



Prometheus Exporter 解析



以繁體中文詳細說明此技術文件:Prometheus Exporter 技術文件

1. 簡介

本 Prometheus Exporter 會從 CSV 讀取 log 數據，解析 host, job_name，並動態偵測可能的標籤 (service_name, container_name, module_name)。
根據不同的標籤組合，Exporter 會建立 3 種不同的 metrics：

log_host_job_basic：僅包含 host 和 job_name

log_host_job_service：包含 service_name, container_name

log_host_job_module：包含 module_name

2. CSV 檔案格式

示例 (bak-data_collect-1.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
```

3. 運作流程

讀取 CSV，解析 host 和 job_name

動態偵測額外標籤 (service_name, container_name,



```
module_name)
```

計算 metrics 數值

將 metrics 更新到 Prometheus

4. Prometheus 指標計算

```
counts_basic = {}  
counts_service = {}  
counts_module = {}
```

```
for host, job, extra_labels in log_data:
```

```
    # 基本 metrics
```

```
    basic_key = (host, job)
```

```
    counts_basic[basic_key] =
```

```
counts_basic.get(basic_key, 0) + 1
```

```
    # 包含 service_name, container_name 的 metrics
```

```
    if "service_name" in extra_labels or
```

```
"container_name" in extra_labels:
```

```
        service_key = (host, job, frozenset({k: v for k, v in  
extra_labels.items() if k in ["service_name",  
"container_name"]}).items()))
```

```
        counts_service[service_key] =
```

```
counts_service.get(service_key, 0) + 1
```

```
    # 包含 module_name 的 metrics
```

```
    if "module_name" in extra_labels:
```

```
        module_key = (host, job, frozenset({k: v for k, v in  
extra_labels.items() if k == "module_name"}.items()))
```

```
        counts_module[module_key] =
```

```
counts_module.get(module_key, 0) + 1
```

5. Prometheus 指標格式

```
# HELP log_host_job_service Count of occurrences with  
service-related labels
```

```
# TYPE log_host_job_service gauge
```

```
log_host_job_service{container_name="bbbb",host="h
```



```
ost_1",job_name="job_A",service_name="aaa"} 1.0

# HELP log_host_job_module Count of occurrences with
module-related labels
# TYPE log_host_job_module gauge
log_host_job_module{host="host_3",job_name="job_B"
,module_name="cbbb"} 1.0
```

6. 主要技術概念

set：用來存 labels，確保不會重複。

dict：存 metrics 計數，使用 .get() 來累加數值。

frozenset：用來當 dict 的 key，讓 labels 組合成唯一標識。

print debug：觀察 metrics 計算過程，確保計算正確。

7. Exporter 啟動方式

```
python3 exporter.py
```

8. 訪問 Prometheus Exporter

打開瀏覽器，訪問：

```
http://localhost:8080/metrics
```



當然可以，以下是這份 **Prometheus Exporter 技術文件** 的詳細說明，採用繁體中文，並補充了每個部分的背景與用途，幫助你更容易理解整體運作原理。

1. 簡介

這個 **Prometheus Exporter** 的主要功能是：



- 讀取 CSV 格式的 log 檔案
- 解析出基本資訊：host 和 job_name
- 自動偵測其他額外的標籤 (label)，例如：
 - `service_name`
 - `container_name`
 - `module_name`

根據出現的 **label 組合**，Exporter 會建立並暴露給 Prometheus 不同的 metrics：

Metric 名稱	包含的 Labels
`log_host_job_basic`	host, job_name
`log_host_job_service`	host, job_name, service_name, container_name
`log_host_job_module`	host, job_name, module_name



2. CSV 檔案格式

CSV 是此 Exporter 的資料來源，內容大致如下：

```
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
```





每行格式為：

```
host,job_name[, {label_key="label_value", ...}]
```

其中，第三欄是可選的，可包含以 `{}` 包起來的 label 字典。

3. 運作流程

1. 讀取 CSV 檔案

- 一行一行讀入 log 資料

2. 解析欄位

- 每行最少包含 `host` 和 `job_name`
- 若存在第三欄，則解析成 dict，當作額外的 labels

3. 偵測額外標籤

- 根據有沒有 `service_name` / `container_name` / `module_name` 決定要加入哪類 metrics

4. 計算每種指標的數量

- 用 dict 統計出現次數

5. 建立對應的 metrics 並回傳給 Prometheus

- 在 `/metrics` endpoint 上提供 Prometheus 可讀取的格式

4. Prometheus 指標計算邏輯



使用三個字典記錄對應的 metrics 數量：

```
counts_basic = {}
counts_service = {}
counts_module = {}
```

針對每筆資料 (host, job, extra_labels)，進行分類與計數：

```
# 基本 metrics : log_host_job_basic
basic_key = (host, job)
counts_basic[basic_key] = counts_basic.get(basic_key, 0) + 1

