端到端的 Trace ID 解决方案

这篇博客文章由 *ChatGPT-4o* 协助撰写。

我开发了一个端到端的 Trace ID 解决方案，以确保系统中的每个请求和响应都能在前端和后端之间一致地追踪。此解决方案通过将每个操作与唯一的 Trace ID 关联起来，帮助调试、监控和日志记录。下面是该解决方案的详细说明及代码示例。

工作原理

前端

解决方案的前端部分涉及为每个请求生成一个 Trace ID，并将其与客户端信息一起发送到后端。该 Trace ID 用于跟踪请求在后端处理的各个阶段。

1. **收集客户端信息**：我们从客户端收集相关信息，例如屏幕尺寸、网络类型、时区等。这些信息会随请求头一起发送。
2. **生成 Trace ID**：为每个请求生成一个唯一的 Trace ID。这个 Trace ID 被包含在请求头中，使我们能够追踪请求的整个生命周期。
3. **API Fetch**：`apiFetch` 函数用于发起 API 调用。它在每个请求的头中包含 Trace ID 和客户端信息。

后端

解决方案的后端部分涉及在每条日志消息中记录 Trace ID，并在响应中包含 Trace ID。这使我们能够跟踪请求在后端处理的过程，并将响应与请求匹配。

1. **Trace ID 处理**：后端从请求头中接收 Trace ID，如果没有提供，则生成一个新的 Trace ID。该 Trace ID 存储在 Flask 全局对象中，以便在请求生命周期内使用。
2. **日志记录**：自定义日志格式器用于在每条日志消息中包含 Trace ID。这确保了与请求相关的所有日志消息都可以使用 Trace ID 进行关联。
3. **响应处理**：Trace ID 被包含在响应头中。如果发生错误，Trace ID 也会被包含在错误响应体中，以帮助调试。

Kibana

Kibana 是一个强大的工具，用于可视化和搜索存储在 Elasticsearch 中的日志数据。通过我们的 Trace ID 解决方案，您可以轻松地使用 Kibana 跟踪和调试请求。每个日志条目中都包含 Trace ID，可以用于过滤和搜索特定日志。

要搜索具有特定 Trace ID 的日志，可以使用 Kibana Query Language (KQL)。例如，您可以使用以下查询来搜索与特定 Trace ID 相关的所有日志：

```
trace_id:"Lc6t"
```

此查询将返回所有包含 Trace ID “Lc6t”的日志条目，使您能够追踪请求在系统中的路径。此外，您还可以将此查询与其他条件结合使用，以缩小搜索结果范围，例如按日志级别、时间戳或日志消息中的特定关键字进行过滤。

通过利用 Kibana 的可视化功能，您还可以创建基于 Trace ID 的仪表板，显示处理的请求数量、平均响应时间和错误率等指标和趋势。这有助于识别应用程序性能和可靠性中的模式和潜在问题。

将 Kibana 与我们的 Trace ID 解决方案结合使用，提供了一种全面的方法来监控、调试和分析系统的行为，确保每个请求都能得到有效跟踪和调查。

前端

api.js

```
const BASE_URL = process.env.REACT_APP_BASE_URL;

// Function to get client information
const getClientInfo = () => {
    const { language, platform, cookieEnabled, doNotTrack, onLine } = navigator;
    const { width, height } = window.screen;
    const connection = navigator.connection || navigator.mozConnection || navigator.webkitConnection;
    const networkType = connection ? connection.effectiveType : 'unknown';
    const timeZone = Intl.DateTimeFormat().resolvedOptions().timeZone;
    const referrer = document.referrer;
    const viewportWidth = window.innerWidth;
    const viewportHeight = window.innerHeight;

    return {
        screenWidth: width,
        screenHeight: height,
        networkType,
        timeZone,
        language,
        platform,
        cookieEnabled,
```

```
        doNotTrack,
        onLine,
        referrer,
        viewportWidth,
        viewportHeight
    };
};

// Function to generate a unique trace ID
export const generateTraceId = (length = 4) => {
    const characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789';
    let traceId = '';
    for (let i = 0; i < length; i++) {
        const randomIndex = Math.floor(Math.random() * characters.length);
        traceId += characters.charAt(randomIndex);
    }
    return traceId;
};

export const apiFetch = async (endpoint, options = {}) => {
    const url = `${BASE_URL}${endpoint}`;
    const clientInfo = getClientInfo();

    const traceId = options.traceId || generateTraceId();

    const headers = {
        'Content-Type': 'application/json',
        'X-Client-Info': JSON.stringify(clientInfo),
        'X-Trace-Id': traceId,
        ...(options.headers || {})
    };

    const response = await fetch(url, {
        ...options,
        headers
    });
```

```
    return response;
};
```

