Module/Class Name: Conversation

Introduction

The `Conversation` class is a powerful tool for managing and structuring conversation data in a Python program. It enables you to create, manipulate, and analyze conversations easily. This documentation will provide you with a comprehensive understanding of the `Conversation` class, its attributes, methods, and how to effectively use it.

Table of Contents

- `return_history_as_string(self)`
- `save_as_json(self, filename: str)`
- `load_from_json(self, filename: str)`
- `search_keyword_in_conversation(self, keyword: str)`
- `pretty_print_conversation(self, messages)`

---

1. Class Definition

Overview

The `Conversation` class is designed to manage conversations by keeping track of messages and their attributes. It offers methods for adding, deleting, updating, querying, and displaying messages within the conversation. Additionally, it supports exporting and importing conversations, searching for specific keywords, and more.

Attributes

- `time_enabled (bool)`: A flag indicating whether to enable timestamp recording for messages.
- `conversation_history (list)`: A list that stores messages in the conversation.

2. Methods

`__init__(self, time_enabled: bool = False, *args, **kwargs)`

- **Description**: Initializes a new Conversation object.
- **Parameters**:
  - `time_enabled (bool)`: If `True`, timestamps will be recorded for each message. Default is `False`.

`add(self, role: str, content: str, *args, **kwargs)`

- **Description**: Adds a message to the conversation history.
- **Parameters**:
  - `role (str)`: The role of the speaker (e.g., "user," "assistant").
  - `content (str)`: The content of the message.

`delete(self, index: str)`

- **Description**: Deletes a message from the conversation history.
- **Parameters**:
  - `index (str)`: The index of the message to delete.

`update(self, index: str, role, content)`

- **Description**: Updates a message in the conversation history.
- **Parameters**:
  - `index (str)`: The index of the message to update.
  - `role (_type_)`: The new role of the speaker.
  - `content (_type_)`: The new content of the message.

`query(self, index: str)`

- **Description**: Retrieves a message from the conversation history.
- **Parameters**:
  - `index (str)`: The index of the message to query.
- **Returns**: The message as a string.

`search(self, keyword: str)`

- **Description**: Searches for messages containing a specific keyword in the conversation history.
- **Parameters**:
  - `keyword (str)`: The keyword to search for.
- **Returns**: A list of messages that contain the keyword.

`display_conversation(self, detailed: bool = False)`

- **Description**: Displays the conversation history.
- **Parameters**:
  - `detailed (bool, optional)`: If `True`, provides detailed information about each message. Default is `False`.

`export_conversation(self, filename: str)`

- **Description**: Exports the conversation history to a text file.
- **Parameters**:
  - `filename (str)`: The name of the file to export to.

`import_conversation(self, filename: str)`

- **Description**: Imports a conversation history from a text file.
- **Parameters**:
  - `filename (str)`: The name of the file to import from.

`count_messages_by_role(self)`

- **Description**: Counts the number of messages by role in the conversation.
- **Returns**: A dictionary containing the count of messages for each role.

`return_history_as_string(self)`

- **Description**: Returns the entire conversation history as a single string.
- **Returns**: The conversation history as a string.

`save_as_json(self, filename: str)`

- **Description**: Saves the conversation history as a JSON file.
- **Parameters**:
  - `filename (str)`: The name of the JSON file to save.

`load_from_json(self, filename: str)`

- **Description**: Loads a conversation history from a JSON file.
- **Parameters**:

  - `filename (str)`: The name of the JSON file to load.

`search_keyword_in_conversation(self, keyword: str)`

- **Description**: Searches for a keyword in the conversation history and returns matching messages.
- **Parameters**:
  - `keyword (str)`: The keyword to search for.
- **Returns**: A list of messages containing the keyword.

`pretty_print_conversation(self, messages)`

- **Description**: Pretty prints a list of messages with colored role indicators.
- **Parameters**:
  - `messages (list)`: A list of messages to print.

Examples

Here are some usage examples of the `Conversation` class:

Creating a Conversation

```python
from swarms.structs import Conversation

conv = Conversation()
```

```

### Adding Messages

```python
conv.add("user", "Hello, world!")
conv.add("assistant", "Hello, user!")
```

Displaying the Conversation

```python
conv.display_conversation()
```

Searching for Messages

```python
result = conv.search("Hello")
```

Exporting and Importing Conversations

```python
conv.export_conversation("conversation.txt")
conv.import_conversation("conversation.txt")
```
```

```

### Counting Messages by Role

```python
counts = conv.count_messages_by_role()
```

Loading and Saving as JSON

```python
conv.save_as_json("conversation.json")
conv.load_from_json("conversation.json")
```

Certainly! Let's continue with more examples and additional information about the `Conversation` class.

Querying a Specific Message

You can retrieve a specific message from the conversation by its index:

```python
message = conv.query(0)  # Retrieves the first message
```
```

Updating a Message

You can update a message's content or role within the conversation:

```python
conv.update(0, "user", "Hi there!")  # Updates the first message
```

Deleting a Message

If you want to remove a message from the conversation, you can use the `delete` method:

```python
conv.delete(0)  # Deletes the first message
```

Counting Messages by Role

You can count the number of messages by role in the conversation:

```python
counts = conv.count_messages_by_role()
# Example result: {'user': 2, 'assistant': 2}
```

Exporting and Importing as Text

You can export the conversation to a text file and later import it:

```python
conv.export_conversation("conversation.txt")  # Export
conv.import_conversation("conversation.txt")  # Import
```

Exporting and Importing as JSON

Conversations can also be saved and loaded as JSON files:

```python
conv.save_as_json("conversation.json")  # Save as JSON
conv.load_from_json("conversation.json")  # Load from JSON
```

Searching for a Keyword

You can search for messages containing a specific keyword within the conversation:

```python
results = conv.search_keyword_in_conversation("Hello")
```

Pretty Printing

The `pretty_print_conversation` method provides a visually appealing way to display messages with colored role indicators:

```python
conv.pretty_print_conversation(conv.conversation_history)
```

These examples demonstrate the versatility of the `Conversation` class in managing and interacting with conversation data. Whether you're building a chatbot, conducting analysis, or simply organizing dialogues, this class offers a robust set of tools to help you accomplish your goals.

Conclusion

The `Conversation` class is a valuable utility for handling conversation data in Python. With its ability to add, update, delete, search, export, and import messages, you have the flexibility to work with conversations in various ways. Feel free to explore its features and adapt them to your specific projects and applications.

If you have any further questions or need additional assistance, please don't hesitate to ask!