Jaeger trace 文档

打点代码分支：

https://github.com/pyxyzc/sglang/tree/jeager

安装依赖服务

安装 启动 Jaeger （UI）

代码块

```
wget https://github.com/jaegertracing/jaeger/releases/download/v1.76.0/jaeger-1.76.0-linux-amd64.tar.gz

tar -xzvf jaeger-1.76.0-linux-amd64.tar.gz

cd jaeger-1.76.0-linux-amd64

# 使用绝对路径进行软连接保证全局可用，修改为jaeger-all-in-one的绝对路径
ln -s /sgl-workspace/jaeger/jaeger-1.76.0-linux-amd64/jaeger-all-in-one \
/usr/local/bin/jaeger-all-in-one

jaeger-all-in-one   --collector.otlp.enabled=true   --collector.otlp.grpc.host-port=0.0.0.0:4327   --collector.otlp.http.host-port=0.0.0.0:4328
```

> 报错 too many open files，具体报错信息：
>
> `{"level":"panic","ts":1765965126.7292247,"caller":"app/static_handler.go:46","msg":"Could not create static assets handler","error":"too many open files","stacktrace": ...}`
>
> `panic: Could not create static assets handler`
>
> 解决方法如下：
>
> 编辑 /etc/security/limits.conf
>
> 代码块
>
> ```
> jaeger_user soft nofile 65536
> jaeger_user hard nofile 65536
> ```

安装 启动 otel-collector (数据中转)

代码块

```
wget https://github.com/open-telemetry/opentelemetry-collector-releases/releases/download/v0.114.0/otelcol-contrib_0.114.0_linux_amd64.tar.gz

tar -xzvf otelcol-contrib_0.114.0_*.tar.gz

ln -s otelcol-contrib /usr/local/bin/otelcol-contrib


# 启动前先创建配置文件
otelcol-contrib  --config otel-collector.yaml
```

otel-collector.yaml

```yaml
receivers:
  otlp:
    protocols:
      grpc:
        endpoint: 0.0.0.0:4317
      http:
        endpoint: 0.0.0.0:4318

exporters:
  otlp:
    endpoint: "localhost:4327"    # Jaeger OTLP gRPC 地址
    tls:
      insecure: true              # 本地环境常用

service:
  pipelines:
    traces:
      receivers: [otlp]
      exporters: [otlp]

```

Sglang 启动参数

代码块

```bash
#!/bin/bash
```

```

# 模型路径（改成你本地的，比如 /data/models/Llama-2-7b-hf）
MODEL_PATH=/home/models/QwQ-32B

# 端口号（可以改，比如 30000）
PORT=30000
# 卡数量
TP=2

# 只使用
export CUDA_VISIBLE_DEVICES=0,1
# 启动命令
python python/sglang/launch_server.py \
    --model-path $MODEL_PATH \
    --tp $TP \
    --port $PORT \
    --enable-trace \
    --otlp-traces-endpoint 127.0.0.1:4317 \
    # --enable-metrics
```

同时启停脚本

需要将otel-collector启动配置文件放到sglang文件夹内，直接运行即可。

run_server_jaeger.sh

```bash
#!/usr/bin/env bash
set -e

#--------------------------------
# 端口检查函数
#--------------------------------
check_and_kill_port() {
    local port=$1
    local pid

    pid=$(lsof -t -i:"$port" || true)
    if [[ -n "$pid" ]]; then
        echo "Port $port is in use by PID $pid. Killing..."
        kill -9 "$pid" || true
        echo "Port $port cleared."
    else
        echo "Port $port is free."
    fi
}
```

