集中日志处理（logging\gunicorn\sentry\elk）

一、logging 模块

logging模块是Python内置的标准模块，主要用于输出运行日志，可以设置输出日志的等级、日志保存路径、日志文件回滚等；相比print，具备如下优点：

1. 可以通过设置不同的日志等级，在release版本中只输出重要信息，而不必显示大量的调试信息；

2. print将所有信息都输出到标准输出中，严重影响开发者从标准输出中查看其它数据；logging则可以由开发者决定将信息输出到什么地方，以及怎么输出；

核心概念

```
1.Logger      记录器：暴露了应用程序代码能直接使用的接口。
2.Handler     处理器：将（记录器产生的）日志记录发送至合适的目的地。
3.Filter      过滤器：提供了更好的粒度控制，它可以决定输出哪些日志记录。
4.Formatter   格式化器：指明了最终输出中日志记录的布局。
```

工作流程

- 1）（在用户代码中进行）日志记录函数调用，如：logger.info(...)，logger.debug(...)等；
- 2）判断要记录的日志级别是否满足日志器设置的级别要求（要记录的日志级别要大于或等于日志器设置的级别才算满足要求），如果不满足则该日志记录会被丢弃并终止后续的操作，如果满足则继续下一步操作；
- 3）根据日志记录函数调用时掺入的参数，创建一个日志记录（LogRecord类）对象；
- 4）判断日志记录器上设置的过滤器是否拒绝这条日志记录，如果日志记录器上的某个过滤器拒绝，则该日志记录会被丢弃并终止后续的操作，如果日志记录器上设置的过滤器不拒绝这条日志记录或者日志记录器上没有设置过滤器则继续下一步操作--将日志记录分别交给该日志器上添加的各个处理器；
- 5）判断要记录的日志级别是否满足处理器设置的级别要求（要记录的日志级别要大于或等于该处理器设置的日志级别才算满足要求），如果不满足记录将会被该处理器丢弃并终止后续的操作，如果满足则继续下一步操作；
- 6）判断该处理器上设置的过滤器是否拒绝这条日志记录，如果该处理器上的某个过滤器拒绝，则该日志记录会被当前处理器丢弃并终止后续的操作，如果当前处理器上设置的过滤器不拒绝这条日志记录或当前处理器上没有设置过滤器测继续下一步操作；
- 7）如果能到这一步，说明这条日志记录经过了层层关卡允许被输出了，此时当前处理器会根据自身被设置的格式器（如果没有设置则使用默认格式）将这条日志记录进行格式化，最后将格式化后的结果输出到指定位置（文件、网络、类文件的Stream等）；
- 8）如果日志器被设置了多个处理器的话，上面的第5-8步会执行多次；

- 9）这里才是完整流程的最后一步：判断该日志器输出的日志消息是否需要传递给上一级logger（之前提到过，日志器是有层级关系的）的处理器，如果propagate属性值为1则表示日志消息将会被输出到处理器指定的位置，同时还会被传递给parent日志器的handlers进行处理直到当前日志器的propagate属性为0停止，如果propagate值为0则表示不向parent日志器的handlers传递该消息，到此结束。

可见，一条日志信息要想被最终输出需要依次经过以下几次过滤：

- 日志器等级过滤；
- 日志器的过滤器过滤；
- 日志器的处理器等级过滤；
- 日志器的处理器的过滤器过滤；
- ```
  # services/bs-whatweb/gunicorn_logging.conf
  [loggers]
  keys = root

  [handlers]
  keys = access

  [formatters]
  keys = generic

  [handler_access]
  class = StreamHandler
  formatter = generic
  args = (sys.stdout,)

  [formatter_generic]
  format = [whatweb][%(levelname)s] [%(name)s]: %(message)s
  ```

二、gunicorn

1. 简单介绍

   Gunicorn“绿色独角兽”是一个被广泛使用的高性能的Python WSGI UNIX HTTP服务器，移植自Ruby的独角兽（Unicorn ）项目,使用pre-fork worker模式，具有使用非常简单，轻量级的资源消耗，以及高性能等特点。

