

3. level

   日志的级别，从DEBUG到FATAL，默认是INFO。

4. sampling

   日志采样的频率。

5. include/exclude

   每一条信息都是被一个logger所发出，这个logger有一个名字，通常是信息所在模块的名称，使用include/exclude机制，在记录日志时可以包含或排除某些指定的模块。

第 2 章 配置(Caddyfile中日志的配置)

caddy有三种配置方式：*Caddyfile*、json和API的方式。*Caddyfile*对人来说是比较容易阅读和容易使用的一种，这个章节介绍如何在*Caddyfile*中配置caddy的日志。

1. 语法（syntax）

```
log {
 output ❶ <writer_module> ...
 format ❷ <encoder_module> ...
 level  ❸ <level>
}
```

❶ output：日志输出到什么地方，默认值是stderr

❷ format：日志的输出格式。如果output配置为stdout的话，默认值就是console，否则就是json

❸ level：日志的级别，默认值是INFO。目前，caddy日志仅输出INFO和ERROR两个级别。

https://caddyserver.com/docs/caddyfile/directives/log

2. 输出（Output modules）

output指令让我们定义将日志输出到什么地方，一共有6个配置选项：

1. stderr

   这个选项是将日志直接输出到控制台（console），也是caddy默认使用的选项。

   ```
   output stderr
   ```

   different

2. stdout

   将日志直接输出到控制台上。

   ```
   output stdout
   ```

   提示

   笔者试验的结果，本以为stderr只是在控制台输出错误日志，但stderr和stdout似乎输出日志的结果是一样的，都是在console台上输出，也没看出什么差别。

3. discard

不在任何地方输出日志信息。

```
output discard
```

4. `file`

   把日志记录到文件中，默认情况下，为防止磁盘耗尽，和其他系统日志一样，caddy的新日志会采用某种机制覆盖老的日志（滚动机制），有几个选项来配置：

   ```
   output file <filename> {
    roll_disabled
    roll_size     <size>
    roll_keep     <num>
    roll_keep_for <days>
   }
   ```

   a. `filename`：指定日志输出到哪个文件，例如：*/var/log/caddy/access.log*

   b. `roll_disabled`：禁用日志滚动。这可能会导致磁盘空间耗尽，因此只有在有其他方式维护日志文件时才使用此方法。

   c. `roll_size`：滚动日志文件的大小，默认值是100MB。

   d. `roll_keep`：在删除旧日志文件之前所保存的日志文件的数量，默认值是10。

   e. `roll_keep_for`：is how long to keep rolled files as a duration string. The current implementation supports day resolution; fractional values are rounded up to the next whole day. For example, 36h (1.5 days) is rounded up to 48h (2 days). Default: 2160h (90 days)

5. `net`

   将日志输出到网络中的其他日志服务器。如果日志服务器不可用，那caddy会将日志输出到`stderr`，直到日志服务器恢复。

   ```
   output net <address> {
    dial_timeout <duration>
   }
   ```

   a. `address`：有两个部分组成`protocol/address`，`protocol`指得是协议，默认为`tcp`，`address`指得是地址，如192.168.1.10:514。

      例如，我们要把日志记录到日志服务器192.168.1.10，其端口是514，我们可以使用如下的指令：

      ```
      log {
        output net tcp/127.0.0.1:514
      }
      ```

   b. `dial_timeout`：成功地写到日志服务器中可以等待的时间。

      提示

      `dial_timeout`似乎应该有个默认值，但caddy的官方文档并没有给出，查看caddy的源代码，在日志模块文件*modules/logging/netwriter.go*中也没有找到相应的设置。

3. 格式（Format modules）

format指令可以让我们指定日志的格式，即日志信息如何来封装。这个指令包含在log里面，

```
log {
  format json
}
```

3.1. 格式类型

1. console

   使用如下命令指定console

   ```
   format console
   ```

   这种方式的日志格式看起来是混合了文本与JSON两种格式，也是caddy默认使用的日志格式。每行日志的前面是以时间戳为首的4个字段，后面是一段JSON格式的日志信息，如下：

