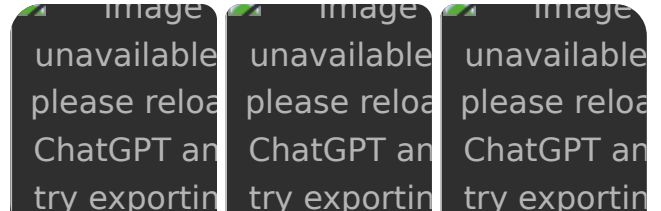


程式碼格式化請求



以codeblock格式給我這3張圖片的程式碼內容



以下是你提供的三張圖片中的程式碼內容，已整理為 `codeblock` 格式：

第1張圖（`collect` 方法）

```
def collect(self) -> Iterable[Metric]:
    # 確保 Scraper 在 `metric` 更新週期內只能抓取一次
    scraper_id = self.scraper_id
    scraper_ip = self.scraper_ip
    # 取得 IP + User-Agent
    scraper_version = f"{scraper_ip}_{scraper_id}"

    with self.cache_lock:
        # 如果 Scraper 已經抓取過這個 `metric` 週期，則拒絕
        if scraper_version in self.scraper_access_record:
            logger.warning(
                f"Scraper {scraper_version} "
                "already accessed metrics in this cycle."
            )
        return
```

```

        # 記錄 Scraper 這次抓取的時間
        self.scrapers_access_record[scraper_version] = time.time()

    metric = GaugeMetricFamily(
        "log_host_job_count",
        "Count of occurrences of host and job_name in log",
        labels=["host", "job_name"]
    )
    for (host, job_name), count in self.metric_cache.items():
        metric.add_metric([host, job_name], count)

    yield metric # 返回 metric 指標

```

第2張圖（`update_metrics` 與 `_count_host_job` 方法）

```

def update_metrics(self) -> None:
    # 從最新的 tmp_log_<timestamp>.csv 更新 metric
    if not self.tmp_log_file or not os.path.exists(self.tmp_log_file):
        logger.warning(
            f"Temporary log file {self.tmp_log_file} does not exist."
        )
        return

    counts = self._count_host_job(self.tmp_log_file)

    with self.cache_lock:
        self.metric_cache = counts
        self.update_timestamp = time.time()
        # 重設 Scraper 記錄, 允許 Scraper 再次抓取
        self.scrapers_access_record.clear()

    logger.info("Metrics updated successfully.")

def _count_host_job(self, tmp_log_file: str) -> Dict[Tuple[str, str], int]:
    # 計算 data_collect_tmp.csv 中 host 和 job_name 的出現次數
    counts: Dict[Tuple[str, str], int] = {}

```

```

try:
    with open(tmp_log_file, 'r', encoding='utf-8') as temp_file:
        reader = csv.reader(temp_file)
        for row in reader:
            host, job_name = row[0], row[1]
            key = (host, job_name)
            counts[key] = counts.get(key, 0) + 1
except Exception as read_error:
    logger.error(f"Error reading file {tmp_log_file}: {read_error}")

return counts

```

第3張圖（初始化 metric 並更新）

```

# **Initialize metric**
counts, labels_list = parse_csv()
logging.info(f"Set Prometheus metric, Labels: {labels_list}") # Debug
log_host_job_count = Gauge("log_host_job_count", "Count of occurrences of host
and job_name in log", labels_list)

def update_metrics():
    """Update Prometheus metric by reading data from CSV"""
    log_host_job_count.clear() # **Clear old data**
    counts, labels_list = parse_csv()

    logging.info("\n[DEBUG] Updating metrics:")
    for key, count in counts.items():
        labels_dict = dict(key) # **Convert key back to dict**
        # Ensure `labels_dict` only contains the labels from the CSV**
        labels_dict = {key: labels_dict.get(key, "") for key in labels_list}

        logging.info(f"[DEBUG] Set `metrics` -> {labels_dict} : {count}") #
Debug
        log_host_job_count.labels(**labels_dict).set(count)

