Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт цифрового развития
Кафедра инфокоммуникаций

РЕФЕРАТ

На тему: «Паттерн Посетитель в обработке естественного языка: анализ и манипуляции с текстовыми данными»

Выполнил:
Богдан Александр Анатольевич
3 курс, группа ИВТ-б-о-21-1,
09.03.01 – Информатика и вычислительная техника, профиль
(профиль)
09.03.01 – Информатика и вычислительная техника, профиль «Автоматизированные системы обработки информации и управления», очная форма обучения

_____
(подпись)

Проверил:
Воронкин Р.А., канд. тех. наук, доцент,
доцент кафедры инфокоммуникаций
Института цифрового развития,

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2023 г.

Оглавление

ВВЕДЕНИЕ

Структурные паттерны программирования предоставляют эффективные решения для построения сложных структур программного обеспечения. Одним из таких паттернов является Посетитель (Visitor), который играет ключевую роль в разделении абстракции от реализации, повышая гибкость и улучшая структуру кода. Паттерн Посетитель предоставляет механизм обхода структуры объектов и применения операций к каждому из них, не изменяя сами объекты.

Актуальность: Современные программные проекты сталкиваются с необходимостью обработки и анализа больших объемов текстовых данных, таких как естественные языковые тексты. Развитие технологий в области обработки естественного языка (Natural Language Processing, NLP) поднимает важность эффективных паттернов, способных обеспечить гибкость и масштабируемость кода.

Цель: Подробно изучить паттерн Посетитель, его сущности и механизмы действия, а также рассмотреть его применение в контексте обработки текстовых данных с использованием естественного языка.

Задачи:

• Анализ проблем обработки текстовых данных без паттерна Посетитель: Изучить сценарии, где отсутствие Посетителя может затруднить обработку и анализ текстовых данных, усложняя добавление новых функциональностей и поддержку проекта.

• Рассмотрение структуры и примеров использования Посетителя: Разобраться с основной структурой паттерна Посетитель и предоставить примеры его применения в реальных проектах, особенно в контексте обработки текстовых данных.

• Применение в области обработки естественного языка: Исследовать, как паттерн Посетитель может быть применен в различных областях

обработки текстовых данных, включая анализ естественного языка, выделение ключевых фраз, определение тональности текста и т. д.1.

ОСНОВНЫЕ ПОНЯТИЯ И ПАТТЕРНЫ ПРОЕКТИРОВАНИЯ

2.1 Паттерны проектирования. Общее описание паттерна посетитель

Паттерн Посетитель (Visitor) представляет собой структурный паттерн проектирования, который позволяет определить новую операцию, не изменяя классы объектов, над которыми эта операция выполняется. Посетитель позволяет добавлять новые функциональности к объектам без необходимости модификации их кода.

Основная идея паттерна заключается в разделении структуры объектов и операций, которые выполняются над этой структурой. Вместо того чтобы добавлять новые методы или изменять существующие в классах объектов, Посетитель предлагает создать отдельный класс, представляющий операцию, и передать его в объекты для их обработки. Таким образом, объекты "принимают посетителя" и позволяют ему выполнить необходимые операции.

Основные участники паттерна Посетитель:

1. **Посетитель (Visitor):** Определяет интерфейс с методами, соответствующими различным типам объектов, которые могут быть посещены.
2. **Конкретный посетитель (ConcreteVisitor):** Реализует интерфейс посетителя и предоставляет конкретную реализацию операций для каждого типа объекта.
3. **Объект (Element):** Определяет интерфейс для объектов, которые могут быть посещены посетителем.

4. **Конкретный объект (ConcreteElement):** Реализует интерфейс объекта и предоставляет конкретную реализацию метода, который принимает посетителя.
5. **Структура (ObjectStructure):** Содержит коллекцию объектов, которые могут быть посещены посетителем. Предоставляет метод для выполнения операций посетителя над всей коллекцией объектов.

Применение паттерна Посетитель упрощает добавление новых операций к существующей структуре объектов, делая код более гибким, расширяемым и поддерживаемым. Он особенно полезен, когда необходимо проводить различные операции над группой объектов, не изменяя их классы.

2.2 Принцип работы паттерна посетитель и его основные компоненты

Принцип работы паттерна Посетитель основан на разделении структуры объектов и операций, которые могут быть выполнены над этой структурой. Паттерн вводит две основные иерархии классов: иерархию объектов, которые могут быть посещены (Element), и иерархию посетителей (Visitor), которые определяют операции, выполняемые над этими объектами.

