Be careful of your DAG definition (Avoid heavy process)

Metadata

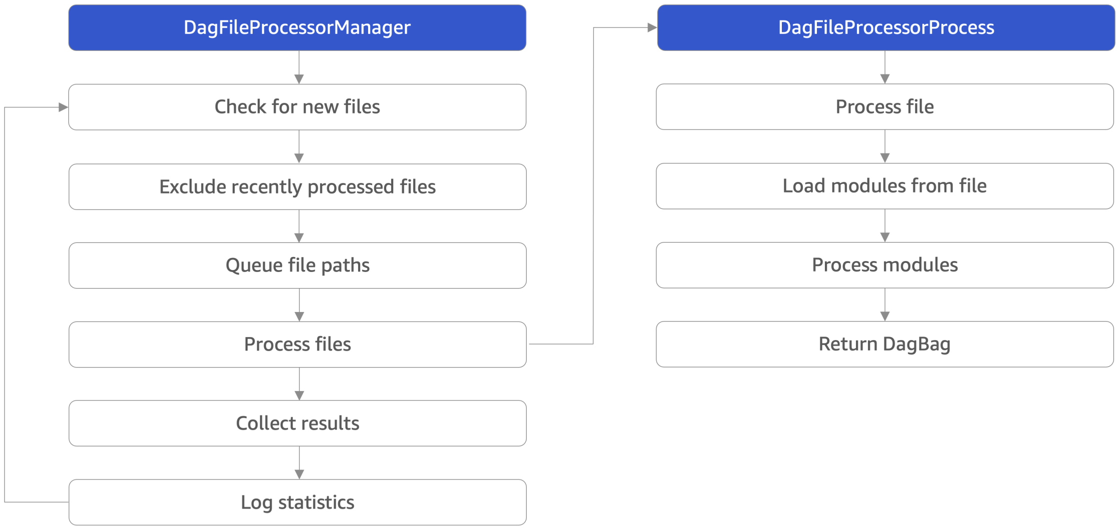
Author	@jasontr
Date	20220421
About	Airflow

How are the DAG files parsed by Airflow

- What: parse DAG files
- Who: Airflow scheduler(`DagFileProcessorManager`)
- When: `dag_dir_list_interval` defined by Airflow configuration
- Why: without the DAG files parse processing, we(user&system) can't know what dags have been defined.

HOW(According to official docs)

Parse logic of `DagFileProcessorManager`.



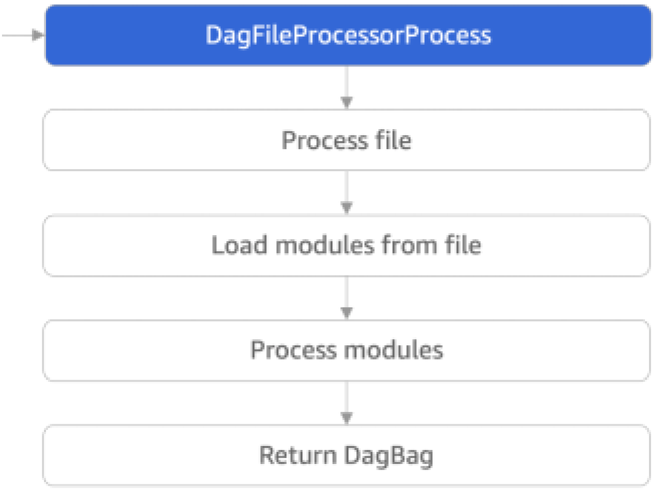
`DagFileProcessorManager` has the following steps:

1. Check for new files: If the elapsed time since the DAG was last refreshed is > dag_dir_list_interval then update the file paths list
2. Exclude recently processed files: Exclude files that have been processed more recently than min_file_process_interval and have not been modified
3. Queue file paths: Add files discovered to the file path queue
4. Process files: Start a new `DagFileProcessorProcess` for each file, up to a maximum of parsing_processes
5. Collect results: Collect the result from any finished DAG processors
6. Log statistics: Print statistics and emit `dag_processing.total_parse_time`

`DagFileProcessorProcess` has the following steps:

1. Process file: ***The entire process*** must complete within dag_file_processor_timeout (default: 50s)
2. Load modules from file: Uses Python imp command, must complete within dagbag_import_timeout (default: 30s)
3. Process modules: Find DAG objects within the Python module
4. Return DagBag: Provide the `DagFileProcessorManager` a list of the discovered DAG objects

U give me translate translate what is called THE ENTIRE PROCESS



From the flow, it seems that the `Process file` and the other three are the individual tasks

According to the source code

```
for file_path, processor in self._processors.items():
    duration = now - processor.start_time
    if duration > self._processor_timeout:
```