```

如需進一步說明此程式架構與 Prometheus 指標如何整合，歡迎告訴我！



以CodeBlock格式給我這3段程式碼的技術文件



當然可以，以下是針對你提供的三段程式碼所撰寫的**技術文件**，已用 ``codeblock`` 格式包裝，並且清楚標明各段功能與用途：

技術文件：``collect(self) -> Iterable[Metric]``

函數名稱

`collect(self) -> Iterable[Metric]`

功能說明

此函數由 Prometheus Exporter 呼叫，用來產生自定義 metric。

為了避免重複統計，同一個 Scraper（由 IP + ID 判斷）在一個更新週期內只能抓取一次。

參數

無（self 為實例物件）

回傳

一個 ``Metric`` 物件（由 Prometheus client library 的 ``GaugeMetricFamily`` 建立）

主要流程

1. 組合 ``scraper_version = f"{scraper_ip}_{scraper_id}"`` 作為唯一識別。
2. 使用 ``self.cache_lock`` 鎖定區塊，避免多個 scraper 同時操作 cache。
3. 若 ``scraper_version`` 存在於 ``scraper_access_record``，表示已經抓過資料，則跳過。
4. 若未抓過，紀錄此次抓取時間。
5. 建立 ``GaugeMetricFamily``，並從 ``self.metric_cache`` 中加入對應的 ``(host, job_name)`` 與數值。
6. 最後使用 ``yield metric`` 回傳資料。

相關欄位

- `self.scrapers_id`: 使用者或 Agent 的 ID。
- `self.scrapers_ip`: 使用者的 IP。
- `self.metric_cache`: 暫存計算好的 metrics。
- `self.scrapers_access_record`: 紀錄哪個 scraper 已經抓過資料。

技術文件：`update_metrics(self) -> None` & `_count_host_job(...)`

函數名稱

`update_metrics(self) -> None`

功能說明

從暫存的 CSV 檔案中讀取最新資料，計算每組 `(host, job_name)` 的出現次數，並更新內部 metric cache。

這樣 Prometheus 抓取時才能取得最新資料。

使用情境

通常配合定時工作（如 cron job 或 CI/CD）寫入 tmp_log 檔後使用。

程式流程

1. 檢查 `self.tmp_log_file` 是否存在，若不存在則記錄 warning 並 return。
2. 呼叫 `_count_host_job` 讀取檔案並統計每組 `(host, job_name)` 的出現次數。
3. 用 `self.cache_lock` 鎖住資料結構，更新 `self.metric_cache`。
4. 清空 `self.scrapers_access_record` 以允許 Scraper 再次抓取。

函數名稱

`_count_host_job(self, tmp_log_file: str) -> Dict[Tuple[str, str], int]`

功能說明

讀取指定的 CSV 檔案，統計每一組 `(host, job_name)` 的出現次數。

傳入參數

- ``tmp_log_file``: 要讀取的 CSV 檔案路徑

回傳

- ``Dict[Tuple[str, str], int]``: 每組 (host, job_name) 的出現次數統計

備註

若讀檔過程有錯誤，會記錄 ``logger.error`` 並回傳空 dict。

技術文件：Prometheus `Gauge` 初始化與 CSV 資料更新

區塊功能

初始化 Prometheus 的 ``Gauge`` 指標，並根據 CSV 資料更新指標值。

初始化流程

1. 呼叫 ``parse_csv()`` 取得：

- ``counts``: 每組 key (如 ``(host, job_name)``) 的統計數量
- ``labels_list``: 標籤欄位名稱 (如 ``["host", "job_name"]``)

2. 建立 ``Gauge``：

```
```python
Gauge("log_host_job_count", "Count of occurrences of host and job_name in
log", labels_list)
```

## 更新流程 ``update_metrics()``

1. 清除原有的指標資料：``log_host_job_count.clear()``
2. 再次解析最新的 CSV 取得統計資訊與標籤名稱。
3. 遍歷 ``counts.items()``：

- 將 `key`（如 tuple）轉回 dict 格式：`labels_dict`
- 確保 `labels_dict` 只包含 `labels_list` 中允許的欄位
- 呼叫 `log_host_job_count.labels(**labels_dict).set(count)` 寫入 Prometheus

---

## 說明

- 使用 `.labels().set()` 的方式可動態設定多組不同標籤的數據。
- 可搭配 Grafana 或 PromQL 使用 `log_host_job_count{host="...", job_name="..."}` 查詢。

## 適用場景

此類設計適合於由外部系統（如 log parser、pipeline）產出 `.csv` 統計後，提供 Prometheus 抓取的應用情境。

---

如需整合成 README 文件、API 文件，或補上範例 `.csv` 與 Prometheus 設定，也可以再告訴我，我可以幫你補上完整說明。





以CodeBlock格式給我這張圖片的程式碼內容&技術文件



以CodeBlock格式給我這張圖片的程式碼內容&技術文件





以下是你提供圖片中的 Python 程式碼內容與對應的技術文件，已使用 `codeblock` 格式整理。

## 程式碼內容

