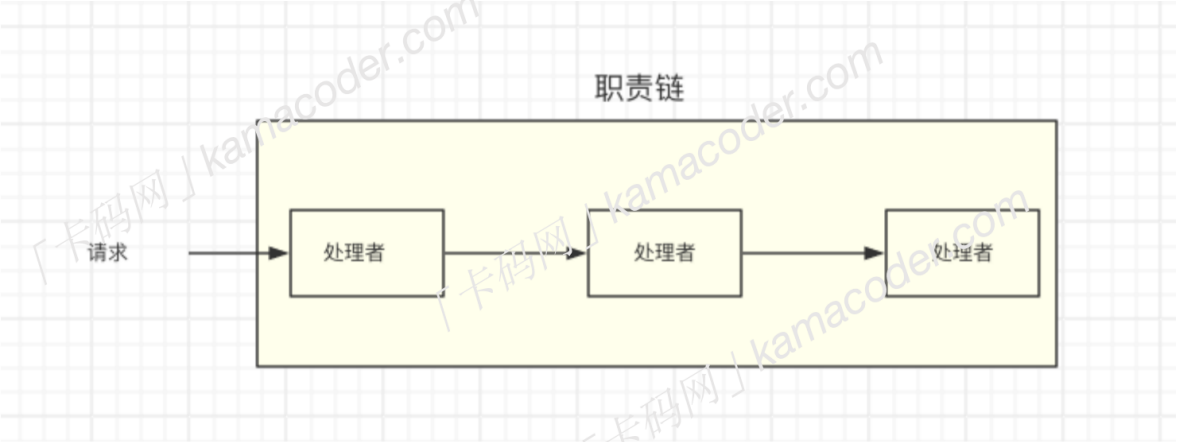
责任链模式

题目链接

责任链模式-请假审批

基本概念

责任链模式是一种行为型设计模式，它允许你构建一个对象链，让请求从链的一端进入，然后沿着链上的对象依次处理，直到链上的某个对象能够处理该请求为止。

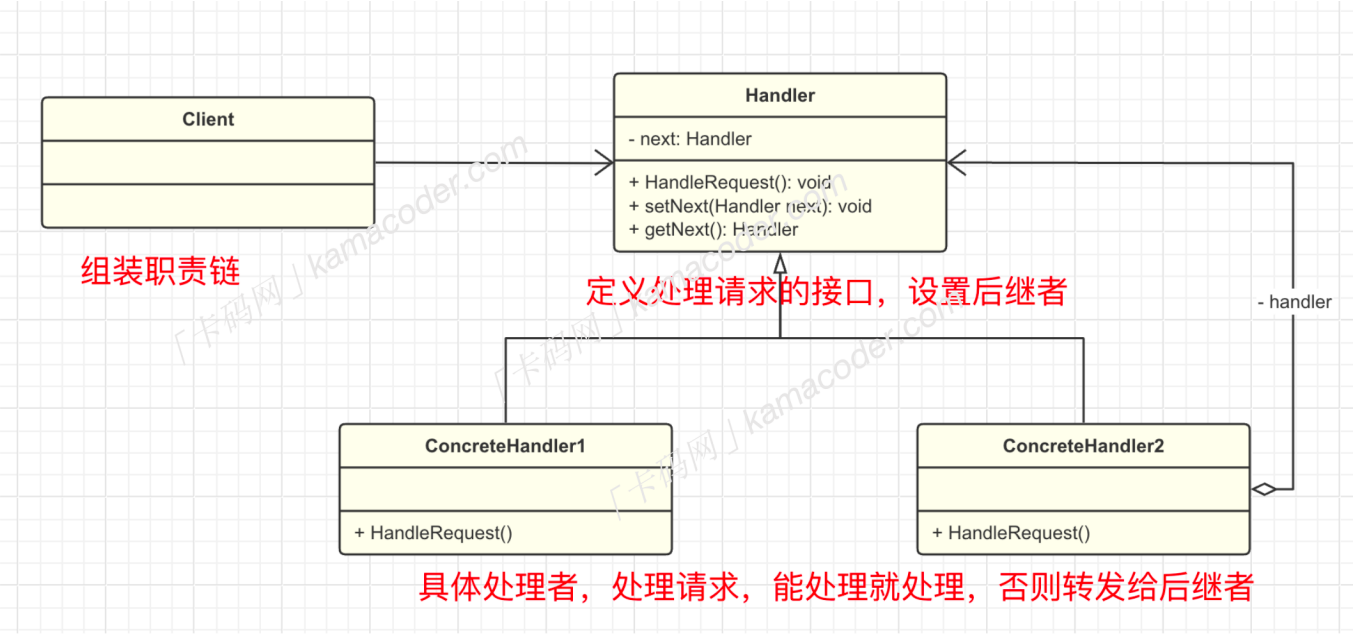


职责链上的处理者就是一个对象，可以对请求进行处理或者将请求转发给下一个节点，这个场景在生活中很常见，就是一个逐层向上递交的过程，最终的请求要么被处理者所处理，要么处理不了，这也因此可能导致请求无法被处理。