# 若有 service_name 或 container_name, 計入 log_host_job_service
if "service_name" in extra_labels or "container_name" in extra_labels:
    service_labels = {k: v for k, v in extra_labels.items() if k in
["service_name", "container_name"]}
    service_key = (host, job, frozenset(service_labels.items()))
    counts_service[service_key] = counts_service.get(service_key, 0) + 1

# 若有 module_name, 計入 log_host_job_module
if "module_name" in extra_labels:
    module_key = (host, job, frozenset({k: v for k, v in extra_labels.items()
if k == "module_name"}.items()))
    counts_module[module_key] = counts_module.get(module_key, 0) + 1
```

? 為什麼要用 `frozenset` ?

因為 dict 是無法當作 key 的（因為它是可變的），但 frozenset 是不可變、可哈希的，可以當作 dict 的 key，確保每種 label 組合是唯一的。



5. Prometheus 指標格式 (Metrics Format)

當你用瀏覽器打開 `/metrics` 時，會看到像這樣的輸出：





```
# HELP log_host_job_service Count of occurrences with service-related labels
# TYPE log_host_job_service gauge
log_host_job_service{container_name="bbbb",host="host_1",job_name="job_A",service_name="aaa"} 1.0

# HELP log_host_job_module Count of occurrences with module-related labels
# TYPE log_host_job_module gauge
log_host_job_module{host="host_3",job_name="job_B",module_name="cbbb"} 1.0
```

格式說明：

- ``# HELP``：說明這個 metrics 的用途
- ``# TYPE``：指定這個 metrics 的類型（通常為 gauge）
- ``log_host_job_xxx{labels}``：指標名稱與標籤
- 數值：表示出現的次數

6. 主要技術概念補充

技術	說明
<code>`set()`</code>	確保 labels 不重複
<code>`dict.get(key, default)`</code>	若 key 存在則回傳值，否則給預設值；這裡主要用來做累加計數
<code>`frozenset()`</code>	用來建立不可變的 label 組合，當作 dict 的 key
<code>`print debug`</code>	在開發過程中印出中間計算結果，有助於除錯與驗證邏輯正確性



7. Exporter 啟動方式

只要執行這行指令即可啟動：

```
python3 exporter.py
```

啟動後會在本地的 8080 port 開啟 HTTP 伺服器。

8. 訪問 Exporter

打開你的瀏覽器，輸入以下網址，即可看到 Prometheus metrics 的輸出：

```
http://localhost:8080/metrics
```

Prometheus 也可以設置這個 Exporter 作為 scrape target，自動收集資料。

如果你需要，我也可以幫你補上程式碼範例或是幫你修改成你需要的格式。是否需要繼續補充呢？



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

# 設置日誌
logging.basicConfig(
```





```
level=logging.INFO,
format="%%(asctime)s - %(levelname)s - %(message)s",
)

# **設定 CSV 檔案來源**
CSV_FILE = "bak-data_collect-1.csv"

def parse_csv():
    """從 bak-data_collect-1.csv 讀取資料，並動態解析
    標籤"""
    counts_basic = {}
    counts_service = {}
    counts_module = {}
    dynamic_labels_service = set()
    dynamic_labels_module = set()

    if not os.path.exists(CSV_FILE):
        logging.error(f"CSV 檔案 {CSV_FILE} 不存在！")
        return counts_basic, counts_service,
        counts_module, ["host", "job_name"], ["host",
        "job_name"]

    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 = {}

            # **解析 {} 內的標籤**
            for col in row[2:]:
                col = col.strip()

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



```
# **解析 key=value 格式**
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))

        # **確保 key 沒有 {}**
        key = key.replace("{", "").replace("}",
"".strip()

        # **修正標籤值，去除不必要的 "、\"**
        value = value.replace("\"",
"".replace("\\\"", "").strip("\\\"")

        if key and value:
            extra_labels[key] = value

# **記錄基本計數**
basic_key = (host, job_name)
counts_basic[basic_key] =
counts_basic.get(basic_key, 0) + 1

# **根據標籤分類到不同 metric**
if "service_name" in extra_labels or
"container_name" in extra_labels:
    service_key = (host, job_name, frozenset({k:
v for k, v in extra_labels.items() if k in ["service_name",
"container_name"]}.items()))
    counts_service[service_key] =
counts_service.get(service_key, 0) + 1

dynamic_labels_service.update(["service_name",
"container_name"])

if "module_name" in extra_labels:
    module_key = (host, job_name, frozenset({k:
v for k, v in extra_labels.items() if k ==
```



```
"module_name"}.items()))
        counts_module[module_key] =
counts_module.get(module_key, 0) + 1
        dynamic_labels_module.add("module_name")