```
from prometheus_client import start_http_server, Gauge, REGISTRY
from prometheus_client.core import GaugeMetricFamily
import time

class CustomGauge:
 """
 Custom Gauge to allow dynamic label keys (omit empty labels).
 """

 def __init__(self, name, documentation):
 self.name = name
 self.documentation = documentation
 self.metrics = {}

 def set(self, labels, value):
 # Filter out labels with empty string values
 filtered_labels = {k: v for k, v in labels.items() if v}
 # Use a tuple of sorted (k,v) pairs as key for uniqueness
 key = tuple(sorted(filtered_labels.items()))
 self.metrics[key] = (filtered_labels, value)

 def collect(self):
 # Group metrics by label keys for correct GaugeMetricFamily grouping
 group = {}
 for label_tuple, (labels, value) in self.metrics.items():
 label_keys = tuple(labels.keys())
 if label_keys not in group:
 group[label_keys] = []
 group[label_keys].append((labels, value))

 for label_keys, series in group.items():
```

```

 gauge = GaugeMetricFamily(self.name, self.documentation,
labels=label_keys)
 for labels, value in series:
 gauge.add_metric([labels[k] for k in label_keys], value)
 yield gauge

Create a custom gauge instance
custom_gauge = CustomGauge('my_test_metric', 'test')

Register the collector
REGISTRY.register(custom_gauge)

Start HTTP server
start_http_server(8000)

while True:
 # Only non-empty labels will appear
 custom_gauge.set({'a': 'foo', 'b': '', 'c': 'bar'}, 1.0)
 custom_gauge.set({'a': 'abc', 'b': 'def', 'c': ''}, 2.0)
 time.sleep(10)

```

## 技術文件

# 技術文件：CustomGauge 實作與 Prometheus Exporter 整合

## 模組用途

此模組實作了一個自定義 Prometheus Gauge，支援：

- 動態 label key 數量與名稱
- 自動略過空值 label
- 多組 label 組合的收集與暴露

---

## 類別說明：CustomGauge

### \_\_init\_\_(self, name, documentation)

初始化 metric 名稱與說明。

- `name`: Prometheus metric 名稱 (如 `my\_test\_metric`)
- `documentation`: 說明文字

---

### set(self, labels: Dict[str, str], value: float)

新增一筆 metric 觀測值。

- 移除 label 中 value 為空字串的項目
- 將剩餘 labels 做 `sorted()` 並轉為 tuple 作為 key, 確保唯一性
- 儲存進內部字典: `self.metrics[key] = (labels, value)`

---

### collect(self)

為 Prometheus 收集並回傳 metrics 的方法:

1. 將 metric 根據 label keys 做 grouping, 因為 Prometheus 的 `GaugeMetricFamily` 要指定 label 名稱。
2. 建立對應的 `GaugeMetricFamily` 並將相同 label keys 的值加進去。
3. 用 `yield` 回傳每組 Gauge。

---

## 執行流程

### 初始化

```python

```
custom_gauge = CustomGauge('my_test_metric', 'test')
REGISTRY.register(custom_gauge)
start_http_server(8000)
```

執行時定期更新

```
while True:
    custom_gauge.set({'a': 'foo', 'b': '', 'c': 'bar'}, 1.0)
    custom_gauge.set({'a': 'abc', 'b': 'def', 'c': ''}, 2.0)
    time.sleep(10)
```

“僅有非空的 *label* 會進入資料中。”

HTTP 暴露介面

啟動在 `localhost:8000`，可以讓 Prometheus 使用 `/metrics` 抓取資料。

適用情境

- CSV 或 JSON 資料來源欄位動態
- 指標 *label* 組合可能不固定
- 不想暴露空值的 *label*

如需改寫為類似 Prometheus 官方 Collector 用法，或整合成 class-based Exporter 結構，也可以協助提供範例。



我想要理解這段程式碼用法，為了加到我的exporter中做實驗時，為什麼這樣可以不顯示空值的labels，我對這些code不理解，你能教我嗎，或是你能給更簡單的範例：

"""

```
from prometheus_client import start_http_server, Gauge,
REGISTRY
from prometheus_client.core import GaugeMetricFamily
import time
```

```
class CustomGauge:
```

```

"""
    Custom Gauge to allow dynamic label keys (omit
    empty labels).
"""

def __init__(self, name, documentation):
    self.name = name
    self.documentation = documentation
    self.metrics = {}

def set(self, labels, value):
    # Filter out labels with empty string values
    filtered_labels = {k: v for k, v in labels.items() if v}
    # Use a tuple of sorted (k,v) pairs as key for
    uniqueness
    key = tuple(sorted(filtered_labels.items()))
    self.metrics[key] = (filtered_labels, value)

def collect(self):
    # Group metrics by label keys for correct
    GaugeMetricFamily grouping
    group = {}
    for label_tuple, (labels, value) in
self.metrics.items():
        label_keys = tuple(labels.keys())
        if label_keys not in group:
            group[label_keys] = []
        group[label_keys].append((labels, value))

    for label_keys, series in group.items():
        gauge = GaugeMetricFamily(self.name,
self.documentation, labels=label_keys)
        for labels, value in series:
            gauge.add_metric([labels[k] for k in
label_keys], value)
        yield gauge

# Create a custom gauge instance
custom_gauge = CustomGauge('my_test_metric', 'test')

# Register the collector