   ```
   2021/11/26 06:32:43.823 info http.log.access.log0 handled request
   {
     "request": {
       "remote_addr": "192.168.31.56:42292",
       "proto": "HTTP/2.0",
       "method": "GET",
       "host": "192.168.31.9",
       "uri": "/index.html",
       "headers": {
         "Accept-Language": ["zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2" ],
         "Accept-Encoding": ["gzip, deflate, br"],
         "Upgrade-Insecure-Requests": [ "1" ],
         "Te": ["trailers"],
         "User-Agent": [ "Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0" ],
         "Accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8"]
       },
       "tls": {
         "resumed": false,
         "version": 772,
         "cipher_suite": 4865,
         "proto": "h2",
         "proto_mutual": true,
         "server_name": ""
       }
     },
     "common_log":"192.168.31.56 - -[26/Nov/2021:06:32:43 +0000] \"GET /index.html HTTP/2.0\" 200 7",
     "user_id": "",
     "duration": 0.005788781,
     "size": 7,
     "status": 200,
     "resp_headers": {
       "Server": ["Caddy"],
       "Etag": ["\"r2yh597\""],
       "Content-Type": ["text/html; charset=utf-8"],
       "Last-Modified": ["Mon, 22 Nov 2021 04:15:09 GMT"],
       "Accept-Ranges": ["bytes"],
       "Content-Length": ["7"]
     }
   }
   ```

   前面的4个字段是：level、time、name、msg。

2. `json`

   json指令指定日志使用json格式。这会指定每条日志条目都使用json格式

   ```
   format json
   ```

   可以使用命令`cat access.log | jq`来格式化日志信息。

3. `single_field`

   这是一个被弃用、并且在未来的某一个版本中就会被移除的格式。

   ```
   format single_field <field_name>
   ```

   如果我们想要common_log格式的日志，可以使用这个命令：`format single_field common_log`。在caddy的文档中，这个功能已经被**format-encoder**替代。https://github.com/caddyserver/format-encoder

4. `filter`

   `filter`是一种可以根据需要对日志进行“筛选”的格式，它需要将json或console格式中的一种重新“包装”起来，然后再对字段进行“筛选”。

   ```
   format filter {
    wrap json ...
    fields {
     <field> <filter> ...
    }
   }
   ```

   - `<field>`：因为fields是可以嵌套的，在这里，为了表示嵌套的关系可以使用>来表示，例如有一个对象{"a":{"b":0}}，我们就可以使用a>b来表示。
   - `<filter>`：目前有3个“筛选器”：`delete`、`replace`和`ip_mask`

   其中，`ts`、`level`、`logger`和`msg`四个字段不能应用于“筛选”机制。这个四个字段是日志的基础内容，是由日志系统而非caddy决定的。

   以下是“筛选”器的说明：

   a. `delete`

      使用这个命令来标注哪个字段将被忽略。

      ```
      <field> delete
      ```

   b. `replace`

      使用这个命令来标记某个字段使用自定义的字符串替代。

      ```
      <field> replace <replacement>
      ```

   c. `ip_mask`

标记IP地址使用CIDR掩码，以IPv4的ip地址例，有32个bit为，如果标记为ipv4 24，那么在日志中只记录前24个bit位，后面的8个记录为0，例如，有一个ip地址192.168.2.1，那么会被记录为192.168.2.0。使用方式如下：

```
<field> ip_mask {
 ipv4 <cidr>
 ipv6 <cidr>
}
```

作为filter的配置，我们可以用下面的例子来说明：

```
log {
  format filter { ❶
    wrap console
    fields {
      common_log delete ❷
      request>remote_addr ip_mask {
        ipv4 24 ❸
        ipv6 32
      }
    }
  }
}
```

❶ 在这个filter中，封装了console

❷ 将common_log部分删除

❸ 对ipv4的ip地址只记录前24位，ipv6的ip地址只记录前32位

3.2. 格式属性

对每种日志格式，特别是console和json还有一些属性可以设置：

```
format <encoder_module> {
    message_key <key>
    level_key   <key>
    time_key    <key>
    name_key    <key>
    caller_key  <key>
    stacktrace_key <key>
    line_ending  <char>
    time_format  <format>
    level_format <format>
}
```