# **動態標籤**
labels_list_service = sorted(["host", "job_name"] +
list(dynamic_labels_service))
labels_list_module = sorted(["host", "job_name"] +
list(dynamic_labels_module))

logging.info(f"Final dynamic_labels_service:
{labels_list_service}")
logging.info(f"Final dynamic_labels_module:
{labels_list_module}")

return counts_basic, counts_service, counts_module,
labels_list_service, labels_list_module

# **初始化 Prometheus 指標**
counts_basic, counts_service, counts_module,
labels_list_service, labels_list_module = parse_csv()

# **Metric 1：只包含基本 labels**
log_host_job_basic = Gauge(
    "log_host_job_basic",
    "Basic count of occurrences of host and job_name in
log",
    ["host", "job_name"]
)

# **Metric 2：包含 service_name, container_name**
log_host_job_service = Gauge(
    "log_host_job_service",
    "Count of occurrences with service-related labels",
    labels_list_service
)

# **Metric 3：包含 module_name**
log_host_job_module = Gauge(
```



```
"log_host_job_module",
"Count of occurrences with module-related labels",
labels_list_module
)

logging.info(f"[DEBUG] 設定 Prometheus 指標")
logging.info(f" - log_host_job_basic Labels: ['host',
'job_name']")
logging.info(f" - log_host_job_service Labels:
{labels_list_service}")
logging.info(f" - log_host_job_module Labels:
{labels_list_module}") # 🔍 Debug

def update_metrics():
    """更新 Prometheus 指標"""
    log_host_job_basic._metrics.clear()
    log_host_job_service._metrics.clear()
    log_host_job_module._metrics.clear()

    counts_basic, counts_service, counts_module,
labels_list_service, labels_list_module = parse_csv()

    logging.info("\n[DEBUG] 更新 metrics:")

    # **填充 log_host_job_basic**
    for (host, job), count in counts_basic.items():
        labels_dict = {"host": host, "job_name": job}
        logging.info(f"[DEBUG] 設定 log_host_job_basic
=> {labels_dict} : {count}") # 🔍 Debug
        log_host_job_basic.labels(**labels_dict).set(count)

    # **填充 log_host_job_service**
    for (host, job, extra_labels_tuple), count in
counts_service.items():
        extra_labels = dict(extra_labels_tuple)
        labels_dict = {label: extra_labels.get(label, "") for
label in labels_list_service}
        labels_dict["host"] = host
        labels_dict["job_name"] = job
        sorted_labels_dict = {key: labels_dict[key] for key
```



```
in labels_list_service}
    logging.info(f"[DEBUG] 設定
log_host_job_service => {sorted_labels_dict} :
{count}")

log_host_job_service.labels(**sorted_labels_dict).set(count)

# **填充 log_host_job_module**
for (host, job, extra_labels_tuple), count in
counts_module.items():
    extra_labels = dict(extra_labels_tuple)
    labels_dict = {label: extra_labels.get(label, "") for
label in labels_list_module}
    labels_dict["host"] = host
    labels_dict["job_name"] = job
    sorted_labels_dict = {key: labels_dict[key] for key
in labels_list_module}
    logging.info(f"[DEBUG] 設定 log_host_job_module
=> {sorted_labels_dict} : {count}")

log_host_job_module.labels(**sorted_labels_dict).set(count)

if __name__ == "__main__":
    # **啟動 Prometheus HTTP 伺服器**
    start_http_server(8080)
    logging.info("Prometheus exporter running on
http://localhost:8080/metrics")

    # **定期更新 metrics**
    while True:
        update_metrics()
        time.sleep(10)
```

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



```
start_http_server

# 設置日誌
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
)

# **設定 CSV 檔案來源**
CSV_FILE = "bak-data_collect-1.csv"

def parse_csv():
    """從 bak-data_collect-1.csv 讀取資料，並動態解析標籤"""
    counts_basic = {}
    counts_service = {}
    counts_module = {}
    dynamic_labels_service = set()
    dynamic_labels_module = set()

    if not os.path.exists(CSV_FILE):
        logging.error(f"CSV 檔案 {CSV_FILE} 不存在！")
        return counts_basic, counts_service, counts_module, ["host", "job_name"], ["host", "job_name"]