Основные компоненты паттерна Посетитель:

1. **Интерфейс Посетителя (Visitor Interface):** Определяет набор методов, каждый из которых соответствует конкретному типу объекта, который посетитель может обработать. Эти методы обычно имеют сигнатуру, включающую объект типа Element.

```
# Интерфейс Посетителя
class Visitor(ABC):
    @abstractmethod
    def visit_concrete_element_a(self, element_a):
        pass
```

```
    @abstractmethod
    def visit_concrete_element_b(self, element_b):
        pass
```

2. **Конкретный посетитель (Concrete Visitor):** Реализует интерфейс посетителя и предоставляет конкретную реализацию каждого метода для обработки различных типов объектов.

```
# Конкретный посетитель
class ConcreteVisitor(Visitor):
    def visit_concrete_element_a(self, element_a):
        print(f"Visitor is processing ConcreteElementA: {element_a.operation_a()}")

    def visit_concrete_element_b(self, element_b):
        print(f"Visitor is processing ConcreteElementB: {element_b.operation_b()}")
```

3. **Интерфейс Объекта (Element Interface):** Определяет метод accept, который принимает посетителя в качестве аргумента. Этот метод вызывает соответствующий метод посетителя, передавая себя в качестве аргумента.

```
# Интерфейс Объекта
class Element(ABC):
    @abstractmethod
    def accept(self, visitor: Visitor):
        pass
```

4. **Конкретный объект (Concrete Element):** Реализует интерфейс объекта и предоставляет конкретную реализацию метода accept, который вызывает соответствующий метод посетителя для данного типа объекта.

```
# Конкретный объект
class ConcreteElementA(Element):
    def accept(self, visitor: Visitor):
        visitor.visit_concrete_element_a(self)
```

```
    def operation_a(self):
        return "Operation A"
```

5. **Структура объектов (Object Structure):** Содержит коллекцию объектов, которые могут быть посещены. Предоставляет метод для выполнения операций посетителя над всей коллекцией объектов.

```
# Структура объектов
class ObjectStructure:
    def __init__(self):
        self.elements: List[Element] = []

    def add_element(self, element: Element):
        self.elements.append(element)

    def accept_visitor(self, visitor: Visitor):
        for element in self.elements:
            element.accept(visitor)
```

Используя паттерн Посетитель, объекты не зависят от конкретных операций, которые могут быть выполнены над ними. Вместо этого они принимают посетителя и передают себя в метод посетителя для выполнения соответствующей операции. Это обеспечивает гибкость и расширяемость системы, позволяя легко добавлять новые операции, не изменяя классы объектов.

2.3 Применимость паттерна Посетитель в обработке естественного языка

Применимость паттерна Посетитель в обработке естественного языка (NLP) обусловлена необходимостью эффективного анализа и манипуляций с текстовыми данными, такими как предложения, слова и фразы. Паттерн Посетитель вносит структурированный и гибкий подход к обработке текста,

разделяя аспекты структуры текста от операций, которые могут быть выполнены над этой структурой.

В контексте NLP, элементы текста могут быть абстрагированы в виде объектов, реализующих общий интерфейс **Element**. Этот интерфейс определяет метод **accept**, который принимает посетителя, тем самым предоставляя механизм для выполнения операций над этим элементом.

Посетители (классы, реализующие интерфейс **Visitor**) представляют собой конкретные операции, которые могут быть выполнены над текстовыми элементами. Например, посетитель может производить анализ синтаксиса, извлечение ключевых слов, определение тональности и другие операции, специфичные для NLP.

Структура текста может быть абстрагирована через объект, который содержит коллекцию текстовых элементов. Этот объект также реализует интерфейс **Element**, позволяя посетителю применять операции ко всей структуре текста.

Преимущества паттерна Посетитель в области NLP включают в себя возможность добавления новых операций над текстом, не изменяя классы элементов. Это обеспечивает легкость расширения функциональности системы обработки текста, а также обеспечивает четкую структуру, упрощающую сопровождение и поддержку кода.

