20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Kishanth K
CB.EN.U4CYS21031
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

Outline

Reactive Programming Paradigm

Reactive programming is a programming paradigm that focuses on constructing responsive and robust soft- ware applications that can handle asynchronous data streams and change propagation, allowing developers to create scalable and more easily maintainable applications that can adapt to a dynamic environment.

The reactive programming paradigm is particularly useful for developing applications that require real- time updates, such as user interfaces, web applications, and distributed systems.

The Main Key Features of Reactive Programming Paradigm

- Data Streams
- Observables
- Observers
- Operators
- Declarative Programming
- Backpressure Handling

Key Features of Reactive Programming Paradigm

- Asynchronous Data Flow
- Event-Driven Architecture
- Concurrency and Parallelism
- Testability
- Automatic Change Propagation

Use Cases of Reactive Programming Paradigm :

- IOT Applications
- Network Applications
- Highly Interface User to User Handling
- Foreground and Background Applications
- AWS Cloud Processing

Reactive Programming Paradigm

Popular frameworks and libraries that support reactive programming:

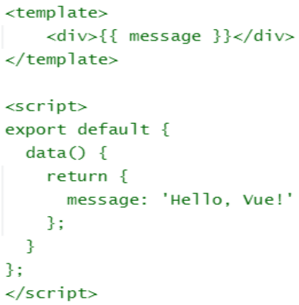
- **RxJava/RxJS/Rx.NET:**
  - Reactive Extensions (Rx) is a set of libraries for various programming languages that brings reactive programming concepts. RxJava is for Java, RxJS is for JavaScript, and Rx.NET is for .NET languages.
- **Reactor:**
  - A reactive programming library for building non-blocking applications on the Java Virtual Machine. It is often used in conjunction with the Spring Framework.
- **Akka Streams:**
  - Part of the Akka toolkit for building concurrent and distributed applications in Scala and Java. It provides a high-level API for working with reactive streams.

Reactive Programming Paradigm - Vue.js

- Reactive programming is a paradigm that deals with asynchronous data streams and the propagation of changes. Vue.js, a JavaScript framework for building user interfaces, embraces reactive programming to make it easier to handle dynamic and reactive data.
- In Vue.js, the reactivity system revolves around the use of a reactive data object, which is typically created using the data function in a component. When the da changes, the associated parts of the UI automatically update to reflect those changes.

Features related to reactive programming in Vue.js

- **Data Reactivity** In Vue.js, when you declare data properties within a component, Vue automatically makes them reactive. This means that any changes to these properties will trigger updates to any part of the user interface that depends on them.



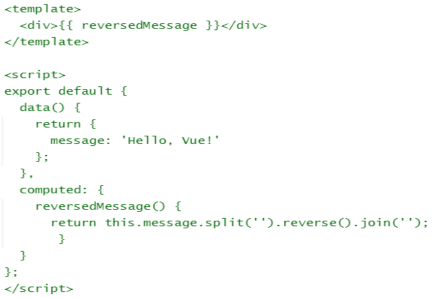
```
<template>
    <div>{{ message }}</div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Hello, Vue!'
    };
  }
};
</script>
```

- When message changes, the corresponding part of the Document Object Model is automatically updated.

Features related to reactive programming in Vue.js

- **Computed Properties:** Vue allows the creation of computed properties, which are reactive and automatically update when their dependencies change. Computed properties are useful for performing calculations based on reactive data.



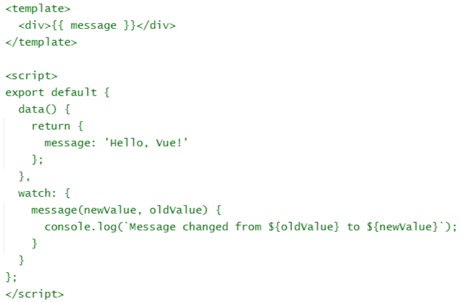
```
<template>
  <div>{{ reversedMessage }}</div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Hello, Vue!'
    };
  },
  computed: {
    reversedMessage() {
      return this.message.split('').reverse().join('');
       }
  }
};
</script>
```

- Here, reversed Message depends on message, and it updates automatically when message changes.

Features related to reactive programming in Vue.js

- **Watchers:** Watchers allow you to perform custom logic in response to changes in a specific data property. This is useful when you need to react to changes with more complex or asynchronous logic.