1. message_key

   日志条目中“message”字段的关键字，默认值是"msg"。

2. level_key

   日志条目中“level”字段的关键字，默认值是“level”。

3. time_key

   日志条目中“time”字段的关键字，默认值是“ts”。

4. name_key

日志条目中“name”字段的关键字，默认值是“logger”。

提示

在caddy的官方文档中，说这个字段的默认值是"name"，但笔者试验下来，其实是"logger"。

5. `caller_key`

   日志条目中“caller”字段的关键字。

6. `stacktrace_key`

   日志条目中“stracktrace”字段的关键字。

7. `line_ending`

   每条日志条目的结尾字符，默认是回车符。

8. `time_format`

   日志条目中时间戳的格式。

   笔者没有在caddy的官方文档中找到具体有哪些“时间格式”，通过查阅源代码，在*modules/logging/encoders.go*文件中找到如下代码：

```
// time format
var timeFormatter zapcore.TimeEncoder
switch lec.TimeFormat {
case "", "unix_seconds_float":
 timeFormatter = zapcore.EpochTimeEncoder
case "unix_milli_float":
 timeFormatter = zapcore.EpochMillisTimeEncoder
case "unix_nano":
 timeFormatter = zapcore.EpochNanosTimeEncoder
case "iso8601":
 timeFormatter = zapcore.ISO8601TimeEncoder
default:
 timeFormat := lec.TimeFormat
 switch lec.TimeFormat {
 case "rfc3339":
  timeFormat = time.RFC3339
 case "rfc3339_nano":
  timeFormat = time.RFC3339Nano
 case "wall":
  timeFormat = "2006/01/02 15:04:05"
 case "wall_milli":
  timeFormat = "2006/01/02 15:04:05.000"
 case "wall_nano":
  timeFormat = "2006/01/02 15:04:05.000000000"
 case "common_log":
  timeFormat = "02/Jan/2006:15:04:05 -0700"
 }
 timeFormatter = func(ts time.Time, encoder zapcore.PrimitiveArrayEncoder) {
  encoder.AppendString(ts.UTC().Format(timeFormat))
 }
}
cfg.EncodeTime = timeFormatter
```

从这段代码中可以看到，时间格式有如下几种：

unix_seconds_float、unix_milli_float、unix_nano、iso8601、rfc3339、rfc3339_nano、wall、 wall_milli、wall_nano、common_log。

其默认值是unix_seconds_float。

9. level_format

日志条目中level的格式。

同样，在caddy的官方文档中没有找到具体有哪些“level格式”，通过查阅源代码，在*modules/logging/encoders.go*文件中找到如下代码：

```
// level format
var levelFormatter zapcore.LevelEncoder
switch lec.LevelFormat {
case "", "lower":
 levelFormatter = zapcore.LowercaseLevelEncoder
case "upper":
 levelFormatter = zapcore.CapitalLevelEncoder
case "color":
 levelFormatter = zapcore.CapitalColorLevelEncoder
}
cfg.EncodeLevel = levelFormatter
```

从这段代码中可以看到，时间格式有如下几种：lower、upper、color。

其默认值是lower。即小写、大写和彩色。当设置为color时，在日志条目中level字段会有一段彩色编码："level":"\u001b[34mINFO\u001b[0m" ...

提示

笔者实验下来，在所有这些格式属性中，大部分只是对json格式的日志有效，对console等格式是无效的，例如：message_key、level_key、time_key和name_key这四项，在console的格式配置中是无意义的，因为console日志格式的第一行前四项就是这四个字段，他们以"无key"的方式直接显示。

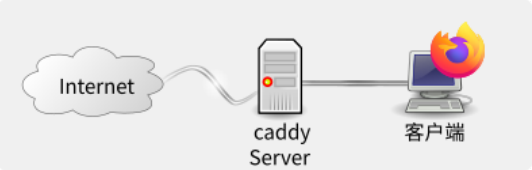
第 3 章 配置案例及日志解析

为了能更清晰地描述，我们在局域网中设立一台caddy服务器，我们称为**Server**，其内网IP地址是192.168.31.2/24，外网IP地址是192.168.1.2/24。我们在其上安装Debian Linux（V12）。使用如下命令安装caddy：

```
$ sudo apt install -y debian-keyring debian-archive-keyring apt-transport-https
$ curl -1sLf 'https://dl.cloudsmith.io/public/caddy/testing/gpg.key' | \
      sudo tee /etc/apt/trusted.gpg.d/caddy-testing.asc
$ curl -1sLf 'https://dl.cloudsmith.io/public/caddy/testing/debian.deb.txt' | \
      sudo tee /etc/apt/sources.list.d/caddy-testing.list
$ sudo apt update
$ sudo apt install caddy
```

安装结束后，caddy的配置文件位于*/etc/caddy*目录下。工作目录位于*/usr/share/caddy*目录下。

图 3.1. caddy服务器网络拓扑图



在配置和查看caddy日志过程中会碰到很多json格式的文件，为了能更好的查看这些文件，需要一个json格式化工具jq。jq是一个基于命令行的、轻量级JSON文件处理工具。在Debian或Ubuntu下可通过`apt-get install jq`来安装。关于其更多内容请访问：https://stedolan.github.io/jq/。