3. РЕАЛИЗАЦИЯ ПАТТЕРНА ПОСЕТИТЕЛЬ

3.1 Основные вопросы реализации в обработке естественного языка

При реализации паттерна Посетитель в обработке естественного языка возникают несколько основных вопросов, которые следует учитывать для

эффективной и гибкой обработки текстовых данных (далее будут представлены примеры на псевдокоде):

Структура текста: Какие элементы текста будут рассматриваться как объекты для обработки? Например, это могут быть предложения, слова, фразы, абзацы и так далее. Определение структуры текста важно для того, чтобы определить объекты, с которыми будет взаимодействовать посетитель.

```
class TextElement:
    method accept(visitor)
        // Пустой метод, будет переопределен в дочерних классах

class Sentence(TextElement):
    method accept(visitor)
        visitor.visit_sentence(self)

class Word(TextElement):
    method accept(visitor)
        visitor.visit_word(self)
```

Виды операций: Какие операции необходимо выполнить над текстовыми элементами? Это могут быть анализ синтаксиса, извлечение ключевых слов, анализ тональности, выделение сущностей, определение частей речи и другие. Определение различных видов операций важно для определения методов в интерфейсе посетителя.

```
interface NLPVisitor:
    method visit_sentence(sentence: Sentence)
    method visit_word(word: Word)

class KeywordExtractor implements NLPVisitor:
    method visit_sentence(sentence: Sentence)
```

```
        // Реализация извлечения ключевых слов из предложения
    method visit_word(word: Word)
        // Реализация извлечения ключевых слов из слова

class SyntaxAnalyzer implements NLPVisitor:
    method visit_sentence(sentence: Sentence)
        // Реализация анализа синтаксиса предложения
    method visit_word(word: Word)
        // Реализация анализа синтаксиса слова
```

Гибкость системы: Насколько гибкой и расширяемой является система обработки текста? Паттерн Посетитель позволяет легко добавлять новые операции и обрабатывать различные элементы текста, не изменяя существующий код. Таким образом, важно учитывать гибкость системы при её проектировании и реализации.

```
// Паттерн Посетитель позволяет добавлять новые операции, не изменяя существующий код

// Предположим, у нас появляется новая операция для обработки текста
class NewOperation implements NLPVisitor:
    // Реализация новой операции
```

Уровень абстракции: Какой уровень абстракции выбрать для текстовых элементов и операций? Например, абстрактные классы или интерфейсы могут быть использованы для определения общего поведения элементов и операций, облегчая добавление новых типов элементов или операций в будущем.

```
// Абстрактный класс или интерфейс для элементов текста и посетителей позволяет определить общее поведение
```

```
interface TextElement:
    method accept(visitor: NLPVisitor)

class Sentence implements TextElement:
    method accept(visitor: NLPVisitor)
        visitor.visit_sentence(self)

class Word implements TextElement:
    method accept(visitor: NLPVisitor)
        visitor.visit_word(self)
```

Учитывая эти вопросы при реализации паттерна Посетитель в обработке естественного языка, можно создать гибкую, расширяемую и эффективную систему для анализа и манипуляций с текстовыми данными.

3.2 Реализация паттерна посетитель в Python

```
# Конкретный элемент текста - предложение
class Sentence:
    def __init__(self, content):
        self.content = content

    def accept(self, visitor):
        visitor.visit_sentence(self)

# Конкретный элемент текста - слово
```

```
class Word:
    def __init__(self, content):
        self.content = content

    def accept(self, visitor):
        visitor.visit_word(self)

# Конкретный посетитель - извлекает два самых длинных слова из предложений
class KeywordExtractor:
    def __init__(self):
        self.keywords = []

    def visit_sentence(self, sentence):
        # Извлекаем все слова из предложения
        words = sentence.content.split()
        # Сортируем слова по длине в порядке убывания
        words.sort(key=lambda x: len(x), reverse=True)
        # Извлекаем два самых длинных слова и добавляем их в список ключевых слов
        if len(words) >= 2:
            self.keywords.append(words[0])
            self.keywords.append(words[1])

    def visit_word(self, word):
        pass

# Пример использования
if __name__ == "__main__":
    # Создаем текстовые элементы (предложения и слова)
```

```
elements = [
    Sentence("Python is a popular programming language."),
    Sentence("It is known for its simplicity and versatility."),
     Sentence("Python is widely used in various fields such as web development, data science, and artificial intelligence.")
]

# Создаем посетителя для извлечения ключевых слов
keyword_extractor = KeywordExtractor()

# Применяем посетителя к каждому элементу текста
for element in elements:
    element.accept(keyword_extractor)

# Выводим извлеченные ключевые слова
print("Extracted keywords:")
for keyword in keyword_extractor.keywords:
    print(keyword)
```