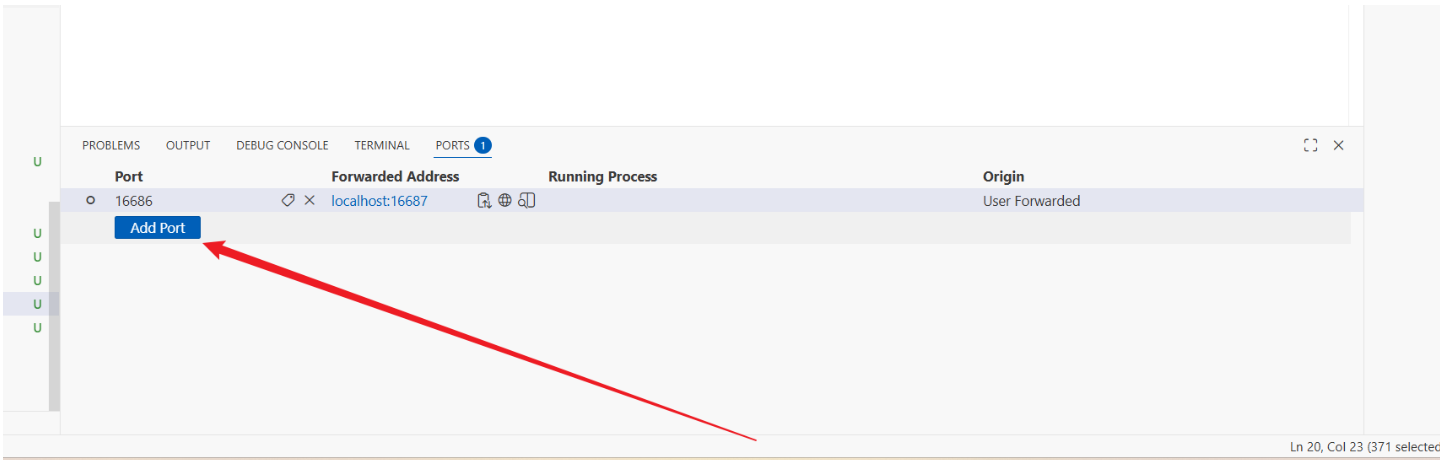
```bash

#-------------------------------
# 启动前的环境变量
#-------------------------------
export SGLANG_HICACHE_FILE_BACKEND_STORAGE_DIR=/sgl-workspace/hicachefile

MODEL_PATH=/home/models/QwQ-32B
PORT=30000
TP=2

export CUDA_VISIBLE_DEVICES=0,1

#-------------------------------
# 启动前检查端口
#-------------------------------
echo "Checking required ports..."
check_and_kill_port 4317   # otlp grpc
check_and_kill_port 4327   # jaeger grpc
check_and_kill_port 4328   # jaeger http
check_and_kill_port $PORT  # sglang server

#-------------------------------
# 捕获退出信号，统一杀死所有子进程
#-------------------------------
pids=()

cleanup() {
    echo "Received stop signal. Killing all subprocesses..."
    for pid in "${pids[@]}"; do
        if kill -0 "$pid" 2>/dev/null; then
            kill -9 "$pid" 2>/dev/null || true
        fi
    done
    exit 1
}

trap cleanup SIGINT SIGTERM EXIT

#-------------------------------
# 启动第 1 个: Jaeger
#-------------------------------
jaeger-all-in-one \
  --collector.otlp.enabled=true \
  --collector.otlp.grpc.host-port=0.0.0.0:4327 \
  --collector.otlp.http.host-port=0.0.0.0:4328 &
pids+=($!)
echo "Started Jaeger, PID=${pids[-1]}"
```

```

#-------------------------------
# 启动第 2 个: otelcol-contrib
#-------------------------------
otelcol-contrib --config otel-collector.yaml &
pids+=($!)
echo "Started otelcol-contrib, PID=${pids[-1]}"

#-------------------------------
# 启动第 3 个: sglang server
#-------------------------------
python python/sglang/launch_server.py \
    --model-path $MODEL_PATH \
    --page-size 128 \
    --tp $TP \
    --port $PORT \
    --enable-hierarchical-cache \
    --hicache-write-policy write_through \
    --hicache-storage-backend file \
    --hicache-storage-prefetch-policy wait_complete \
    --enable-trace \
    --otlp-traces-endpoint 127.0.0.1:4317 &
pids+=($!)
echo "Started sglang server, PID=${pids[-1]}"

#-------------------------------
# 等待任意子进程退出
#-------------------------------
echo "All processes started. Waiting for any to exit..."
wait -n

echo "One process exited. Killing all..."
cleanup