```

```
REGISTRY.register(custom_gauge)

# Start HTTP server
start_http_server(8000)

while True:
    # Only non-empty labels will appear
    custom_gauge.set({'a': 'foo', 'b': '', 'c': 'bar'}, 1.0)
    custom_gauge.set({'a': 'abc', 'b': 'def', 'c': ''}, 2.0)
    time.sleep(10) """
```



非常好的問題，我會分三個部分幫你理解這段程式碼，並補上一個**更簡化的範例**，讓你更容易加到你的 exporter 裡做實驗。

為什麼這段程式碼可以「不顯示空值的 labels」？

原因在這段：

```
filtered_labels = {k: v for k, v in labels.items() if v}
```

這行是 dictionary comprehension，意思是：

- 把傳進來的 labels 中「**value 不為空字串** (``)`」的留下來。
- 只保留有值的 labels。

所以當你執行：

```
custom_gauge.set({'a': 'foo', 'b': '', 'c': 'bar'}, 1.0)
```

會被轉成：

```
{'a': 'foo', 'c': 'bar'}
```

接著 `GaugeMetricFamily` 被建立時，只會看到 `a` 跟 `c`，`b` 完全消失。Prometheus 就不會看到空 label 的指標！

🧠 為什麼還要 group？為什麼不能直接 yield？

Prometheus 的 `GaugeMetricFamily` 要你「同一組 label 名稱要丟在一起」。

舉例：

```
gauge = GaugeMetricFamily('metric_name', 'desc', labels=['a', 'c'])
gauge.add_metric(['foo', 'bar'], 1.0)
```

這就代表：

```
metric_name{a="foo", c="bar"} 1.0
```

但如果下一筆是：

```
{'a': 'abc', 'b': 'def'}
```

那這筆的 label 名稱是 `['a', 'b']`，和上面那筆不一樣，不能丟在同一個 `GaugeMetricFamily` 物件裡。這就是為什麼 `collect()` 裡要先依照 `label_keys` 做分組。

✅ 更簡單版本的範例

這是一個簡化版 **CustomGauge**，功能一樣是「忽略空 label」，但寫法更精簡，更適合你做實驗：

```
from prometheus_client import start_http_server, REGISTRY
from prometheus_client.core import GaugeMetricFamily
import time

class SimpleGauge:
    def __init__(self, name, doc):
        self.name = name
        self.doc = doc
        self.data = [] # 儲存 (labels, value)

    def set(self, labels: dict, value: float):
        # 移除空字串的 label
        filtered = {k: v for k, v in labels.items() if v}
        self.data.append((filtered, value))

    def collect(self):
        groups = {}
        for labels, value in self.data:
            key = tuple(sorted(labels.keys()))
            if key not in groups:
                groups[key] = []
            groups[key].append((labels, value))

        for keys, series in groups.items():
            gauge = GaugeMetricFamily(self.name, self.doc, labels=list(keys))
            for labels, value in series:
                gauge.add_metric([labels[k] for k in keys], value)
            yield gauge

# 建立實例並註冊
g = SimpleGauge("demo_metric", "Show only non-empty labels")
REGISTRY.register(g)

start_http_server(8000)

while True:
    g.set({"host": "srv1", "zone": ""}, 5) # zone 被忽略
```



```
g.set({"host": "srv2", "zone": "us"}, 8)
time.sleep(10)
```

“到 `http://localhost:8000/metrics` 你只會看到：

```
demo_metric{host="srv1"} 5.0
demo_metric{host="srv2", zone="us"} 8.0
```

建議你實驗時可以這樣進行：

1. 先把簡化版 `SimpleGauge` 放到 exporter 裡。
2. 嘗試用 `.set()` 加入不同 label 組合（有的空、有的不空）。
3. 查看 `/metrics`，觀察 Prometheus 如何只暴露有效的 label。
4. 再進階使用你原本圖片裡那段進階版 `CustomGauge`。