2. 特点

   - 采用epoll (Linux下) 非阻塞网络I/O 模型
  ```

- 自动化worker进程管理
- 简单的Python配置
- 多种Worker类型可以选择 同步的，基于事件的（gevent tornado等），基于多线程的
- 各种服务器钩子，扩展性极强
- 支持 Python 2.x >= 2.6 or Python 3.x >= 3.2

服务模型(Server Model)

Gunicorn是基于 pre-fork 模型的。也就意味着有一个中心管理进程(master process)用来管理 worker 进程集合。Master从不知道任何关于客户端的信息。所有的请求和响应处理都是由 worker 进程来处理的。

Master(管理者)

主程序是一个简单的循环,监听各种信号以及相应的响应进程。master管理着正在运行的worker集合,通过监听各种信号比如TTIN, TTOU, and CHLD. TTIN and TTOU 响应的增加和减少worker的数目。CHLD信号表明一个子进程已经结束了,在这种情况下master会自动的重启失败的worker

```
# services/bs-whatweb/gunicorn_config.py
address = "0.0.0.0"
port = "5000"
bind = "{0}:{1}".format(address, port)  # 监听地址及端口
backlog = 2048  # 服务器在 pending 状态的最大连接数
workers = 2  # worker 进程数量
worker_class = 'gevent'  # worker进程的工作方式, sync, eventlet, gevent, tornado, gthread, 缺省值sync
worker_connections = 1000  # 客户端最大同时连接数, 只适用于 eventlet, gevent工作方式
timeout = 30  # 最大连接时间
keepalive = 2  # server 端保持连接时间
errorlog = '-'  # 错误日志路径
loglevel = 'error'  # 日志级别, debug, info, warning, error, critical
accesslog = '-'  # 访问日志路径
access_log_format = '{"request_address": "%(h)s", ' \
                    '"request_time": "%(t)s", ' \
                    '"request": "%(r)s", ' \
                    '"http_status_code": "%(s)s", ' \
                    '"http_request_url": "%(U)s", ' \
                    '"http_query_string": "%(q)s", ' \
                    '"request_headers": {' \
                    '"content-type": "%({content-type}i)s", ' \
                    '"content-length": "%({content-length}i)s", ' \
```

```
                    '"user-agent": "%(a)s"' \
                    '}}'
```

```
# services/bs-whatweb/gunicorn_logging.conf
# Logging configuration

[loggers]
keys = root, gunicorn.access, gunicorn.error

[handlers]
keys = access, error

[formatters]
keys = json, generic

# Root logger
# The root logger sends messages to the console and to Sentry.
[logger_root]
handlers = error

# Gunicorn loggers
# Gunicorn logging is configured with two loggers: 'gunicorn.access' and 'gunicorn.error'.
# The access log is sent to stdout and the error log is sent to stderr, both without propagation.
# Only the critical logger has a handler to send messages to Sentry.

[logger_gunicorn.access]
handlers = access
qualname = gunicorn.access
level = INFO
propagate = 0

[logger_gunicorn.error]
handlers = error
qualname = gunicorn.error
level = ERROR
propagate = 0

# Handlers
[handler_access]
class = StreamHandler
formatter = json
args = (sys.stdout,)
```

```
[handler_error]
class = StreamHandler
formatter = json
args = (sys.stderr,)

[formatter_generic]
format = [bs-whatweb][%(levelname)s] [%(name)s]: %(message)s
[formatter_json]
class = project.api.utils.logger.JSONFormatter
```

三、ELK

1. 简单介绍

   [ELK]由Elasticsearch、Logstash和Kibana三部分组件组成；

   Elasticsearch是个开源分布式搜索引擎，它的特点有：分布式，零配置，自动发现，索引自动分片，索引副本机制，restful风格接口，多数据源，自动搜索负载等。

   Logstash是一个完全开源的工具，它可以对你的日志进行收集、分析，并将其存储供以后使用

   kibana 是一个开源和免费的工具，它可以为 Logstash 和 ElasticSearch 提供的日志分析友好的 Web 界面，可以帮助您汇总、分析和搜索重要数据日志

2. 工作流程

   将服务的日志全部发送到远程 syslog 服务器存储，然后 filebeat 进行读取，在需要收集日志的所有服务上部署logstash，logstash agent 用于监控并过滤收集日志，将过滤后的内容发送到Redis/KAFKA，然后logstash indexer将日志收集在一起交给全文搜索服务ElasticSearch，可以用ElasticSearch进行自定义搜索通过 Kibana 来结合自定义搜索进行页面展示。

   ![image-20191024180026108](file:///Users/zhangweijie/Library/Application%20Support/typora-user-images/image-20191024180026108.png?lastModify=1571911254)