```

结果查看

VScode 转发端口

`16686` 端口

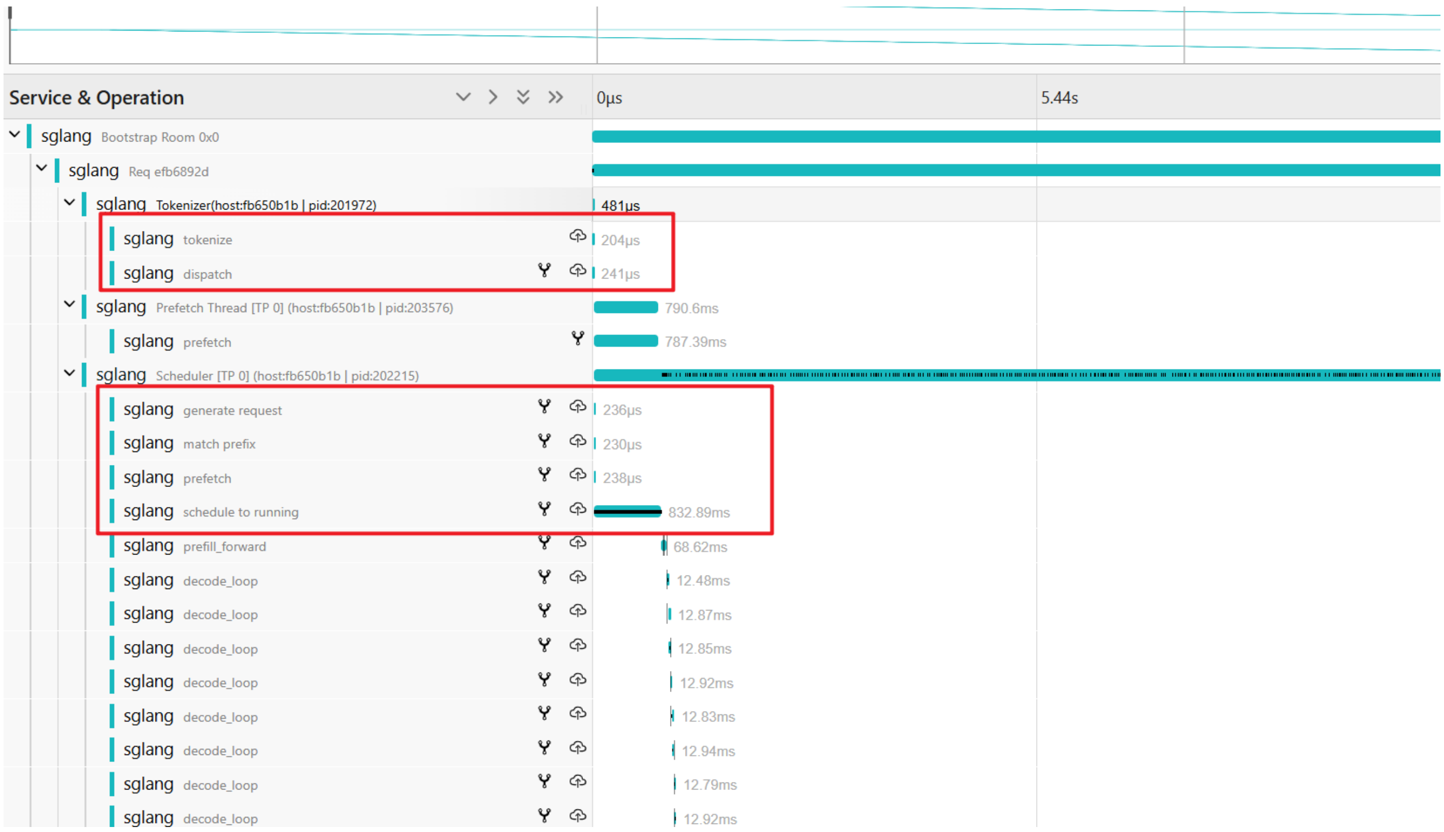


Port	Forwarded Address	Running Process	Origin
16686	localhost:16687		User Forwarded