如果你想要加進你原本那個 exporter class 架構中，我也可以幫你整合 `CustomGauge` 成 class 成員，讓你更方便實驗。是否需要我幫你整合進你的 exporter 架構？



任務

依照“任務說明”教會我並給我範例程式碼

任務說明

1. 將我給你的程式碼好好利用
2. 說明並讓我理解整合的方法
3. 給我範例程式碼
4. ‘程式碼一’要怎麼整合進‘程式碼二’呢

```

# 數據庫
## 程式碼1
"""

from prometheus_client import start_http_server, Gauge,
REGISTRY
from prometheus_client.core import GaugeMetricFamily
import time

class CustomGauge:
    """
    Custom Gauge to allow dynamic label keys (omit
    empty labels).
    """

    def __init__(self, name, documentation):
        self.name = name
        self.documentation = documentation
        self.metrics = {}

    def set(self, labels, value):
        # Filter out labels with empty string values
        filtered_labels = {k: v for k, v in labels.items() if v}
        # Use a tuple of sorted (k,v) pairs as key for
        uniqueness
        key = tuple(sorted(filtered_labels.items()))
        self.metrics[key] = (filtered_labels, value)

    def collect(self):
        # Group metrics by label keys for correct
        GaugeMetricFamily grouping
        group = {}
        for label_tuple, (labels, value) in
self.metrics.items():
            label_keys = tuple(labels.keys())
            if label_keys not in group:
                group[label_keys] = []
            group[label_keys].append((labels, value))

        for label_keys, series in group.items():
            gauge = GaugeMetricFamily(self.name,

```

```

self.documentation, labels=label_keys)
    for labels, value in series:
        gauge.add_metric([labels[k] for k in
label_keys], value)
    yield gauge

# Create a custom gauge instance
custom_gauge = CustomGauge('my_test_metric', 'test')

# Register the collector
REGISTRY.register(custom_gauge)

# Start HTTP server
start_http_server(8000)

while True:
    # Only non-empty labels will appear
    custom_gauge.set({'a': 'foo', 'b': '', 'c': 'bar'}, 1.0)
    custom_gauge.set({'a': 'abc', 'b': 'def', 'c': ''}, 2.0)
    time.sleep(10)
"""

## 程式碼2
"""

import csv
import os
import time
import logging
from prometheus_client import Gauge, start_http_server

# === [1] 設定 log 紀錄格式，方便我們觀察 metrics 的更新
狀況 ===
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(
message)s",
)

# === [2] 設定要讀取的 CSV 檔案 ===
CSV_FILE = "bak-data_collect.csv"

```

```

# === [3] 初始化 Prometheus 的 Gauge（我們稍後會根據
labels 來建立） ===
log_host_job_count = None # 先留空，等 labels 決定好再
建立

def parse_csv():
    """解析 CSV，動態收集所有出現過的 labels，並統計每組
    labels 的出現次數"""
    counts = {} # key = frozenset(labels), value = 次數
    dynamic_labels = {"host", "job_name"} # 初始固定兩
    個 labels

    if not os.path.exists(CSV_FILE):
        logging.error(f"CSV 檔案 {CSV_FILE} 不存在！")
        return counts, sorted(list(dynamic_labels))

    with open(CSV_FILE, 'r', encoding='utf-8') as f:
        reader = csv.reader(f)
        for row in reader:
            if len(row) < 2:
                continue # 至少要有 host 和 job_name

            host = row[0].strip()
            job_name = row[1].strip()
            extra_labels = {} # 額外的 labels

            # === 處理第三欄之後的欄位：包含額外 labels
            ===
            for col in row[2:]:
                col = col.strip()

                if col.startswith("{") and col.endswith("}"):
                    col = col[1:-1].strip() # 拿掉大括號

                # 拆解 key=value
                key_value_pairs = col.split(",")
                for pair in key_value_pairs:
                    pair = pair.strip()
                    if "=" in pair:
                        key, value = map(str.strip, pair.split("=",
1))