    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 = {}

            # **解析 {} 內的標籤**
            for col in row[2:]:
                col = col.strip()
```



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

# **解析 key=value 格式**
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))

        # **確保 key 沒有 {}**
        key = key.replace("{", "").replace("}",
"".strip())

        # **修正標籤值，去除不必要的 ", \"**
        value = value.replace("'",
"".replace("'", "").strip("\\'"))

        if key and value:
            extra_labels[key] = value

# **記錄基本計數**
basic_key = (host, job_name)
counts_basic[basic_key] =
counts_basic.get(basic_key, 0) + 1

# **根據標籤分類到不同 metric**
if "service_name" in extra_labels or
"container_name" in extra_labels:
    service_key = (host, job_name, frozenset({k:
v for k, v in extra_labels.items() if k in ["service_name",
"container_name"]}.items()))
    counts_service[service_key] =
counts_service.get(service_key, 0) + 1

dynamic_labels_service.update(["service_name",
"container_name"])
```



```
        if "module_name" in extra_labels:
            module_key = (host, job_name, frozenset({k:
v for k, v in extra_labels.items() if k ==
"module_name"}.items()))
            counts_module[module_key] =
counts_module.get(module_key, 0) + 1
            dynamic_labels_module.add("module_name")

# **動態標籤**
labels_list_service = sorted(["host", "job_name"] +
list(dynamic_labels_service))
labels_list_module = sorted(["host", "job_name"] +
list(dynamic_labels_module))

logging.info(f"Final dynamic_labels_service:
{labels_list_service}")
logging.info(f"Final dynamic_labels_module:
{labels_list_module}")

return counts_basic, counts_service, counts_module,
labels_list_service, labels_list_module

# **初始化 Prometheus 指標**
counts_basic, counts_service, counts_module,
labels_list_service, labels_list_module = parse_csv()

# **Metric 1：只包含基本 labels**
log_host_job_basic = Gauge(
    "log_host_job_basic",
    "Basic count of occurrences of host and job_name in
log",
    ["host", "job_name"]
)

# **Metric 2：包含 service_name, container_name**
log_host_job_service = Gauge(
    "log_host_job_service",
    "Count of occurrences with service-related labels",
    labels_list_service
```




```
)

# **Metric 3 : 包含 module_name**
log_host_job_module = Gauge(
    "log_host_job_module",
    "Count of occurrences with module-related labels",
    labels_list_module
)

logging.info(f"[DEBUG] 設定 Prometheus 指標")
logging.info(f" - log_host_job_basic Labels: ['host',
'job_name']")
logging.info(f" - log_host_job_service Labels:
{labels_list_service}")
logging.info(f" - log_host_job_module Labels:
{labels_list_module}") # 🔍 Debug

def update_metrics():
    """更新 Prometheus 指標"""
    log_host_job_basic._metrics.clear()
    log_host_job_service._metrics.clear()
    log_host_job_module._metrics.clear()

    counts_basic, counts_service, counts_module,
    labels_list_service, labels_list_module = parse_csv()

    logging.info("\n[DEBUG] 更新 metrics:")

    # **填充 log_host_job_basic**
    for (host, job), count in counts_basic.items():
        labels_dict = {"host": host, "job_name": job}
        logging.info(f"[DEBUG] 設定 log_host_job_basic =>
{labels_dict} : {count}") # 🔍 Debug
        log_host_job_basic.labels(**labels_dict).set(count)

    # **填充 log_host_job_service**
    for (host, job, extra_labels_tuple), count in
counts_service.items():
        extra_labels = dict(extra_labels_tuple)
        labels_dict = {label: extra_labels.get(label, "") for
```



```
label in labels_list_service}
    labels_dict["host"] = host
    labels_dict["job_name"] = job
    sorted_labels_dict = {key: labels_dict[key] for key
in labels_list_service}
    logging.info(f"[DEBUG] 設定 log_host_job_service
=> {sorted_labels_dict} : {count}")

log_host_job_service.labels(**sorted_labels_dict).set(co
unt)

# **填充 log_host_job_module**
for (host, job, extra_labels_tuple), count in
counts_module.items():
    extra_labels = dict(extra_labels_tuple)
    labels_dict = {label: extra_labels.get(label, "") for
label in labels_list_module}
    labels_dict["host"] = host
    labels_dict["job_name"] = job
    sorted_labels_dict = {key: labels_dict[key] for key
in labels_list_module}
    logging.info(f"[DEBUG] 設定 log_host_job_module
=> {sorted_labels_dict} : {count}")

log_host_job_module.labels(**sorted_labels_dict).set(co
unt)

if __name__ == "__main__":
    # **啟動 Prometheus HTTP 伺服器**
    start_http_server(8080)
    logging.info("Prometheus exporter running on
http://localhost:8080/metrics")

    # **定期更新 metrics**
    while True:
        update_metrics()
        time.sleep(10)
```





你這段程式碼寫得非常清晰，整體架構完整，也很好地處理了動態標籤與 Prometheus 指標更新的邏輯 🍌

以下是這份程式的詳細中文解說（含原理與設計亮點），幫你更好地理解與說明這個 Prometheus Exporter：

主要功能概覽

這個 Python 程式的功能是建立一個 **Prometheus Exporter**，每隔 10 秒：

1. 從 **CSV** 檔讀取 **log** 資料
2. 分析 **host**、**job_name** 及其他動態標籤（如 **service_name**、**container_name**、**module_name**）
3. 建立/更新三種 **metrics**
 - ``log_host_job_basic``
 - ``log_host_job_service``
 - ``log_host_job_module``
4. 在 ``/metrics`` 上提供 **Prometheus** 可抓取的格式

