Task Class Documentation

The `Task` class is a pivotal component designed for managing tasks in a sequential workflow. This class allows for the execution of tasks using various agents, which can be callable objects or specific instances of the `Agent` class. It supports the scheduling of tasks, handling their dependencies, and setting conditions and actions that govern their execution.

Key features of the `Task` class include:
- Executing tasks with specified agents and handling their results.
- Scheduling tasks to run at specified times.
- Setting triggers, actions, and conditions for tasks.
- Managing task dependencies and priorities.
- Providing a history of task executions for tracking purposes.

Class Definition

The `Task` class is defined as follows:

Attributes

Attribute	Type	Description
`agent`	`Union[Callable, Agent]`	The agent or callable object to run the task.

`description`	`str`	Description of the task.
`result`	`Any`	Result of the task.
`history`	`List[Any]`	History of the task.
`schedule_time`	`datetime`	Time to schedule the task.
`scheduler`	`sched.scheduler`	Scheduler to schedule the task.
`trigger`	`Callable`	Trigger to run the task.
`action`	`Callable`	Action to run the task.
`condition`	`Callable`	Condition to run the task.
`priority`	`int`	Priority of the task.
`dependencies`	`List[Task]`	List of tasks that need to be completed before this task can be executed.
`args`	`List[Any]`	Arguments to pass to the agent or callable object.
`kwargs`	`Dict[str, Any]`	Keyword arguments to pass to the agent or callable object.

Methods

`execute(self, *args, **kwargs)`

Executes the task by calling the agent or model with the specified arguments and keyword arguments. If a condition is set, the task will only execute if the condition returns `True`.

Parameters

- `args`: Arguments to pass to the agent or callable object.
- `kwargs`: Keyword arguments to pass to the agent or callable object.

Examples

```python
>>> from swarms.structs import Task, Agent
>>> from swarm_models import OpenAIChat
>>> agent = Agent(llm=OpenAIChat(openai_api_key=""), max_loops=1, dashboard=False)
>>> task = Task(description="What's the weather in Miami?", agent=agent)
>>> task.run()
>>> task.result
```

`handle_scheduled_task(self)`

Handles the execution of a scheduled task. If the schedule time is not set or has already passed, the task is executed immediately. Otherwise, the task is scheduled to be executed at the specified schedule time.

Examples

```python
>>> task.schedule_time = datetime.now() + timedelta(seconds=10)
>>> task.handle_scheduled_task()
```

```

### `set_trigger(self, trigger: Callable)`

Sets the trigger for the task.

#### Parameters

- `trigger` (`Callable`): The trigger to set.

#### Examples

```python
>>> def my_trigger():
>>>     print("Trigger executed")
>>> task.set_trigger(my_trigger)
```

`set_action(self, action: Callable)`

Sets the action for the task.

Parameters

- `action` (`Callable`): The action to set.

Examples
```

```python
>>> def my_action():
>>>     print("Action executed")
>>> task.set_action(my_action)
```

`set_condition(self, condition: Callable)`

Sets the condition for the task.

Parameters

- `condition` (`Callable`): The condition to set.

Examples

```python
>>> def my_condition():
>>>     print("Condition checked")
>>>     return True
>>> task.set_condition(my_condition)
```

`is_completed(self)`

Checks whether the task has been completed.

Returns

- `bool`: `True` if the task has been completed, `False` otherwise.

Examples

```python
>>> task.is_completed()
```

`add_dependency(self, task)`

Adds a task to the list of dependencies.

Parameters

- `task` (`Task`): The task to add as a dependency.

Examples

```python
>>> dependent_task = Task(description="Dependent Task")
>>> task.add_dependency(dependent_task)
```

`set_priority(self, priority: int)`

Sets the priority of the task.

Parameters

- `priority` (`int`): The priority to set.

Examples

```python
>>> task.set_priority(5)
```

`check_dependency_completion(self)`

Checks whether all the dependencies have been completed.