```

```

        key = key.replace("{", "").replace("}", "",
    "").strip()
        value = value.replace("'", "").replace('"',
    "").strip("\\'")

    if key and value:
        extra_labels[key] = value # 收集有效
label

    # 更新動態 labels (只加入 key, 不重複)
    dynamic_labels.update(extra_labels.keys())

    # 組合完整的 labels
    full_label_dict = {**extra_labels, "host": host,
    "job_name": job_name}
    key = frozenset(full_label_dict.items())
    counts[key] = counts.get(key, 0) + 1

    logging.info(f"[parse_csv] 最終 labels:
    {sorted(dynamic_labels)}")
    return counts, sorted(dynamic_labels)

def update_metrics():
    """每次更新 metrics (Prometheus 會來抓), 我們就重建一
    次"""
    global log_host_job_count
    counts, labels_list = parse_csv()

    # 初始化或重新初始化 Gauge
    log_host_job_count = Gauge("log_host_job_count",
    "Count of host/job_name with optional labels",
    labels_list)

    log_host_job_count._metrics.clear() # 清掉舊資料
    logging.info("[update_metrics] 開始設定 metrics...")

    for key, count in counts.items():
        labels_dict = dict(key)
        complete_labels = {k: labels_dict.get(k, "") for k in
    labels_list} # 確保所有 label key 都存在
        logging.info(f"[metric] {complete_labels} =>

```

```

{count}"))

log_host_job_count.labels(**complete_labels).set(count)

if __name__ == "__main__":
    start_http_server(8080)
    logging.info("Exporter 正在
http://localhost:8080/metrics 執行...")

    while True:
        update_metrics()
        time.sleep(10) # 每 10 秒更新一次
"""

## bak-data_collect.csv
"""
host_1,job_A, {service_name="aaa",
container_name="bbbb"}
host_1,job_A
host_1,job_B
host_2,job_A
host_2,job_C
host_3,job_B, {module_name="cbbb"}
host_3,job_B
host_3,job_B
"""

```



好的，讓我一步步教你怎麼完成你的任務，包含「說明整合方法」與「提供完整範例程式碼」，幫助你把「**程式碼1**」（自定義 CustomGauge）整合進「**程式碼2**」（CSV 分析並產生 metrics）。

✓ 任務說明整理

我們的目標是：

1. 用 `CustomGauge` 替代 `程式碼2` 中的 `Gauge(...)`；
2. 利用 `set()` 方法加入資料，並自動忽略空 label；
3. 保留原本每 10 秒更新 CSV 並重新統計的流程。

如何整合（步驟說明）

步驟 1：

將 `程式碼1` 的 `CustomGauge` 類別複製貼到你的 `程式碼2` 開頭，取代 `log_host_job_count = Gauge(...)` 的方式。

步驟 2：

初始化：

```
log_host_job_count = CustomGauge("log_host_job_count", "Count of host/job_name  
with optional labels")
```

步驟 3：

把 `Gauge(...).labels(...).set(...)` 改為：

```
log_host_job_count.set(labels_dict, count)
```

它會自動忽略空值，不需要再用 `.get(k, "")` 強塞空字串。

步驟 4：

把原本 `log_host_job_count._metrics.clear()` 改為：

```
log_host_job_count.metrics.clear()
```

(因為 CustomGauge 內部叫做 `.metrics`)

整合後的完整範例程式碼

```
import csv
import os
import time
import logging
from prometheus_client import start_http_server, REGISTRY
from prometheus_client.core import GaugeMetricFamily

# === 自定義 CustomGauge 類別 ===
class CustomGauge:
    """
    Custom Gauge to allow dynamic label keys (omit empty labels).
    """
    def __init__(self, name, documentation):
        self.name = name
        self.documentation = documentation
        self.metrics = {}

    def set(self, labels, value):
        filtered_labels = {k: v for k, v in labels.items() if v}
        key = tuple(sorted(filtered_labels.items()))
        self.metrics[key] = (filtered_labels, value)

    def collect(self):
        group = {}
        for label_tuple, (labels, value) in self.metrics.items():
            label_keys = tuple(labels.keys())
            if label_keys not in group:
                group[label_keys] = []
            group[label_keys].append((labels, value))
```



```

        for label_keys, series in group.items():
            gauge = GaugeMetricFamily(self.name, self.documentation,
labels=label_keys)
            for labels, value in series:
                gauge.add_metric([labels[k] for k in label_keys], value)
            yield gauge

# === Logging 設定 ===
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
)

# === 讀 CSV 並解析 labels ===
CSV_FILE = "bak-data_collect.csv"
log_host_job_count = CustomGauge("log_host_job_count", "Count of host/job_name
with optional labels")
REGISTRY.register(log_host_job_count)

def parse_csv():
    counts = {}
    if not os.path.exists(CSV_FILE):
        logging.error(f"CSV 檔案 `{CSV_FILE}` 不存在!")
        return counts

    with open(CSV_FILE, 'r', encoding='utf-8') as f:
        reader = csv.reader(f)
        for row in reader:
            if len(row) < 2:
                continue

            host = row[0].strip()
            job_name = row[1].strip()
            extra_labels = {}

            for col in row[2:]:
                col = col.strip()
                if col.startswith("{") and col.endswith("}"):
                    col = col[1:-1].strip()
                    for pair in col.split(","):