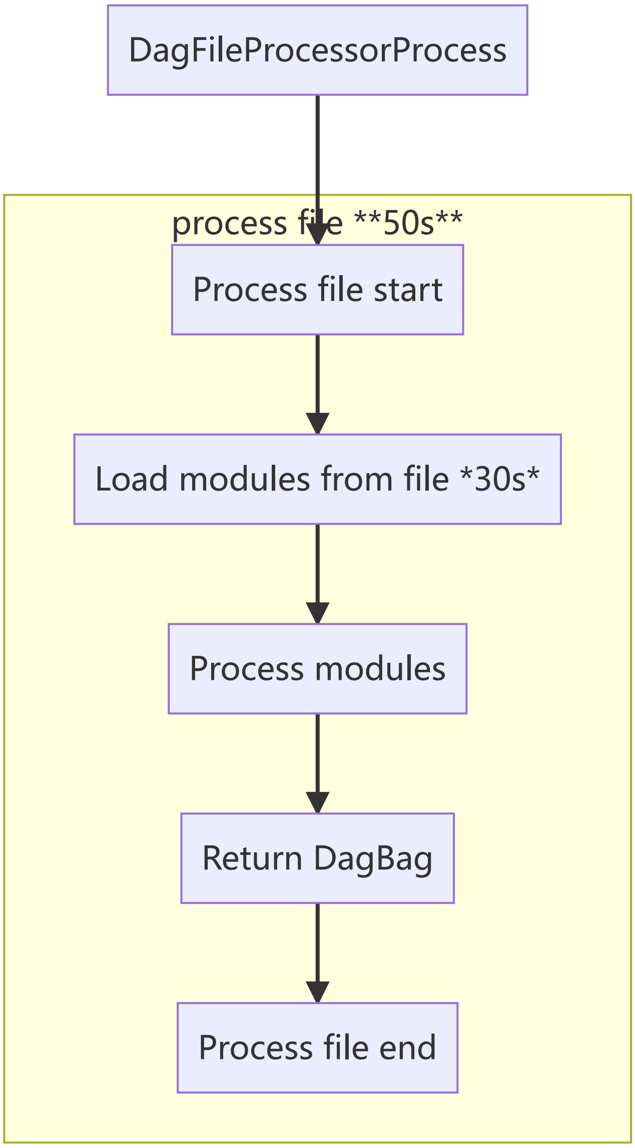
and the `self._processors`

```
self._processors: Dict[str, DagFileProcessorProcess] = {}
```

Yes! Type hint!

`dag_file_processor_timeout` is controlling the execution time of `DagFileProcessorProcess` which was created here and started at here. According to `start()` definition we can know `processor.start_time` presents the start time of the whole process of `DagFileProcess`.

So, the flow should be



The time limitation of Process modules + Return DagBag is 20s (in default).

Load modules from file? Process modules? Let's see some example.

```
#Load modules
import pendulum

from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.model import Variable

import numpy as np
#Load modules

#Process modules
with DAG(
```

```python
    dag_id="example_python_operator",
    schedule_interval=None,
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    tags=["example"],
) as dag:

    def print_array():
        """Print Numpy array."""
        a = np.arange(15).reshape(3, 5)
        print(a)
        return a

    run_this = PythonOperator(
        task_id="print_the_context" + Variable.get('task_id'),
        python_callable=print_array,
    )
#Process modules
```

What should be careful of?

Where is the heavy process

```python
#Load modules
import pendulum

from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.model import Variable

import numpy as np # <-- heavy
#Load modules

#Process modules
with DAG(
    dag_id="example_python_operator",
    schedule_interval=None,
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    tags=["example"],
) as dag:

    def print_array():
        """Print Numpy array."""
        a = np.arange(15).reshape(3, 5)
        print(a)
        return a

    run_this = PythonOperator(
        task_id="print_the_context" + Variable.get('task_id'), # <-- maybe heavy, Variable.get will create a database acess process.
        python_callable=print_array,
    )

# let's have a 20 second fun,
for i in range(1e1000):
    for j in range(1e1000):
        i*j
#Process modules
```

DAG parsing will timeout and fail

What should we do

avoid writing the top-level code which is not necessary to create Operators and build DAG relations between them

```
#Load modules
import pendulum

from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.model import Variable



#Load modules

#Process modules
with DAG(
    dag_id="example_python_operator",
    schedule_interval=None,
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
    catchup=False,
    tags=["example"],
) as dag:

    def print_array():
        import numpy as np # move import inside of python callable (load as local imports)
        """Print Numpy array."""
        a = np.arange(15).reshape(3, 5)
        print(a)
        return a

    run_this = PythonOperator(
        task_id="print_the_context" + Variable.get('task_id'), # <-- be careful with database acess duration and frequency.
        python_callable=print_array,
    )

# have fun in other way
#for i in range(1e1000):
#    for j in range(1e1000):
#        i*j
#Process modules
```

Reference

https://airflow.apache.org/docs/apache-airflow/stable/concepts/scheduler.html

https://airflow.apache.org/docs/apache-airflow/stable/configurations-ref.html

https://airflow.apache.org/docs/apache-airflow/stable/best-practices.html