Returns

- `bool`: `True` if all the dependencies have been completed, `False` otherwise.

Examples

```python
>>> task.check_dependency_completion()
```

`context(self, task: "Task" = None, context: List["Task"] = None, *args, **kwargs)`

Sets the context for the task. For a sequential workflow, it sequentially adds the context of the

previous task in the list.

Parameters

- `task` (`Task`, optional): The task whose context is to be set.
- `context` (`List[Task]`, optional): The list of tasks to set the context.

Examples

```python
>>> task1 = Task(description="Task 1")
>>> task2 = Task(description="Task 2")
>>> task2.context(context=[task1])
```

Usage Examples

Basic Usage

```python
import os
from dotenv import load_dotenv
from swarms import Agent, OpenAIChat, Task

# Load the environment variables
load_dotenv()
```

```
# Define a function to be used as the action
def my_action():
    print("Action executed")

# Define a function to be used as the condition
def my_condition():
    print("Condition checked")
    return True

# Create an agent
agent = Agent(
    llm=OpenAIChat(openai_api_key=os.environ["OPENAI_API_KEY"]),
    max_loops=1,
    dashboard=False,
)

# Create a task
task = Task(
      description="Generate a report on the top 3 biggest expenses for small businesses and how businesses can save 20%",
    agent=agent,
)

# Set the action and condition
task.set_action(my_action)
task.set_condition(my_condition)
```

```
# Execute the task
print("Executing task...")
task.run()

# Check if the task is completed
if task.is_completed():
    print("Task completed")
else:
    print("Task not completed")

# Output the result of the task
print(f"Task result: {task.result}")
```

Scheduled Task Execution

```python
from datetime import datetime, timedelta
import os
from dotenv import load_dotenv
from swarms import Agent, OpenAIChat, Task

# Load the environment variables
load_dotenv()
```

```
# Create an agent
agent = Agent(
    llm=OpenAIChat(openai_api_key=os.environ["OPENAI_API_KEY"]),
    max_loops=1,
    dashboard=False,
)

# Create a task
task = Task(
    description="Scheduled task example",
    agent=agent,
    schedule_time=datetime.now() + timedelta(seconds=10)
)

# Handle scheduled task
task.handle_scheduled_task()
```

Task with Dependencies

```python
import os
from dotenv import load_dotenv
from swarms import Agent, OpenAIChat, Task

# Load the environment variables
```

```
load_dotenv()

# Create agents
agent1 = Agent(
    llm=OpenAIChat(openai_api_key=os.environ["OPENAI_API_KEY"]),
    max_loops=1,
    dashboard=False,
)
agent2 = Agent(
    llm=OpenAIChat(openai_api_key=os.environ["OPENAI_API_KEY"]),
    max_loops=1,
    dashboard=False,
)

# Create tasks
task1 = Task(description="First task", agent=agent1)
task2 = Task(description="Second task", agent=agent2)

# Add dependency
task2.add_dependency(task1)

# Execute tasks
print("Executing first task...")
task1.run()

print("Executing second task...")
```

task2.run()

Check if tasks are completed

print(f"Task 1 completed: {task1.is_completed()}")

print(f"Task 2 completed: {task2.is_completed()}")

```

### Task Context

```python

import os

from dotenv import load_dotenv

from swarms import Agent, OpenAIChat, Task

# Load the environment variables

load_dotenv()

# Create an agent

agent = Agent(

    llm=OpenAIChat(openai_api_key=os.environ["OPENAI_API_KEY"]),

    max_loops

=1,

    dashboard=False,

)

```
# Create tasks
task1 = Task(description="First task", agent=agent)
task2 = Task(description="Second task", agent=agent)

# Set context for the second task
task2.context(context=[task1])

# Execute tasks
print("Executing first task...")
task1.run()

print("Executing second task...")
task2.run()

# Output the context of the second task
print(f"Task 2 context: {task2.history}")
```