```

```

        if "=" in pair:
            k, v = map(str.strip, pair.split("=", 1))
            k = k.strip("{}").strip()
            v = v.replace("'", "").replace('"', "").strip("\\"")
            if k and v:
                extra_labels[k] = v

    full_label_dict = {**extra_labels, "host": host, "job_name":
job_name}

    key = frozenset(full_label_dict.items())
    counts[key] = counts.get(key, 0) + 1

    return counts

def update_metrics():
    log_host_job_count.metrics.clear()
    counts = parse_csv()
    for key, count in counts.items():
        labels_dict = dict(key)
        log_host_job_count.set(labels_dict, count)
        logging.info(f"[metric] {labels_dict} => {count}")

if __name__ == "__main__":
    start_http_server(8080)
    logging.info("Exporter 正在 http://localhost:8080/metrics 執行...")

    while True:
        update_metrics()
        time.sleep(10)

```

實驗效果

給定你提供的 `bak-data_collect.csv`：

```

host_1,job_A,{service_name="aaa", container_name="bbbb"}
host_1,job_A

```

```
host_1,job_B
host_2,job_A
host_2,job_C
host_3,job_B,{module_name="cbbb"}
host_3,job_B
host_3,job_B
```

會輸出如下 Prometheus 指標（只含有非空 label）：

```
log_host_job_count{host="host_1", job_name="job_A", service_name="aaa",
container_name="bbbb"} 1.0
log_host_job_count{host="host_1", job_name="job_A"} 1.0
log_host_job_count{host="host_1", job_name="job_B"} 1.0
log_host_job_count{host="host_2", job_name="job_A"} 1.0
log_host_job_count{host="host_2", job_name="job_C"} 1.0
log_host_job_count{host="host_3", job_name="job_B", module_name="cbbb"} 1.0
log_host_job_count{host="host_3", job_name="job_B"} 2.0
```

如果你想再進一步學會如何使用 label filtering、label 動態排序，或如何結合 `vmalert` 做 alerting，也可以繼續問我！是否需要我幫你加上 Grafana 顯示或 alert 寫法呢？



以CodeBlock格式，給我此段程式碼的技術文件

```
"""
```

```
import csv
import os
import time
import logging
from prometheus_client import start_http_server,
REGISTRY
from prometheus_client.core import GaugeMetricFamily

# === 自定義 CustomGauge 類別 ===
class CustomGauge:
```



```

"""
Custom Gauge to allow dynamic label keys (omit
empty labels).
"""

def __init__(self, name, documentation):
    self.name = name
    self.documentation = documentation
    self.metrics = {}

def set(self, labels, value):
    filtered_labels = {k: v for k, v in labels.items() if v}
    key = tuple(sorted(filtered_labels.items()))
    self.metrics[key] = (filtered_labels, value)

def collect(self):
    group = {}
    for label_tuple, (labels, value) in
self.metrics.items():
        label_keys = tuple(labels.keys())
        if label_keys not in group:
            group[label_keys] = []
        group[label_keys].append((labels, value))

    for label_keys, series in group.items():
        gauge = GaugeMetricFamily(self.name,
self.documentation, labels=label_keys)
        for labels, value in series:
            gauge.add_metric([labels[k] for k in
label_keys], value)
        yield gauge

# === Logging 設定 ===
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(
message)s",
)

# === 讀 CSV 並解析 labels ===
CSV_FILE = "bak-data_collect.csv"
log_host_job_count =

```

```

CustomGauge("log_host_job_count", "Count of
host/job_name with optional labels")
REGISTRY.register(log_host_job_count)

def parse_csv():
    counts = {}
    if not os.path.exists(CSV_FILE):
        logging.error(f"CSV 檔案 {CSV_FILE} 不存在！")
        return counts

    with open(CSV_FILE, 'r', encoding='utf-8') as f:
        reader = csv.reader(f)
        for row in reader:
            if len(row) < 2:
                continue

            host = row[0].strip()
            job_name = row[1].strip()
            extra_labels = {}

            for col in row[2:]:
                col = col.strip()
                if col.startswith("{") and col.endswith("}"):
                    col = col[1:-1].strip()
                    for pair in col.split(","):
                        if "=" in pair:
                            k, v = map(str.strip, pair.split("=", 1))
                            k = k.strip("{}").strip()
                            v = v.replace("'", "").replace('"',
                                "").strip("\\'")
                            if k and v:
                                extra_labels[k] = v

            full_label_dict = {**extra_labels, "host": host,
"job_name": job_name}
            key = frozenset(full_label_dict.items())
            counts[key] = counts.get(key, 0) + 1

        return counts

def update_metrics():