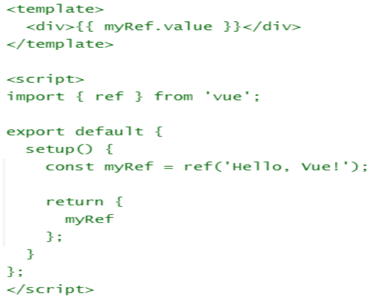
```
<template>
  <div>{{ message }}</div>
</template>

<script>
export default {
  data() {
    return {
      message: 'Hello, Vue!'
    };
  },
  watch: {
    message(newValue, oldValue) {
      console.log(`Message changed from ${oldValue} to ${newValue}`);
    }
  }
};
</script>
```

- The watch option is used to define a watcher for the message property.Watchers in Vue.js allow you to respond to changes in specific data properties. When the watched data changes, you can perform custom actions.

Features related to reactive programming in Vue.js

- **Reactive APIs (Composition API):** With the introduction of the Composition API in Vue.js 3, developers have more fine-grained control over reactivity. The ref and reactive functions are part of the Composition API and can be used to create reactive references and objects, respectively.



```
<template>
  <div>{{ myRef.value }}</div>
</template>

<script>
import { ref } from 'vue';

export default {
  setup() {
    const myRef = ref('Hello, Vue!');

    return {
      myRef
    };
  }
};
</script>
```

- In this example, myRef is a reactive reference created with ref.

Features realted to reactive programming with vue.js

Other Features of Reactive Programming with Vue.js:

- **Directives and Reactive Behavior:** Vue.js introduces directives like v-bind and v-model that facilitate reactive behavior. For example, 'v-bind' allows you to bind an attribute to a data property, and changes to the data property automatically update the associated attribute in the DOM.
- **Event Handling:** Vue.js makes it easy to handle user interactions and events. You can use the v-on directive to listen for events and execute methods in response. This aligns with the reactive programming paradigm, where components respond to events and changes.
- **Vue Router and Vuex:** Vue Router (for routing) and Vuex (for state management) also incorporate reactive patterns. Vue Router allows for reactive navigation, and Vuex provides a reactive state management system that helps manage and update shared state in a Vue.js application.
- **Reactivity in Templates:** Vue's template syntax is designed to be reactive. You can directly use data properties, computed properties, and methods in the template, and the DOM updates automatically when these values change.

Aspect-Oriented Programming

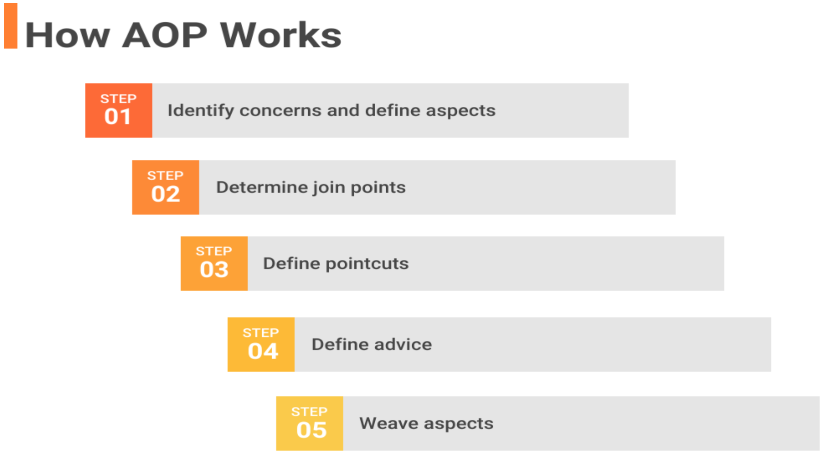
- Aspect-Oriented Programming (AOP) is a programming paradigm that provides a modular approach to software design by separating cross-cutting concerns, such as logging, security, and transaction management, from the core business logic. AOP aims to improve modularity and maintainability by addressing the challenges associated with the scattered and tangled nature of cross-cutting concerns in traditional object-oriented programming.
- Let us take an example to understand about AOP, Imagine you have a program for handling bank transactions. Your main code focuses on transferring money and checking balances. Logging every transaction could be a cross-cutting concern. With AOP, you can have a separate logging aspect. It means your main code stays clear of logging details, and the logging aspect takes care of it.
- In simpler terms, AOP is like having a neat way to deal with additional tasks that are not the main focus of your program. It helps keep your main code tidy and organized.