3.3 Использования паттерна посетитель в других языках программирования

Пример применения паттерна на языке Java:

```
// Интерфейс для посетителя
interface Visitor {
    void visit(Sentence sentence);
    void visit(Word word);
}

// Абстрактный элемент текста
```

```
interface TextElement {
    void accept(Visitor visitor);
}

// Конкретный элемент текста - предложение
class Sentence implements TextElement {
    private String content;

    public Sentence(String content) {
        this.content = content;
    }

    public String getContent() {
        return content;
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }
}

// Конкретный элемент текста - слово
class Word implements TextElement {
    private String content;

    public Word(String content) {
```

```
        this.content = content;
    }

    public String getContent() {
        return content;
    }

    @Override
    public void accept(Visitor visitor) {
        visitor.visit(this);
    }
}

// Конкретный посетитель - извлекает два самых длинных слова из предложений
class KeywordExtractor implements Visitor {
    private List<String> keywords = new ArrayList<>();

    public List<String> getKeywords() {
        return keywords;
    }

    @Override
    public void visit(Sentence sentence) {
        String[] words = sentence.getContent().split("\\s+");
        Arrays.stream(words)
                .sorted(Comparator.comparingInt(String::length).reversed())
```

```
                .limit(2)
                .forEach(keywords::add);
        }

        @Override
        public void visit(Word word) {
            // Не используется
        }
    }

    // Пример использования
    public class Main {
        public static void main(String[] args) {
            List<TextElement> elements = Arrays.asList(
                new Sentence("Java is a versatile programming language."),
                new Sentence("It is widely used in web development and software engineering.")
            );

            KeywordExtractor keywordExtractor = new KeywordExtractor();
            elements.forEach(element -> element.accept(keywordExtractor));

            System.out.println("Extracted keywords:");
            keywordExtractor.getKeywords().forEach(System.out::println);
        }
    }
```

Пример применения паттерна на языке JavaScript:

```
// Интерфейс для посетителя
class Visitor {
    visitSentence(sentence) {}
    visitWord(word) {}
}

// Абстрактный элемент текста
class TextElement {
    accept(visitor) {}
}

// Конкретный элемент текста - предложение
class Sentence extends TextElement {
    constructor(content) {
        super();
        this.content = content;
    }

    accept(visitor) {
        visitor.visitSentence(this);
    }
}

// Конкретный элемент текста - слово
class Word extends TextElement {
    constructor(content) {
        super();
        this.content = content;
    }

    accept(visitor) {
        visitor.visitWord(this);
    }
}

// Конкретный посетитель - извлекает два самых длинных слова из
предложений
class KeywordExtractor extends Visitor {
    constructor() {
        super();
        this.keywords = [];
```

```
    }

    getKeywords() {
        return this.keywords;
    }

    visitSentence(sentence) {
        const words = sentence.content.split(' ').sort((a, b) => b.length - a.length);
        this.keywords.push(words[0], words[1]);
    }

    visitWord(word) {
        // Не используется
    }
}

// Пример использования
const elements = [
    new Sentence("JavaScript is a popular scripting language."),
    new Sentence("It is commonly used in web development.")
];

const keywordExtractor = new KeywordExtractor();
elements.forEach(element => element.accept(keywordExtractor));

console.log("Extracted keywords:");
console.log(keywordExtractor.getKeywords());
```

4. ПРЕИМУЩЕСТВА И НЕДОСТАТКИ ИСПОЛЬЗОВАНИЯ ПАТТЕРНА ПОСЕТИТЕЛЬ

4.1 Преимущества паттерна Посетитель в обработке естественного языка

1. Разделение алгоритмов обработки и структуры текста:

Паттерн посетитель позволяет выделить алгоритмы обработки текста в отдельные классы посетителей, что обеспечивает разделение логики обработки текста от его структуры.

Это улучшает читаемость кода и упрощает поддержку, так как изменения в алгоритмах обработки не затрагивают структуру текста и наоборот.

2. Гибкость и расширяемость:

Паттерн посетитель позволяет добавлять новые операции обработки текста, не изменяя классы текстовых элементов.

Это обеспечивает гибкость и расширяемость системы, позволяя легко вносить изменения и добавлять новую функциональность без модификации существующего кода.

3. Уменьшение связанности:

Посетитель позволяет абстрагироваться от деталей обработки текста, что уменьшает связанность между алгоритмами обработки и структурой текста.

Это делает код более модульным и упрощает его понимание, а также уменьшает вероятность ошибок при изменениях.