1. 静态文件服务器

1.1. 默认情况

编辑*/etc/caddy/Caddyfile*文件，内容如下：

```
1 192.168.31.2 {
2 root * /usr/share/caddy
3 file_server
4 log {
5       output file /var/log/caddy/access.log ❶
6     }
7 }
```

❶ 指定日志的输出路径为*/usr/share/caddy/access.log*。

在配置文件中没有指定日志的格式，当我们从浏览器访问*https://192.168.31.2/index.html*时，会产生如下的一条日志（为了方便显示和说明，我们将这条日志信息进行了分行处理，实际上下面的日志信息是一条记录）：

```
2021/12/29 03:07:44.337❶   info❷   http.log.access.log0❸    handled request❹
{
  "request": {
    "remote_addr": "192.168.31.56:50946",
    "proto": "HTTP/2.0",
    "method": "GET",
    "host": "192.168.31.2",
    "uri": "/index.html",
    "headers": {
      "Accept-Encoding": ["gzip, deflate, br"],
      "Sec-Fetch-Mode": ["navigate"],
      "Sec-Fetch-User": ["?1"],
      "Sec-Fetch-Dest": ["document"],
      "Sec-Ch-Ua-Platform": ["\"Linux\""],
      "Upgrade-Insecure-Requests": ["1"],
      "User-Agent": ["Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
                     Chrome/96.0.4664.110 Safari/537.36"],
      "Accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
                  image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"],
      "Sec-Fetch-Site": ["none"],
      "Cache-Control": ["max-age=0"],
      "Sec-Ch-Ua": ["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google Chrome\";
                    v=\"96\""],
      "Sec-Ch-Ua-Mobile": ["?0"],
      "Accept-Language": ["zh-CN,zh;q=0.9"]
    },
    "tls": {
      "resumed": false,
      "version": 772,
      "cipher_suite": 4867,
      "proto": "h2",
      "proto_mutual": true,
      "server_name": ""
    }
  },
  "common_log": "192.168.31.56 - - [29/Dec/2021:11:07:44 +0800] \"GET /index.html HTTP/2.0\"
                 200 12226",
  "user_id": "",
  "duration": 0.005571188,
  "size": 12226,
  "status": 200,
  "resp_headers": {
    "Accept-Ranges": ["bytes"],
    "Content-Length": ["12226"],
    "Server": ["Caddy"],
    "Etag": ["\"r29u0d9fm\""],
    "Content-Type": ["text/html; charset=utf-8"],
    "Last-Modified": ["Mon, 08 Nov 2021 20:53:01 GMT"]
  }
}
```

❶ 时间戳，默认情况下使用了wall-milli。wall的意思是“墙上的钟”，即这个格式是我们可以很容易辨认的格式，milli是毫秒级的意思，即时间精确到毫秒；

❷ 日志的level，默认情况下是info；

❸ 产生日志的模块；

❹ 消息（msg）名称。

上面的四个元素包含在第一行中，剩下的内容被封装在json结构中。

1.2. console格式

我们接下来更改*Caddyfile*文件，使用console的格式来封装日志信息，并使用caddy run命令来重新运行caddy：

```
192.168.31.2 {
root * /usr/share/caddy
file_server
log {
      output file /var/log/caddy/access.log
      format console❶
    }
}
```

❶ 指定日志格式是console。

```
1.6407473377283428e+09❶ info http.log.access.log0 handled request
{
  "request": {
    "remote_addr": "192.168.31.56:50950",
    "proto": "HTTP/2.0",
    "method": "GET",
    "host": "192.168.31.34",
    "uri": "/index.html",
    "headers": {
      "Accept-Encoding": ["gzip, deflate, br"],
      "Cache-Control": ["max-age=0"],
      "Sec-Ch-Ua": ["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google Chrome\";
                    v=\"96\""],
      "Sec-Ch-Ua-Platform": ["\"Linux\""],
      "User-Agent": ["Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
                      Chrome/96.0.4664.110 Safari/537.36"],
      "Sec-Fetch-Mode": ["navigate"],
      "Sec-Fetch-Dest": ["document"],
      "Accept-Language": ["zh-CN,zh;q=0.9"],
      "Sec-Ch-Ua-Mobile": ["?0"],
      "Upgrade-Insecure-Requests": ["1"],
      "Accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
         image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"],
      "Sec-Fetch-Site": ["none"],
      "Sec-Fetch-User": ["?1"]
    },
    "tls": {
      "resumed": false,
      "version": 772,
      "cipher_suite": 4867,
      "proto": "h2",
      "proto_mutual": true,
      "server_name": ""
    }
  },
  "common_log": "192.168.31.56 - - [29/Dec/2021:11:08:57 +0800] \"GET /index.html HTTP/2.0\"
                 200 12226",
  "user_id": "",
  "duration": 0.00506029,
  "size": 12226,
  "status": 200,
  "resp_headers": {
    "Last-Modified": ["Mon, 08 Nov 2021 20:53:01 GMT"],
    "Accept-Ranges": ["bytes"],
    "Content-Length": ["12226"],
    "Server": ["Caddy"],
    "Etag": ["\"r29u0d9fm\""],
    "Content-Type": ["text/html; charset=utf-8"]
  }
```

```
50 }
```