主要元件與邏輯詳解

1. 日誌系統設定

```
logging.basicConfig(  
    level=logging.INFO,
```



```
format="%(%asctime)s - %(levelname)s - %(message)s",  
)
```

使用 ``logging`` 模組來記錄訊息，幫助 debug 與運作監控。

2. 讀取與解析 CSV：``parse_csv()``

這個函數的工作：

- **讀入每一行 CSV**
- 解析成 ``host``，``job_name``，與 **額外標籤**
- 根據標籤內容分類為三種 metrics：
 - basic
 - service
 - module
- 記錄出現次數（用 dict 來計數）
- 動態建立需要的 label 名稱（保存在 ``dynamic_labels_service``，``dynamic_labels_module``）

“ **重點技巧：**”

- “使用 ``frozenset`` 讓 dict 可以支援複合 key”
- “使用 ``dict.get()`` 來進行累加統計”

3. Prometheus 指標定義（Gauge）



```
log_host_job_basic = Gauge("log_host_job_basic", ..., ["host", "job_name"])
log_host_job_service = Gauge(..., labels_list_service)
log_host_job_module = Gauge(..., labels_list_module)
```

這裡定義了三個 `Gauge` 指標，其中：

- `log_host_job_basic` 是固定標籤
- `log_host_job_service` 與 `log_host_job_module` 的標籤是「動態從資料中分析出來的」

這樣做可以依據真實資料量身打造 Prometheus 指標。

4. Metrics 更新函數：`update_metrics()`

這個函數會每 10 秒：

1. 重新解析一次 CSV（確保資料是最新的）
2. 清除舊的 metrics（用 `_metrics.clear()`）
3. 根據分析結果，用 `_.labels(...).set(...)` 來填入最新的值

```
log_host_job_service.labels(**sorted_labels_dict).set(count)
```

“ 使用 `_.labels(**dict)` 是 *Prometheus client* 常用的方式來填入標籤資料。”

5. 啟動 Exporter 主程式

```
start_http_server(8080)
```



啟動 Prometheus 可讀的 HTTP 伺服器，預設會開在：

```
http://localhost:8080/metrics
```

Prometheus 就可以透過這個 Endpoint 去 scrape metrics。

補充建議與小提醒

建議改進項目：

1. 將 ``parse_csv()`` 拆成「資料讀取」與「**metrics 統計**」兩段邏輯，會讓單元測試更容易。
2. ``_metrics.clear()`` 是私有屬性，雖然可以用，但較推薦使用 ``Gauge._metrics.clear()`` 時加上註解提醒。
3. 可以增加「上次更新時間」的 metric，例如：

```
last_update_time = Gauge("exporter_last_update_timestamp", "最後更新時間的 Unix  
timestamp")  
last_update_time.set_to_current_time()
```

這樣 Prometheus 可以監控 Exporter 是否持續運作。

實際使用方式

將 ``bak-data_collect-1.csv`` 放在與程式同層目錄，格式如下：

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



```
host_2, job_B  
host_3, job_C, {module_name="xyz"}
```

啟動 Exporter：

```
python3 exporter.py
```

然後瀏覽：

```
http://localhost:8080/metrics
```

就會看到 Prometheus 格式的 metrics！

如果你想進一步優化，例如加入重新載入的延遲監控、log 數據熱更新、透過參數指定 CSV 路徑、或改為讀取資料夾內所有 CSV，我也可以幫你改寫 🔧

你接下來會部署到哪個環境？是測試機還是直接放 Kubernetes？我可以順便幫你寫 `Dockerfile` 或 Helm chart～



我想要理解這段程式碼用法，為了加到我到 exporter 中做實驗時，我對這些 code 不理解，你能教我嗎，你能多 print 出過程嗎，或是你能給更簡單的範例：

```
# **根據標籤分類到不同 metric**  
if "service_name" in extra_labels or  
"container_name" in extra_labels:  
    service_key = (host, job_name, frozenset({k:  
v for k, v in extra_labels.items() if k in ["service_name",  
"container_name"]}.items()))  
    counts_service[service_key] =  
counts_service.get(service_key, 0) + 1
```



```
dynamic_labels_service.update(["service_name",
                                "container_name"])

    if "module_name" in extra_labels:
        module_key = (host, job_name, frozenset({k:
v for k, v in extra_labels.items() if k ==
"module_name"}.items()))
        counts_module[module_key] =
counts_module.get(module_key, 0) + 1
        dynamic_labels_module.add("module_name")
```



非常可以，我一步步幫你拆解這段程式碼的用途、概念，並加上 ``print()`` 印出中間過程的簡化版本，讓你可以清楚觀察它怎麼「依照標籤分類並統計」。

你這段程式碼的目的

這兩段程式碼是要將每一筆 log 資料「分類」成對應的 metrics，依照是否有特定標籤

(``service_name``, ``container_name``, ``module_name``)：

- 如果資料中有 ``service_name`` 或 ``container_name``
→ 加到 ``counts_service`` 裡
- 如果資料中有 ``module_name``
→ 加到 ``counts_module`` 裡