```
bs-whatweb:
    image: bs-whatweb:v0.0.2
    build:
      context: ./services/bs-whatweb
      dockerfile: Dockerfile-staging
    volumes:
      - './services/bs-whatweb:/usr/src/app'
    restart: always
    ######### 新增 ########
    logging:
      driver: "syslog"
      options:
        syslog-address: "tcp://192.168.199.142:514"
```

```
    tag: whatweb-server-staging
########################
ports:
  - 5007:5000
env_file:
  - bs-whatweb-staging.env
networks:
  - defnet
```

四、Sentry

1. 基本介绍

   Sentry 是一个现代化的错误日志记录以及聚合平台。支持几乎所有的主流开发语言及平台，并提供现代化 UI。

   ![image-20191017165847184](/Users/zhangweijie/Library/Application Support/typora-user-images/image-20191017165847184.png)

   场景介绍：

   接入 Sentry 前；

   - 用户A: 发布功能用不了
   - 开发者A: 哪个页面? 截个图
   - 用户A: (发截图)
   - 开发者A发现bug可以重现, 登录服务器查看错误日志, 确认程序逻辑无问题, 查看数据库数据, 发现有脏数据. 联系开发者B检查负责更新数据的python脚本C.py.
   - 开发者B登录服务器查看错误日志, 发现一个逻辑错误导致脚本罢工, 已持续了一个小时. 影响了数千条数据

   接入 Sentry 后：

   - 开发者A,B同时收到邮件告警, 一分钟前脚本C.py异常退出.
   - 开发者B进入sentry后台查看错误信息, 定位问题并将其修复, 再清理受影响的数十条数据.
   - 在此过程中没有用户受到影响, 无需开发者A介入

2. 基本概念

   - event

     可操作数据的基本单位，每一次日志输出就会产生一个 event。event并不一定是错误，如果日志级别设置的很低，那么后台就会产生非常多的 event，所以正确的设置日志的级别 非常的重要。

- issue

  同一类 event 的聚合，某一个错误可能因为重复执行而被记录多次，在 sentry系统会自动聚合到一起，方便处理，通常我们操作的对象也就是 issue。

- DNS

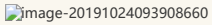
  DNS 及时客户端密匙，用来进行客户端和服务端的通信。DNS 是一个 URL，包含一个公匙，一个私匙，项目标记及服务器地址

- raven

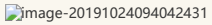
  整个错误日志监控系统包含客户端和服务端，Sentry 是服务端的名称，客户端名称为 Raven，需要两者配合才能工作

3. 项目流程

- 登录 Sentry 系统



- 点击创建项目



- 选择框架

- 根据指示下载安装包并进行配置

在 flask 中的配置有两种方式，可以自由选择

第一种：

```
from raven.contrib.flask import Sentry
sentry = Sentry()
sentry.init_app(app,dsn="",logging=True, level=logging.ERROR)
```

第二种：

```
import sentry_sdk
from sentry_sdk.integrations.flask import FlaskIntegration
sentry_sdk.init(dsn="",
integrations=[FlaskIntegration()])
```

gunicorn 与 Sentry 结合配置

使用 yaml 文件进行 logging 配置

```
# Logging configuration

[loggers]
keys = root, gunicorn.access, gunicorn.error

[handlers]
keys = access, error, sentry

[formatters]
keys = json, generic

# Root logger
# The root logger sends messages to the console and to Sentry.
[logger_root]
handlers = error, sentry

# Gunicorn loggers
# Gunicorn logging is configured with two loggers: 'gunicorn.access' and 'gunicorn.error'.
# The access log is sent to stdout and the error log is sent to stderr, both without propagation.
# Only the critical logger has a handler to send messages to Sentry.

[logger_gunicorn.access]
handlers = access
qualname = gunicorn.access
level = INFO
propagate = 0

[logger_gunicorn.error]
handlers = error, sentry
qualname = gunicorn.error
level = ERROR
propagate = 0

[handler_access]
class = StreamHandler
formatter = json
args = (sys.stdout,)

[handler_error]
```

```
class = StreamHandler
formatter = json
args = (sys.stderr,)

[handler_sentry]
class = raven.handlers.logging.SentryHandler
level = CRITICAL
formatter = generic
args = ("",)

[formatter_generic]
format = [bs-whatweb][%(levelname)s] [%(name)s]: %(message)s
[formatter_json]
class = project.api.utils.logger.JSONFormatter
```