❶ 时间戳，console格式下的时间戳和默认情况的格式还是不一样的。console使用了unix_milli_float类型；

1.3. json格式

我们接下来更改*Caddyfile*文件，使用json的格式来封装日志信息，并使用caddy run命令来重新运行caddy：

```
1 192.168.31.2 {
2 root * /usr/local/work/example.com
3 file_server
4 log {
5       output file /var/log/caddy/access.log
6       format json ❶
7     }
8 }
```

❶ 指定日志格式是json。

所有的日志信息就被封装成一条json结构的数据：

```
1 {
2   "level": "info",
3   "ts": 1640751044.7156549,
4   "logger": "http.log.access.log0",
5   "msg": "handled request",
6   "request": {
7     "remote_addr": "192.168.31.56:50962",
8     "proto": "HTTP/2.0",
9     "method": "GET",
10    "host": "192.168.31.34",
11    "uri": "/index.html",
12    "headers": {
13      "Sec-Ch-Ua-Mobile": ["?0"],
14      "Upgrade-Insecure-Requests": ["1"],
15      "User-Agent": ["Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
16                      Chrome/96.0.4664.110 Safari/537.36"],
17      "Sec-Fetch-Mode": ["navigate"],
18      "Sec-Fetch-Site": ["none"],
19      "Sec-Fetch-User": ["?1"],
20      "Sec-Fetch-Dest": ["document"],
21      "Accept-Encoding": ["gzip, deflate, br"],
22      "Cache-Control": ["max-age=0"],
23      "Sec-Ch-Ua": ["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google Chrome\";
24                     v=\"96\""],
25      "Sec-Ch-Ua-Platform": ["\"Linux\""],
26      "Accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
27                  image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"],
28      "Accept-Language": ["zh-CN,zh;q=0.9"]
29    },
30    "tls": {
31      "resumed": false,
32      "version": 772,
33      "cipher_suite": 4867,
34      "proto": "h2",
35      "proto_mutual": true,
36      "server_name": ""
37    }
38  },
39  "common_log": "192.168.31.56 - - [29/Dec/2021:12:10:44 +0800] \"GET /index.html HTTP/2.0\"
```

```
40                 200 12239",
41   "user_id": "",
42   "duration": 0.000896033,
43   "size": 12239,
44   "status": 200,
45   "resp_headers": {
46     "Server": ["Caddy"],
47     "Etag": ["\"r4uzlo9fz\""],
48     "Content-Type": ["text/html; charset=utf-8"],
49     "Last-Modified": ["Wed, 29 Dec 2021 04:10:36 GMT"],
50     "Accept-Ranges": ["bytes"],
51     "Content-Length": ["12239"]
52   }
53 }
```

1.4. 格式属性案例

我们接下来在*Caddyfile*文件中加入更多的封装属性，并使用caddy run命令来重新运行caddy：

```
1 192.168.31.2 {
2 root * /usr/share/caddy
3 file_server
4 log {
5       output file /var/log/caddy/access.log
6       format json { ❶
7           message_key FileServerMSG ❷
8           level_key FileServerLevel❸
9           time_key TimeStamp ❹
10          name_key FileServerLogger❺
11          time_format unix_seconds_float ❻
12          level_format upper ❼
13      }
14    }
15 }
```