為了避免標籤順序造成 key 不一致 (Python dict 無法當 key)，才會把標籤轉成 ``frozenset`` (不可變、可當 key 的集合)

更簡化版教學 + 印出結果



你可以先執行這個簡單版的程式來理解：

```
log_data = [
    ("host_1", "job_A", {"service_name": "aaa", "container_name": "bbbb"}),
    ("host_2", "job_B", {}),
    ("host_3", "job_C", {"module_name": "xyz"}),
    ("host_1", "job_A", {"service_name": "aaa"}), # 跟第一行重複但少一個 label
]

counts_service = {}
counts_module = {}
dynamic_labels_service = set()
dynamic_labels_module = set()

for host, job, extra_labels in log_data:
    print("\n📄 目前這筆資料：")
    print(f"host={host}, job={job}, extra_labels={extra_labels}")

    # 處理 service_name/container_name 的 metric
    if "service_name" in extra_labels or "container_name" in extra_labels:
        # 把 service_name 和 container_name 拉出來組合成 key
        filtered_labels = {k: v for k, v in extra_labels.items() if k in
["service_name", "container_name"]}
        print(f"✅ [Service] 抽出 service/container labels:
{filtered_labels}")

        # key 要是不可變，所以轉成 frozenset
        service_key = (host, job, frozenset(filtered_labels.items()))
        print(f"🧩 組成 service_key: {service_key}")

        counts_service[service_key] = counts_service.get(service_key, 0) + 1
        dynamic_labels_service.update(["service_name", "container_name"])

    # 處理 module_name 的 metric
    if "module_name" in extra_labels:
        filtered_labels = {k: v for k, v in extra_labels.items() if k ==
"module_name"}
        print(f"✅ [Module] 抽出 module_name label: {filtered_labels}")

        module_key = (host, job, frozenset(filtered_labels.items()))
```



```

print(f" 🧩 組成 module_key: {module_key}")

counts_module[module_key] = counts_module.get(module_key, 0) + 1
dynamic_labels_module.add("module_name")

print("\n📊 最後統計結果:")
print("counts_service:")
for k, v in counts_service.items():
    print(f" {k} => {v}")

print("counts_module:")
for k, v in counts_module.items():
    print(f" {k} => {v}")

```

🧠 執行結果說明（簡化）

📄 目前這筆資料：

host=host_1, job=job_A, extra_labels={'service_name': 'aaa', 'container_name': 'bbbb'}

✅ [Service] 抽出 service/container labels: {'service_name': 'aaa', 'container_name': 'bbbb'}

🧩 組成 service_key: ('host_1', 'job_A', frozenset({'container_name', 'bbbb'}, {'service_name', 'aaa'}))

...

📊 最後統計結果：

counts_service:

('host_1', 'job_A', frozenset({'container_name', 'bbbb'}, {'service_name', 'aaa'})) => 1

('host_1', 'job_A', frozenset({'service_name', 'aaa'})) => 1

counts_module:

('host_3', 'job_C', frozenset({'module_name', 'xyz'})) => 1



✓ 重點整理

用法	說明
<code>`frozenset(labels.items())`</code>	把標籤轉成不可變的集合，可以當作 dict 的 key
<code>`dict.get(key, 0) + 1`</code>	統計某個 key 的出現次數
<code>`dynamic_labels*.update([...])`</code>	記錄出現過哪些 labels，稍後會用來建立 Prometheus metric

? 你可以這樣思考

- 每一筆資料，都可能有不同的 labels
- 但 Prometheus 的每種 metric，**標籤名稱要固定**
(所以才先統計出有哪些 dynamic labels)
- 我們要記錄：哪個 host + job_name + 哪組 labels 出現了幾次

✓ 我可以幫你加到原本程式碼中，加上 debug 印出這些 key 與 frozenset 的內容，這樣你做實驗時可以看到：

```
print(f"[DEBUG] host={host}, job_name={job_name}")
print(f"[DEBUG] extra_labels: {extra_labels}")
print(f"[DEBUG] service_key: {service_key}")
print(f"[DEBUG] counts_service current count: {counts_service[service_key]}")
```



如果你想，我也可以幫你寫一版「不使用 frozenset」但結構比較簡單的版本（雖然就無法當 key），用來對比學習，是否需要呢？或者我可以直接協助你整合這些 print 到你的 exporter 主程式中方便實驗 🔧



看不懂k、v分別代表什麼：

```
module_key = (host, job_name, frozenset({k: v for k, v
in extra_labels.items() if k ==
"module_name"}.items()))
```