Ln 20, Col 23 (371 selected

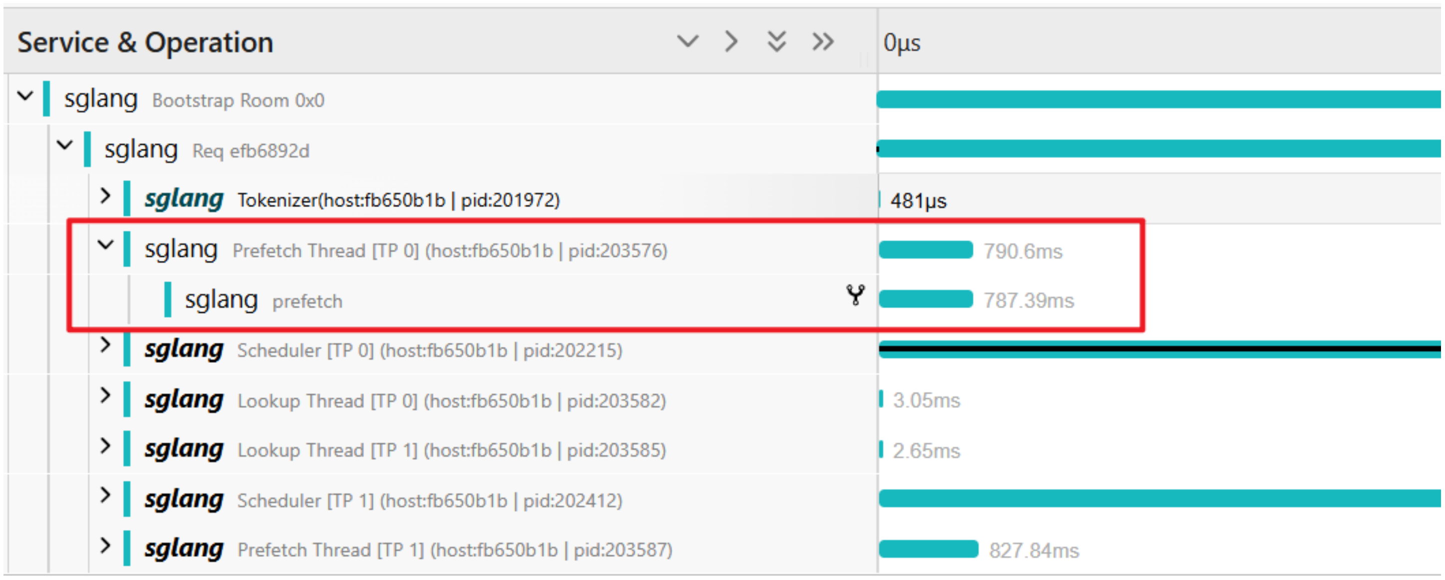
查看打点数据!