Aspect-Oriented Programming

Key Concepts of Aspect Oriented Programming:

- **Aspect:**
  - Modular units that encapsulate cross-cutting concerns.
  - Contain advice (code to be injected) and pointcuts (patterns for identifying join points).
- **Join Points**:
  - Specific points in the execution of a program where advice can be applied (e.g., method calls, field accesses, object creations).
- **Pointcuts:**
  - Expressions that identify join points of interest, allowing for precise control over where advice is applied.
- **Advice:**
  - Code that is triggered at specified join points, often modifying the behavior of the original code.
  - Types of advice include before, after, after returning, after throwing, and around.
- **Weaving:**
  - The process of integrating aspects into the main code, either at compile time, time, or runtime.

Aspect-Oriented Programming



How AOP Works

STEP 01 Identify concerns and define aspects

STEP 02 Determine join points

STEP 03 Define pointcuts

STEP 04 Define advice

STEP 05 Weave aspects

Aspect-Oriented Programming

Prominent AOP Frameworks:

- Spring AOP: Helps organize code in Spring apps, doing tasks like before or after certain actions.
- AspectJ: Powerful tool for flexible code organization, letting you combine code pieces in various ways.
- JBoss AOP: Tool within JBoss server, aids with tasks like actions before or after in your code.
- Guice AOP: Part of Google Guice, organizes code by intercepting actions without changing the code itself.
- PostSharp: .NET tool for efficient code organization, handling tasks like actions before, after, or around certain operations.

Aspect-Oriented Programming - Aspect C++

- AspectC++ is an extension of the programming language that bring Aspect -Oriented Programming concepts into C++. AOP aims to modularize cross-cutting concerns that affect multiple parts of a program, where OOP focuses on encapsulating behavior within classes.
- The importance of AspectC++ relies on managing the complexity of the large code.AspectC++ address this by offering much efficient way which is better that OOP.

Aspect-Oriented Programming - Aspect C++

Example Aspect C++ Code:

```
#include <iostream>

// Define a simple class
class MyClass {
public:
    void myMethod() {
        std::cout << "Executing myMethod" << std::endl;
    }
};

// Define an aspect
aspect LoggingAspect {
    // Define a pointcut that matches the execution of myMethod
    pointcut logPoints() : execution(void MyClass::myMethod());

    // Define an advice that runs before the execution of myMethod
    advice logAdvice() : before() && logPoints() {
        std::cout << "Logging before myMethod call" << std::endl;
    }
};

int main() {
    // Create an instance of MyClass
    MyClass obj;

    // Call myMethod
    obj.myMethod();

    return
```

Aspect-Oriented Programming - Aspect C++

Code Explanation:
Class Definition:
There is a simple C++ class named MyClass with a public method called myMethod. myMethod prints a message indicating its execution.
Aspect Definition:
An aspect named LoggingAspect is defined to encapsulate cross-cutting concerns related to logging.
Pointcut Definition:
A pointcut named logPoints is defined to match the execution of the myMethod within MyClass.
Advice Definition:
An advice named logAdvice is defined to run before the execution of myMethod. This advice prints a log message indicating that logging is performed before calling myMethod.
Main Function:
In the main function, an instance of MyClass named obj is created.
Method Invocation:
The myMethod of the obj instance is invoked.
AspectC++ Behavior:
Due to the defined aspect, the logAdvice runs before the execution of myMethod. As a result, you'll see additional log messages indicating the logging behavior.

Analysis of Reactive Programming Paradigm

Strengths

1. Asynchronous and Non-blocking
2. Responsive User Interfaces
3. Declarative Approach
4. Event-Driven Programming
5. Composability and Reusability

Weakness

- Learning Curve
- Debugging Complexity
- Potential for Overuse
- Resource Consumption
- Error Handling

Notable Features of Reactive Programming

- Asynchrony
- Event Handling
- Data Streams
- Observables
- Observers and Subscribers
- Declarative Programming
- Composition of Operations
- Immutability
- Backpressure Handling
- Hot and Cold Observables
- Functional Programming Concepts
- Error Handling

Analysis of Aspected Oriented Programming

Strengths

1. Modularity and Separation of Concerns
2. Code Reusability
3. Improved Code Organization
4. Enhanced Maintainability
5. Observers and Subscribers
6. Cross-Cutting Concerns Management
7. Aspect Reusability Across Projects

Weakness

- Learning Curve
- Potential for Abstraction Overuse
- Limited Tool Support
- Debugging Challenges

Notable Features of Aspect Oriented Programming Paradigm

- Join Points
- Pointcuts
- Advice
- Aspects
- Weaving
- Aspect Libraries and Frameworks
- Aspect Inheritance and Composition
- Dynamic Aspect Activation
- Meta-Programming and Reflection
- Cross-Cutting Concern Examples