很好！這是一個常見的 Python 語法組合，我來一步步帶你拆解這行程式碼，特別是你問的 `k`，`v` 分別是什麼，並用簡單範例幫你理解 📌

原始程式碼

```
module_key = (host, job_name, frozenset({k: v for k, v in extra_labels.items()
if k == "module_name"}.items()))
```

這行程式主要在建立一個「唯一的 key」，用來記錄某個 `(host, job_name, module_name)` 組合出現了幾次。

拆解語法

我們先聚焦在裡面這段：



```
{k: v for k, v in extra_labels.items() if k == "module_name"}
```

這是一個 **dictionary comprehension**（字典推導式）。

✅ `k, v in extra_labels.items()` 是什麼？

假設你有一個 dict：

```
extra_labels = {  
    "module_name": "abc",  
    "other_label": "xyz"  
}
```

當你用 `extra_labels.items()` 迴圈時，它會一組一組傳回：

k	v
`"module_name"`	`"abc"`
`"other_label"`	`"xyz"`

所以 `k` 是「key 名稱」，`v` 是「對應的值」。

💡 搭配整行理解

這段：

```
{k: v for k, v in extra_labels.items() if k == "module_name"}
```

意思是：



👉 從 `extra_labels` 中，只挑出 **key** 是 `"module_name"` 的項目，組成一個新的 **dict**

如果原本 `extra_labels` 是：

```
{"module_name": "abc", "other": "ignore"}
```

那這段結果就會是：

```
{"module_name": "abc"}
```

🔄 為什麼還要 `.items()` ？

因為我們接下來要放進 `frozenset()` 當作 dict 的 key，而 `frozenset` 不能吃 dict，只能吃一個「集合」，例如：

```
frozenset([("module_name", "abc")])
```

而 `.items()` 剛好就是把 dict 變成這種形式：

```
{"module_name": "abc"}.items()  
➡ [("module_name", "abc")]
```

✅ 最後完整理解這行

```
module_key = (  
    host,  
    job_name,
```



```
frozenset(  
    {k: v for k, v in extra_labels.items() if k == "module_name"}.items()  
)
```

簡化理解：

1. `host` 和 `job_name` 是基本資料
2. 第三項是一組固定標籤：`{"module_name": "abc"}` → 變成 `frozenset`，這樣可以當 dict 的 key

小測試程式

你可以直接跑下面的程式觀察過程：

```
extra_labels = {  
    "module_name": "abc",  
    "service_name": "def"  
}  
  
filtered = {k: v for k, v in extra_labels.items() if k == "module_name"}  
print("🎯 篩選後的 filtered dict:", filtered)  
  
frozen = frozenset(filtered.items())  
print("📦 frozenset 結果:", frozen)  
  
module_key = ("host_1", "job_X", frozen)  
print("🔑 module_key 組成結果:", module_key)
```

總結



名稱	是什麼
<code>`k`</code>	標籤名稱 (例如 <code>`"module_name"`</code>)
<code>`v`</code>	標籤值 (例如 <code>`"abc"`</code>)
<code>`.items()`</code>	讓你可以同時取出 key 和 value
<code>`frozenset(...)`</code>	讓我們能把多個標籤變成「可以當 key」的東西

需要的話我也可以幫你把這段拆成多行 + 加上 ``print()`` 整合進你的 Exporter 方便觀察～

要我幫你整合進主程式印出 debug log 嗎？還是你想自己先試試這個小程式？😊



我想要理解這段程式碼用法，為了加到我到exporter中做實驗時，我對這些code不理解，你能教我嗎，你能多print出過程嗎，或是你能給更簡單的範例：

```
# **填充 log_host_job_service**
for (host, job, extra_labels_tuple), count in
counts_service.items():
    extra_labels = dict(extra_labels_tuple)
    labels_dict = {label: extra_labels.get(label, "") for
label in labels_list_service}
    labels_dict["host"] = host
    labels_dict["job_name"] = job
    sorted_labels_dict = {key: labels_dict[key] for key
in labels_list_service}
    logging.info(f"[DEBUG] 設定
log_host_job_service => {sorted_labels_dict} :
{count}")

log_host_job_service.labels(**sorted_labels_dict).set(co
```




```
unt)

# **填充 log_host_job_module**
for (host, job, extra_labels_tuple), count in
counts_module.items():
    extra_labels = dict(extra_labels_tuple)
    labels_dict = {label: extra_labels.get(label, "") for
label in labels_list_module}
    labels_dict["host"] = host
    labels_dict["job_name"] = job
    sorted_labels_dict = {key: labels_dict[key] for key
in labels_list_module}
    logging.info(f"[DEBUG] 設定 log_host_job_module
=> {sorted_labels_dict} : {count}")

log_host_job_module.labels(**sorted_labels_dict).set(co
unt)
```



你問的這段程式碼是把之前統計出來的 `counts_service` / `counts_module` 資料填進 Prometheus 的指標（metrics）裡，這邊牽涉到幾個重點概念：