scheduler调度数据



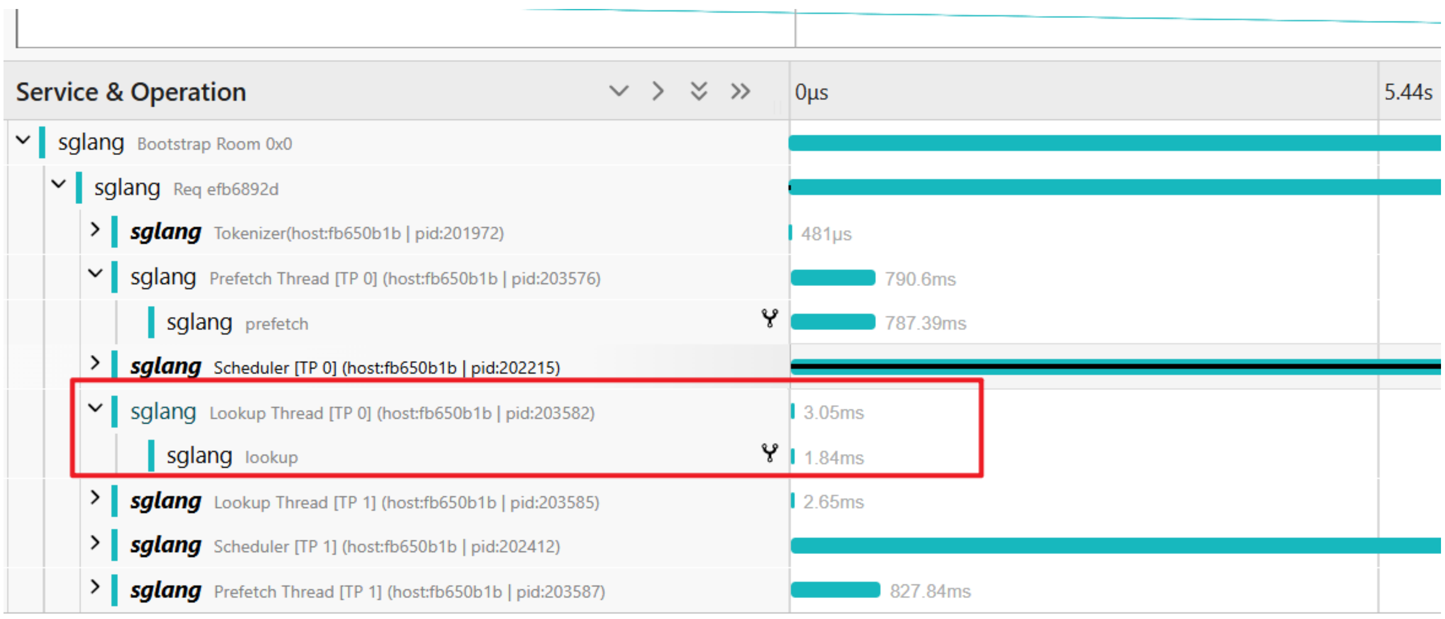
Service & Operation	0μs / 5.44s
sglang Bootstrap Room 0x0	
sglang Req efb6892d	
sglang Tokenizer(host:fb650b1b | pid:201972)	481μs
sglang tokenize	204μs
sglang dispatch	241μs
sglang Prefetch Thread [TP 0] (host:fb650b1b | pid:203576)	790.6ms
sglang prefetch	787.39ms
sglang Scheduler [TP 0] (host:fb650b1b | pid:202215)	
sglang generate request	236μs
sglang match prefix	230μs
sglang prefetch	238μs
sglang schedule to running	832.89ms
sglang prefill_forward	68.62ms
sglang decode_loop	12.48ms
sglang decode_loop	12.87ms
sglang decode_loop	12.85ms
sglang decode_loop	12.92ms
sglang decode_loop	12.83ms
sglang decode_loop	12.94ms
sglang decode_loop	12.79ms
sglang decode_loop	12.92ms

prefetch线程耗时


Service & Operation | 0μs

- sglang Bootstrap Room 0x0
  - sglang Req efb6892d
    - sglang Tokenizer(host:fb650b1b | pid:201972) — 481μs
    - sglang Prefetch Thread [TP 0] (host:fb650b1b | pid:203576) — 790.6ms
      - sglang prefetch — 787.39ms
    - sglang Scheduler [TP 0] (host:fb650b1b | pid:202215)
    - sglang Lookup Thread [TP 0] (host:fb650b1b | pid:203582) — 3.05ms
    - sglang Lookup Thread [TP 1] (host:fb650b1b | pid:203585) — 2.65ms
    - sglang Scheduler [TP 1] (host:fb650b1b | pid:202412)
    - sglang Prefetch Thread [TP 1] (host:fb650b1b | pid:203587) — 827.84ms