```

```

log_host_job_count.metrics.clear()
counts = parse_csv()
for key, count in counts.items():
    labels_dict = dict(key)
    log_host_job_count.set(labels_dict, count)
    logging.info(f"[metric] {labels_dict} => {count}")

if __name__ == "__main__":
    start_http_server(8080)
    logging.info("Exporter 正在
http://localhost:8080/metrics 執行...")

    while True:
        update_metrics()
        time.sleep(10)

```



以下是你提供的程式碼所對應的**技術文件**，格式為 ``CodeBlock``，包含模組用途、類別與函式解說、資料流程、Prometheus 整合說明等。

📖 技術文件：CSV-to-Prometheus Exporter with Dynamic Labels (忽略空值)

📌 模組用途

此模組為一個自訂 Prometheus Exporter：

- 從 ``bak-data_collect.csv`` 讀取資料；
- 分析每筆資料的動態 label 組合 (包含 host、job_name 與其他自定欄位)；
- 動態產出 Prometheus Gauge 指標；
- 自動忽略空值 label；
- 每 10 秒更新一次。

🛠 類別說明：CustomGauge

`class CustomGauge(name, documentation)`

自訂類別，用來動態建立 Prometheus 指標，允許 label 數量/名稱隨筆數變動，並排除空值。

方法：

- `__init__(self, name, documentation)`
 - 初始化指標名稱與說明。
 - `self.metrics`：儲存所有 (labels, value) 組合。
- `set(self, labels: dict, value: float)`
 - 新增一筆資料。
 - 自動移除 label 中為空字串的欄位。
 - 使用 sorted label tuple 作為 key，確保唯一性。
- `collect(self)`
 - 實作 Prometheus 的收集器接口。
 - 根據 label keys 做 grouping。
 - 建立多個 `GaugeMetricFamily` 對象並逐一回傳。

📄 函式說明

`parse_csv() -> Dict[frozenset, int]`

從 `CSV_FILE` 讀取每一列，並：

- 解析欄位：`host`，`job_name`，`{key=value,...}` 格式的附加欄位；
- 將每筆資料的所有 label 組成 dictionary；
- 使用 `frozenset` 作為 key 統計出現次數；
- 回傳統計結果。

`update_metrics()`

- 清空 `CustomGauge.metrics`；
- 呼叫 `parse_csv()` 取得統計結果；
- 逐筆呼叫 `CustomGauge.set()` 將 label-value 記錄進去；
- 同時記錄 log 觀察指標狀況。

🔄 主執行流程

```
```python
if __name__ == "__main__":
 start_http_server(8080)
 logging.info("Exporter 正在 http://localhost:8080/metrics 執行...")

 while True:
 update_metrics()
 time.sleep(10)
```

- 建立 Prometheus HTTP exporter server，暴露於 `http://localhost:8080/metrics` ；
- 每 10 秒重新載入 CSV 並更新指標。



**實驗輸入格式：`bak-data\_collect.csv`**

```
host_1,job_A,{service_name="aaa", container_name="bbbb"}
host_1,job_A
host_1,job_B
host_2,job_A
host_2,job_C
host_3,job_B,{module_name="cbbb"}
host_3,job_B
host_3,job_B
```



**Prometheus 輸出結果（範例）**





```
log_host_job_count{host="host_1", job_name="job_A", service_name="aaa",
container_name="bbbb"} 1.0
log_host_job_count{host="host_1", job_name="job_A"} 1.0
log_host_job_count{host="host_1", job_name="job_B"} 1.0
log_host_job_count{host="host_2", job_name="job_A"} 1.0
log_host_job_count{host="host_2", job_name="job_C"} 1.0
log_host_job_count{host="host_3", job_name="job_B", module_name="cbbb"} 1.0
log_host_job_count{host="host_3", job_name="job_B"} 2.0
```

## ✓ 功能優點總結

- ✓ 自動處理動態 label 組合；
- ✓ 空值 label 自動忽略，不產生不必要的指標；
- ✓ 支援任意多個 labels，不需事先定義；
- ✓ Prometheus 可直接 scrape `/metrics` 取得指標。

## 📌 適合應用場景

- 當你無法事先定義所有 label 名稱；
- 或者某些欄位在部分資料中為空，希望略過；
- 或是 CSV 為外部系統輸出格式，格式不固定。

如需擴充 Alertmanager 規則、自動轉換 JSON log、或產生單一指標多條目等進階功能，也歡迎提出，我可以幫你再擴寫。