Comparsions and Similarities

Aspect	Reactive Programming (RP)	Aspect-Oriented Programming (AOP)
Focus	Primarily focuses on handling asynchronous data streams and events.	Primarily focuses on modularizing and managing cross-cutting concerns.
Paradigm	It is a programming paradigm that deals with data flow and the propagation of changes.	It is a programming paradigm that addresses the modularization of concerns that cross-cut multiple parts of a system.
Key Concepts	Key concepts include Observables, Observers, and operators for handling asynchronous data streams.	Key concepts include Aspects, Pointcuts, and Advices for handling cross-cutting concerns.
Use Cases	Suitable for scenarios where handling asynchronous events, real-time updates, and data streaming are critical, such as user interfaces and reactive systems.	Suitable for scenarios where cross-cutting concerns like logging, security, and transaction management need to be modularized and maintained separately.

Comparsions and Similarities

Libraries and Frameworks	Examples include RxJS (JavaScript), Reactor (Java), and RxSwift (Swift).	Examples include AspectJ (Java), PostSharp (.NET), and AspectC++ (C++).
Cross-Cutting Concerns Handling	Does not specifically address cross-cutting concerns; it focuses on handling asynchronous data streams.	Specializes in modularizing cross-cutting concerns, making it easier to manage aspects like logging, security, and error handling.
Code Modularity	Does not necessarily provide a mechanism for modularizing concerns that span multiple modules or layers.	Enables better code modularity by encapsulating cross-cutting concerns into aspects, promoting cleaner and more maintainable code.

Comparsions and Similarities

Execution Flow Handling	Primarily deals with the flow of data and events in an application.	Primarily deals with the execution flow of a program, intercepting and modifying it at defined join points.
Programming Language Support	Reactive programming can be applied in various programming languages with dedicated libraries or language features.	Often requires language support or specific extensions (e.g., AspectJ).
Runtime Modifications	Generally focuses on handling data at runtime without modifying the structure of the program.	Allows for the modification of a program's structure at compile-time or runtime, inserting aspects into specific points in the code.

Challenges Faced

- In the process of exploring the reactive programming with vye.js and aspect oriented programming with Aspect C++ i encountered few challenges, Starting with Vue.js, the transition from more traditional coding styles to the declarative and event-driven nature of Reactive Programming was bit challenging. Concepts like Observables, reactive data binding, and managing real-time updates seemed bit abstract initially. On the AspectC++ side, embracing AOP's abstract approach is like entering a new coding dimension. Understanding terms like pointcuts and advices, and figuring out how to intercept the flow of execution, was like bit challenging initially.

Conclusion

- Reactive programming revolutionizes how developers handle asynchronous data streams and events. In Vue.js, it brings a transformative development experience, empowering developers to create responsive and interactive user interfaces effortlessly. Vue.js simplifies the codebase, making it more readable and maintainable. It incorporates the reactivity system, including computed properties and watchers, optimizing performance and enabling custom logic in response to specific events.
- Aspect C++ stands out as a powerful tool for addressing cross-cutting concerns, seamlessly integrating AOP principles into C++ development. It enhances code modularity by encapsulating cross-cutting concerns into aspects, promoting cleaner and more maintainable code. The implementation involves defining pointcuts, which identify specific join points in the code, and advices, which contain additional functionality to be executed at those join points. This approach allows developers to modularize and maintain cross-cutting concerns separately from the core application logic, enhancing code organization and readability. Aspect C++ supports the use of named pointcuts and formal arguments, facilitating the creation of reusable and context-aware aspects, crucial for addressing diverse concerns in a flexible and targeted manner.

References

Wikipedia on Reactive Programming:
https://en.wikipedia.org/wiki/Reactive_programming

CodeGym's take on AOP: https://codegym.cc/groups/posts/543-what-is-aop-principles-of-aspect-oriented-programming

"A Study on AOP" (PDF):
https://crpit.scem.westernsydney.edu.au/confpapers/CRPITV10Spinczyk.pdf

More AOP wisdom from Wikipedia:
https://en.wikipedia.org/wiki/Aspect-oriented_programming

Spice it up with Spiceworks:
https://www.spiceworks.com/tech/devops/articles/what-is-aop/

TechTarget's two cents on Reactive Programming: https://www.techtarget.com/searchapparchitecture/definition/reactive-programming