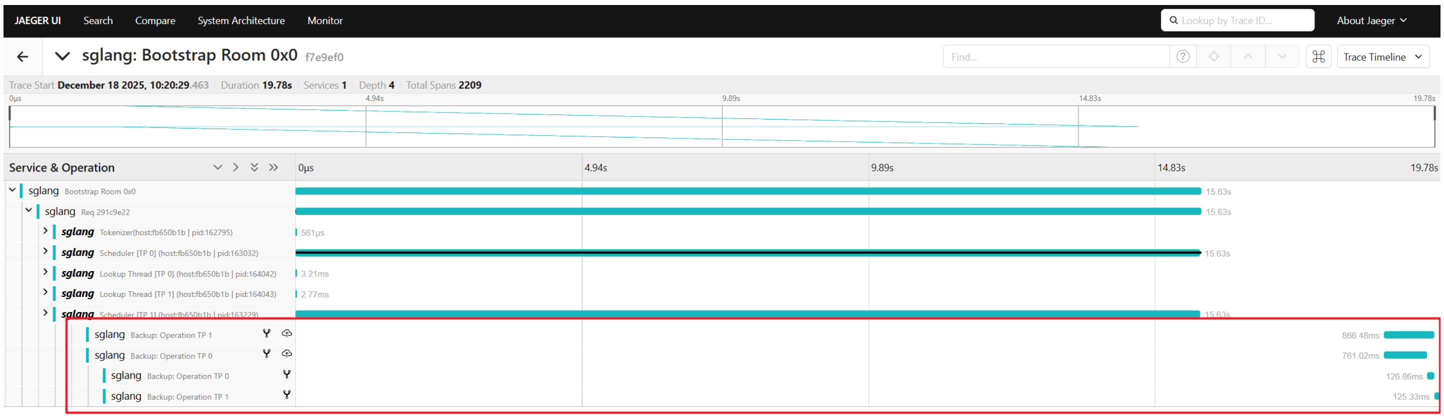
lookup线程耗时


Service & Operation | 0μs | 5.44s

- sglang Bootstrap Room 0x0
  - sglang Req efb6892d
    - sglang Tokenizer(host:fb650b1b | pid:201972) — 481μs
    - sglang Prefetch Thread [TP 0] (host:fb650b1b | pid:203576) — 790.6ms
      - sglang prefetch — 787.39ms
    - sglang Scheduler [TP 0] (host:fb650b1b | pid:202215)
    - sglang Lookup Thread [TP 0] (host:fb650b1b | pid:203582) — 3.05ms
      - sglang lookup — 1.84ms
    - sglang Lookup Thread [TP 1] (host:fb650b1b | pid:203585) — 2.65ms
    - sglang Scheduler [TP 1] (host:fb650b1b | pid:202412)
    - sglang Prefetch Thread [TP 1] (host:fb650b1b | pid:203587) — 827.84ms

dump线程耗时

对dump线程进行绑定后，可以记录dump耗时



sglang: Bootstrap Room 0x0 f7e9ef0

Trace Start **December 18 2025, 10:20:29**.463 | Duration **19.78s** | Services **1** | Depth **4** | Total Spans **2209**

Service & Operation	Duration
sglang Bootstrap Room 0x0	15.63s
sglang Req 291c9e22	15.63s
sglang Tokenizer(host:fb650b1b | pid:162795)	561µs
sglang Scheduler [TP 0] (host:fb650b1b | pid:163032)	15.63s
sglang Lookup Thread [TP 0] (host:fb650b1b | pid:164042)	3.21ms
sglang Lookup Thread [TP 1] (host:fb650b1b | pid:164043)	2.77ms
sglang Scheduler [TP 1] (host:fb650b1b | pid:163229)	15.63s
sglang Backup: Operation TP 1	866.48ms
sglang Backup: Operation TP 0	761.02ms
sglang Backup: Operation TP 0	126.86ms
sglang Backup: Operation TP 1	125.33ms

添加打点教程

https://docs.sglang.io/references/production_request_trace.html

注意：

1. 和req相关的打点直接增加 `trace_slice_start` 、 `trace_slice_end` 即可