❶ 指定日志格式是json。

❷ 消息的关键字。

❸ level的关键字。

❹ 时间戳关键字。

❺ 名称关键字。

❻ 时间格式。

❼ level格式。

所有的日志信息就被封装成一条json结构的数据：

```
1 {
2   "FileServerLevel": "INFO",
3   "TimeStamp": 1640847492.5563374,
4   "FileServerLogger": "http.log.access.log0",
5   "FileServerMSG": "handled request",
6   "request": {
7     "remote_addr": "192.168.31.56:39920",
8     "proto": "HTTP/2.0",
9     "method": "GET",
10    "host": "192.168.31.34",
11    "uri": "/",
12    "headers": {
13      "Sec-Ch-Ua": ["\" Not A;Brand\";v=\"99\", \"Chromium\";v=\"96\", \"Google Chrome\";
14       v=\"96\""],
15      "Sec-Ch-Ua-Platform": ["\"Linux\""],
```

```
      "Sec-Fetch-Mode": ["navigate"],
      "Accept-Encoding": ["gzip, deflate, br"],
      "Sec-Fetch-User": ["?1"],
      "Sec-Fetch-Dest": ["document"],
      "Cache-Control": ["max-age=0"],
      "Sec-Ch-Ua-Mobile": ["?0"],
      "Upgrade-Insecure-Requests": ["1"],
      "User-Agent": ["Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
                     Chrome/96.0.4664.110 Safari/537.36"],
      "Accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,
                  image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9"],
      "Sec-Fetch-Site": ["none"],
      "Accept-Language": ["zh-CN,zh;q=0.9"]
    },
    "tls": {
      "resumed": false,
      "version": 772,
      "cipher_suite": 4867,
      "proto": "h2",
      "proto_mutual": true,
      "server_name": ""
    }
  },
  "common_log": "192.168.31.56 - - [30/Dec/2021:14:58:12 +0800] \"GET / HTTP/2.0\" 200 12239",
  "user_id": "",
  "duration": 0.005645898,
  "size": 12239,
  "status": 200,
  "resp_headers": {
    "Content-Type": ["text/html; charset=utf-8"],
    "Last-Modified": ["Wed, 29 Dec 2021 04:10:36 GMT"],
    "Accept-Ranges": ["bytes"],
    "Content-Length": ["12239"],
    "Server": ["Caddy"],
    "Etag": ["\"r4uzlo9fz\""]
  }
}
```


2. 反向代理服务器

我们接下来更改*Caddyfile*文件，使用json的格式来封装日志信息，并使用caddy run命令来重新运行caddy：

```
:80 {
 file_server
 reverse_proxy 192.168.31.34:80 192.168.31.66:80
 log {
  output file /var/log/caddy/reverse_proxy.log
  format json     ❶
 }
}
```

❶ 指定日志格式是json。

所有的日志信息就被封装成一条json结构的数据：


```
{
  "level": "info",
  "ts": 1640923648.1757836,
  "logger": "http.log.access.log0",
  "msg": "handled request",
  "request": {
    "remote_addr": "192.168.31.56:35206",
```

```
    "proto": "HTTP/1.1",
    "method": "GET",
    "host": "192.168.31.9",
    "uri": "/index.html",
    "headers": {
      "Connection": ["keep-alive"],
      "If-Modified-Since": ["Mon, 15 Nov 2021 08:30:02 GMT"],
      "If-None-Match": ["W/\"61921a8a-2b1\""],
      "Cache-Control": ["max-age=0"],
      "Upgrade-Insecure-Requests": ["1"],
      "User-Agent": ["Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"],
      "Accept": ["text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8"],
      "Accept-Language": ["zh-CN,zh;q=0.8,zh-TW;q=0.7,zh-HK;q=0.5,en-US;q=0.3,en;q=0.2"],
      "Accept-Encoding": ["gzip, deflate"]
    }
  },
  "common_log": "192.168.31.56 - - [31/Dec/2021:12:07:28 +0800] \"GET /index.html HTTP/1.1\"
                 200 399",
  "user_id": "",
  "duration": 0.001988705,
  "size": 399,
  "status": 200,
  "resp_headers": {
    "Date": ["Fri, 31 Dec 2021 04:07:28 GMT"],
    "Server": ["Caddy","nginx/1.18.0"],
    "Content-Type": ["text/html"],
    "Last-Modified": ["Fri, 31 Dec 2021 04:07:06 GMT"],
    "Etag": ["W/\"61ce81ea-27f\""],
    "Content-Encoding": ["gzip"]
  }
}
```


nginx服务器的日志：

```
192.168.31.9 - - [31/Dec/2021:12:07:28 +0800] "GET /index.html HTTP/1.1" 200 411 "-"
             "Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0"
```