4. Поддержка различных операций:

Паттерн посетитель позволяет легко добавлять новые операции обработки текста, не затрагивая существующие операции.

Это обеспечивает поддержку различных видов анализа и обработки текста, таких как извлечение ключевых слов, анализ синтаксиса, морфологический анализ и т.д.

5. Улучшенная поддержка тестирования:

Разделение алгоритмов обработки текста и структуры текста упрощает тестирование, так как изменения в одной части системы не влияют на другую.

Это обеспечивает изоляцию модулей и улучшает поддержку тестирования и отладки кода.

6. Обеспечение гибкой архитектуры:

Паттерн посетитель способствует созданию гибкой архитектуры, позволяя легко комбинировать различные операции обработки текста и структуры текста.

Это обеспечивает возможность создания различных конфигураций обработки текста в зависимости от требований приложения или пользователя.

4.2 Недостатки паттерна Посетитель в обработке естественного языка

1. Множество обновлений:

При использовании паттерна посетитель необходимо обновлять каждого посетителя каждый раз, когда добавляется или удаляется класс из основной иерархии. Это может стать проблемой в больших проектах, где существует множество различных операций обработки, и изменения в структуре данных могут быть частыми.

Необходимость в постоянном обновлении посетителей может увеличить время и усилия, затрачиваемые на поддержку кода, и повлиять на его стабильность.

2. Трудность продления:

Если количество классов-посетителей слишком велико, то становится сложно продлить весь интерфейс класса. Это может привести к увеличению сложности и увеличению времени разработки новых функциональностей.

При большом количестве классов-посетителей может возникнуть проблема управления зависимостями между классами и увеличения связанности, что усложняет разработку и поддержку кода.

3. Отсутствие доступа:

Иногда у посетителей может отсутствовать доступ к закрытым полям определенных классов, с которыми они должны работать. Это может быть вызвано тем, что классы, представляющие структуру текста, скрывают свои внутренние детали, что делает невозможным доступ к ним извне.

Отсутствие доступа к закрытым полям может привести к необходимости изменения структуры классов, что может нарушить инкапсуляцию и увеличить связанность между классами, что нежелательно.

ЗАКЛЮЧЕНИЕ

Паттерн посетитель представляет собой эффективный инструмент в обработке естественного языка, позволяя разделить алгоритмы обработки текста от его структуры. Этот паттерн обеспечивает гибкость и расширяемость системы, позволяя добавлять новые операции обработки текста без изменения существующего кода. Он также уменьшает связанность между компонентами системы, что способствует ее обслуживаемости и тестируемости.

Применение паттерна посетитель особенно полезно в сценариях, где необходимо поддерживать различные виды анализа и обработки текста, такие как извлечение ключевых слов, анализ синтаксиса, морфологический анализ и другие. Этот подход также улучшает безопасность, так как клиентский код имеет ограниченный доступ к внутренним деталям реализации.

Осознанное применение паттерна посетитель в обработке естественного языка позволяет создавать более гибкие, поддерживаемые и расширяемые системы, способствуя повышению эффективности и качества разработки программного обеспечения.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. 2. Э. Гамма, Р. Хелм, Р. Джонсон, Дж. Влиссидес Приемы объектно-ориентированного проектирования. Паттерны проектирования = Design Patterns: Elements of Reusable Object- Oriented Software. — СПб: «Питер», 2007. — С. 366.

2. Паттерны проектирования на платформе .NET. — СПб.: Питер, 2015. — 320 с.: ил. ISBN 978-5-496-01649-0.

3. 3. Марк Гранд Шаблоны проектирования в JAVA. Каталог популярных шаблонов проектирования, проиллюстрированных при помощи UML = Patterns in Java, Volume 1. A Catalog of Reusable Design Patterns Illustrated with UML. — М.: «Новое знание», 2004. — С. 560.

4. 4. Крэг Ларман Применение UML 2.0 и шаблонов проектирования = Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. — М.: «Вильямс», 2006. — С. 736.

5. 5. Джошуа Кериевски Рефакторинг с использованием шаблонов (паттернов проектирования) = Refactoring to Patterns (Addison-Wesley Signature Series). — М.: «Вильямс», 2006. — С. 400.

6. Швец Александр Погружение в паттерны проектирования. Design patterns. Explained simply — 2018.

7. Шаблоны проектирования по-человечески: структурные паттерны (proglib.io)

8. О структурных шаблонах проектирования простым языком (tproger.ru)