App.js

```
try {
  const response = await apiFetch('api', {
    method: 'POST',
    headers: {
      'Content-Type': 'application/json',
    },
    body: JSON.stringify(content),
    traceId: traceId
  });

  if (response.ok) {
    const data = await response.json();
    //...
  } else {
    const errorData = await response.json();
    const errorMessage = errorData.message || '发生未知错误';
    let errorToastMessage = errorMessage;
    errorToastMessage += ` (代号: ${traceId})`;
    toast.error(errorToastMessage, {
      autoClose: 8000
    });
    setError(errorToastMessage);
  }
} catch (error) {
  let errorString = error instanceof Error ? error.message : JSON.stringify(error);

  const duration = (Date.now() - startTime) / 1000;

  if (error.response) {
    // The request was made and the server responded with a status code that falls out of the range of
    errorString += ` (HTTP ${error.response.status}: ${error.response.statusText})`;
```

```
    console.error('响应错误数据:', error.response.data);
  } else if (error.request) {
    // The request was made but no response was received
    errorString += ' (未收到响应)';
    console.error('请求错误数据:', error.request);
  } else {
    // Something happened in setting up the request that triggered an Error
    errorString += ` (设置请求时出错: ${error.message})`;
  }

  errorString += ` (代号: ${traceId})`;

  if (error instanceof Error) {
    errorString += `\n堆栈: ${error.stack}`;
  }

  errorString += JSON.stringify(error);

  errorString += ` (耗时: ${duration} 秒)`;

  toast.error(`出错: ${errorString}`, {
    autoClose: 8000
  });
  setError(errorString);
} finally {
  toast.dismiss(toastId);
}
```

后端

__init__.py

```
# -*- encoding: utf-8 -*-

import os
import json
import time
```

```
import uuid
import string
import random

from flask import Flask, request, Response, g, has_request_context
from flask_cors import CORS

from .routes import initialize_routes
from .models import db, insert_default_config
import logging
from logging.handlers import RotatingFileHandler
from prometheus_client import Counter, generate_latest, Gauge
from flask_migrate import Migrate
from logstash_formatter import LogstashFormatterV1

app = Flask(__name__)

app.config.from_object('api.config.BaseConfig')

db.init_app(app)
initialize_routes(app)

CORS(app)

migrate = Migrate(app, db)


class RequestFormatter(logging.Formatter):
    def format(self, record):
        if has_request_context():
            record.trace_id = getattr(g, 'trace_id', 'unknown')
        else:
            record.trace_id = 'unknown'
        return super().format(record)
```

```
class CustomLogstashFormatter(LogstashFormatterV1):
    def format(self, record):
        if has_request_context():
            record.trace_id = getattr(g, 'trace_id', 'unknown')
        else:
            record.trace_id = 'unknown'
        return super().format(record)


def setup_loggers():
    logstash_handler = RotatingFileHandler(
        'app.log', maxBytes=100000000, backupCount=1)
    logstash_handler.setLevel(logging.DEBUG)
    logstash_formatter = CustomLogstashFormatter()
    logstash_handler.setFormatter(logstash_formatter)

    txt_handler = RotatingFileHandler(
        'plain.log', maxBytes=100000000, backupCount=1)
    txt_handler.setLevel(logging.DEBUG)
    txt_formatter = RequestFormatter(
        '%(asctime)s %(levelname)s: %(message)s [in %(pathname)s:%(lineno)d] [trace_id: %(trace_id)s]')
    txt_handler.setFormatter(txt_formatter)

    root_logger = logging.getLogger()
    root_logger.setLevel(logging.DEBUG)
    root_logger.addHandler(logstash_handler)
    root_logger.addHandler(txt_handler)

    app.logger.addHandler(logstash_handler)
    app.logger.addHandler(txt_handler)

    werkzeug_logger = logging.getLogger('werkzeug')
    werkzeug_logger.setLevel(logging.DEBUG)
    werkzeug_logger.addHandler(logstash_handler)
    werkzeug_logger.addHandler(txt_handler)
```

```
setup_loggers()


def generate_trace_id(length=4):
    characters = string.ascii_letters + string.digits
    return ''.join(random.choice(characters) for _ in range(length))


@app.before_request
def before_request():
    request.start_time = time.time()
    trace_id = request.headers.get('X-Trace-Id', generate_trace_id())
    g.trace_id = trace_id

    client_info = request.headers.get('X-Client-Info')
    if client_info:
        try:
            client_info_json = json.loads(client_info)
            logging.info(f"Client Info: {client_info_json}")
        except json.JSONDecodeError:
            logging.warning("Invalid JSON format for X-Client-Info header")


@app.after_request
def after_request(response):
    response.headers['X-Trace-Id'] = g.trace_id

    if response.status_code != 200:
        logging.error(f'Response status code: {response.status_code}')
        logging.error(f'Response body: {response.get_data(as_text=True)}')

        if response.content_type == 'application/json':
            try:
                response_json = response.get_json()
                response_json['trace_id'] = g.trace_id
```

```
                response.set_data(json.dumps(response_json))
            except Exception as e:
                logging.error(f"Error adding trace_id to response: {e}")

    return response
```