程式碼目的是什麼？

簡單來說：

把你統計好的次數：

```
counts_service[("host_1", "job_A", frozenset(...))] = 3
```

轉成 Prometheus 格式：



```
log_host_job_service{host="host_1", job_name="job_A", service_name="aaa"} 3
```

這段程式碼的任務，就是幫你「把 dict 裡的 key 拆成一組 label」，然後餵進 Prometheus 的 `.labels(...)` 去設定對應的次數。

✨ 每行程式做了什麼？

我們用 **第一段 (service)** 的做說明，module 跟它幾乎一樣。

🔄 迴圈在做什麼？

```
for (host, job, extra_labels_tuple), count in counts_service.items():
```

從你之前的 `counts_service` 取出：

- `(host, job, extra_labels_tuple)`：是你記錄的 key（有 frozenset）
- `count`：是那組 key 出現幾次

📦 轉回 dict 方便使用

```
extra_labels = dict(extra_labels_tuple)
```

因為 `frozenset` 是像這樣的資料：

```
frozenset({"service_name", "aaa"}, {"container_name", "bbbb"})
```

轉回 dict 後你就可以用：



```
extra_labels["service_name"] → "aaa"
```

📦 組出完整 label dict

```
labels_dict = {label: extra_labels.get(label, "") for label in
labels_list_service}
labels_dict["host"] = host
labels_dict["job_name"] = job
```

這邊把 label 組成一個 dict，要注意幾點：

- 為了保證 **每次都包含相同欄位數**，即使某個欄位沒有，也會補成空字串 `""`
- `labels_list_service` 是你在 `parse_csv` 那邊收集出來的標籤欄位順序，例如：

```
['host', 'job_name', 'container_name', 'service_name']
```

🧠 排序一致的標籤字典

```
sorted_labels_dict = {key: labels_dict[key] for key in labels_list_service}
```

這行是為了保證標籤順序一致，雖然 Prometheus 的 labels 是無序的，但有些系統（像 `Gauge._metrics`）內部有時會依順序比較，這樣做比較保險。

📦 餵進 Prometheus 的 metric



```
log_host_job_service.labels(**sorted_labels_dict).set(count)
```

這就是正式更新 Prometheus 指標，會在 `/metrics` 上看到：

```
log_host_job_service{host="host_1", job_name="job_A", service_name="aaa",  
container_name="bbbb"} 3
```

✅ 加上 `print()` 觀察過程（簡化版範例）

這裡給你一個完整可跑的簡化範例：

```
from prometheus_client import Gauge

# 模擬 counts_service
counts_service = {
    ("host_1", "job_A", frozenset({"service_name", "aaa"}, {"container_name", "bbbb"})): 2,
    ("host_2", "job_B", frozenset({"service_name", "ccc"})): 1,
}

# 模擬 labels 順序
labels_list_service = ["host", "job_name", "service_name", "container_name"]

# 建立 Prometheus metric (這邊不啟動 server, 只示範)
log_host_job_service = Gauge(
    "log_host_job_service", "服務標籤計數", labels_list_service
)

# 模擬填充流程 + 加 print
for (host, job, extra_labels_tuple), count in counts_service.items():
    print("\n🔄 一筆資料:")
    print(f" host={host}, job={job}")
    extra_labels = dict(extra_labels_tuple)
    print(f" extra_labels = {extra_labels}")
```



```

labels_dict = {label: extra_labels.get(label, "") for label in
labels_list_service}
labels_dict["host"] = host
labels_dict["job_name"] = job

sorted_labels_dict = {key: labels_dict[key] for key in labels_list_service}

print(f" ✅ 最終填入 Prometheus 的 labels: {sorted_labels_dict}")
print(f" ✅ 設定數值為: {count}")

# 這行正式設定 Prometheus 的指標
log_host_job_service.labels(**sorted_labels_dict).set(count)

```

🧠 小技巧：為什麼需要「填補空字串」？

如果某筆資料只有 `service_name` 沒有 `container_name`：

```
extra_labels = {"service_name": "aaa"}
```

為了讓 Prometheus 的 `log_host_job_service` 每筆資料都包含 **固定的標籤欄位數（4個）**，所以會把 `container_name` 補成 `""`（空字串），這樣才不會報錯。

✅ 總結關鍵點

概念	說明
`extra_labels_tuple` → `dict()`	把 frozenset 轉回 dict 方便取值
`labels_dict.get(label, "")`	補上沒出現的 label 為空字串



概念	說明
<code>`sorted_labels_dict`</code>	確保順序一致，避免指標建立錯誤
<code>`.labels(**labels).set(count)`</code>	更新 Prometheus metric

如果你想，我可以幫你在你的 `exporter.py` 裡面加上這樣的 print debug（或 logging）段落，方便你實驗時觀察。有需要我幫你整合進主程式嗎？或者你想先自己試試這段小程式？😊