日志

要搜索与特定 Trace ID 相关的所有日志，可以使用以下查询：

```
trace_id:"Lc6t"
```


```
{
  "_index": "flask-logs-2024.07.05",
  "_type": "_doc",
  "_id": "Ae9zgZABqOMSOpxCZC5X",
  "_version": 1,
  "_score": 1,
  "_source": {
    "tags": [
      "_grokparsefailure"
    ],
    "filename": "generate.py",
    "funcName": "post",
    "message": "Request processed successfully",
    "@version": 1,
    "name": "root",
    "host": "ip-172-31-35-xxx.ec2.internal",
    "relativeCreated": 685817.8744316101,
    "levelname": "INFO",
    "created": 1720158740.894831,
    "thread": 139715118360128,
    "threadName": "Thread-5",
    "levelno": 20,
    "pathname": "/home/project/project-name/api/routes/generate.py",
    "msecs": 894.8309421539307,
    "processName": "MainProcess",
    "lineno": 287,
```

```
    "path": "/home/project/project-name/app.log",
    "args": [],
    "source_host": "ip-172-31-35-xxx.ec2.internal",
    "module": "generate",
    "trace_id": "Lc6t",
    "stack_info": null,
    "process": 107613,
    "@timestamp": "2024-07-05T05:52:20.894Z"
  },
  "fields": {
    "levelname.keyword": [
      "INFO"
    ],
    "tags.keyword": [
      "_grokparsefailure"
    ],
    "relativeCreated": [
      685817.9
    ],
    "processName.keyword": [
      "MainProcess"
    ],
    "filename.keyword": [
      "generate.py"
    ],
    "funcName": [
      "post"
    ],
    "path": [
      "/home/project/project-name/app.log"
    ],
    "processName": [
      "MainProcess"
    ],
    "@version": [
      1
```

```
],
"host": [
  "ip-172-31-35-xxx.ec2.internal"
],
"msecs": [
  894.83093
],
"source_host.keyword": [
  "ip-172-31-35-xxx.ec2.internal"
],
"host.keyword": [
  "ip-172-31-35-xxx.ec2.internal"
],
"levelname": [
  "INFO"
],
"process": [
  107613
],
"threadName.keyword": [
  "Thread-5"
],
"trace_id": [
  "Lc6t"
],
"source_host": [
  "ip-172-31-35-xxx.ec2.internal"
],
"created": [
  1720158700
],
"module": [
  "generate"
],
"module.keyword": [
  "generate"
```

```
],
"name.keyword": [
  "root"
],
"thread": [
  139715118360128
],
"message": [
  "Request processed successfully"
],
"levelno": [
  20
],
"trace_id.keyword": [
  "Lc6t"
],
"threadName": [
  "Thread-5"
],
"pathname": [
  "/home/project/project-name/api/routes/generate.py"
],
"tags": [
  "_grokparsefailure"
],
"pathname.keyword": [
  "/home/project/project-name/api/routes/generate.py"
],
"@timestamp": [
  "2024-07-05T05:52:20.894Z"
],
"filename": [
  "generate.py"
],
"lineno": [
  287
```

```
    ],
    "message.keyword": [
      "Request processed successfully"
    ],
    "name": [
      "root"
    ],
    "funcName.keyword": [
      "post"
    ],
    "path.keyword": [
      "/home/project/project-name/app.log"
    ]
  }
}
```

如上所示，您可以在日志中看到 Trace ID。