组成结构

责任链模式包括以下几个基本结构:

1. 处理者`Handler`： 定义一个处理请求的接口，包含一个处理请求的抽象方法和一个指向下一个处理者的链接。
2. 具体处理者`ConcreteHandler`: 实现处理请求的方法，并判断能否处理请求，如果能够处理请求则进行处理，否则将请求传递给下一个处理者。
3. 客户端：创建并组装处理者对象链，并将请求发送到链上的第一个处理者。



简易实现

1. 处理者：定义处理请求的接口

```
interface Handler {
    // 处理请求的方法
    void handleRequest(double amount);
    // 设置下一个处理者的方法
    void setNextHandler(Handler nextHandler);
}
```

2. 具体处理者：实现处理请求

```
class ConcreteHandler implements Handler {
    private Handler nextHandler;

    @Override
    public void handleRequest(Request request) {
        // 根据具体情况处理请求，如果无法处理则交给下一个处理者
        if (canHandle(request)) {
            // 处理请求的逻辑
        } else if (nextHandler != null) {
            // 交给下一个处理者处理
            nextHandler.handleRequest(request);
        } else {
            // 无法处理请求的逻辑
        }
    }
```

```
    @Override
    public void setNextHandler(Handler nextHandler) {
        this.nextHandler = nextHandler;
    }

    // 具体处理者自己的判断条件
    private boolean canHandle(Request request) {
        // 根据具体情况判断是否能够处理请求
        return /* 判断条件 */;
    }
}
```

3. 客户端创建并组装处理者对象链，将请求发送给链上第一个处理者

```
public class Main {
    public static void main(String[] args) {
        // 创建处理者实例
        Handler handler1 = new ConcreteHandler();
        Handler handler2 = new ConcreteHandler();
        // ...

        // 构建责任链
        handler1.setNextHandler(handler2);
        // ...

        // 发送请求
        Request request = new Request(/* 请求参数 */);
        handler1.handleRequest(request);
    }
}
```

使用场景

责任链模式具有下面几个优点：

- 降低耦合度：将请求的发送者和接收者解耦，每个具体处理者都只负责处理与自己相关的请求，客户端不需要知道具体是哪个处理者处理请求。
- 增强灵活性：可以动态地添加或删除处理者，改变处理者之间的顺序以满足不同需求。

但是由于一个请求可能会经过多个处理者，这可能会导致一些性能问题，并且如果整个链上也没有合适的处理者来处理请求，就会导致请求无法被处理。

责任链模式是设计模式中简单且常见的设计模式，在日常中也会经常使用到，比如Java开发中过滤器的链式处理，以及Spring框架中的拦截器，都组装成一个处理链对请求、响应进行处理。

本题代码

```
import java.util.Scanner;

// 处理者：定义接口
interface LeaveHandler {
    void handleRequest(LeaveRequest request);
}

// 具体处理者：可以有多个，负责具体处理，这里分为 Supervisor、Manager、Director
class Supervisor implements LeaveHandler {
    private static final int MAX_DAYS_SUPERVISOR_CAN_APPROVE = 3;
    private LeaveHandler nextHandler;

    public Supervisor(LeaveHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    @Override
    public void handleRequest(LeaveRequest request) {
        if (request.getDays() <= MAX_DAYS_SUPERVISOR_CAN_APPROVE) {
            System.out.println(request.getName() + " Approved by
Supervisor.");
        } else if (nextHandler != null) {
            nextHandler.handleRequest(request);
        } else {
            System.out.println(request.getName() + " Denied by
Supervisor.");
        }
    }
}

class Manager implements LeaveHandler {
    private static final int MAX_DAYS_MANAGER_CAN_APPROVE = 7;
    private LeaveHandler nextHandler;

    public Manager(LeaveHandler nextHandler) {
        this.nextHandler = nextHandler;
    }

    @Override
    public void handleRequest(LeaveRequest request) {
        if (request.getDays() <= MAX_DAYS_MANAGER_CAN_APPROVE) {
            System.out.println(request.getName() + " Approved by
Manager.");
        } else if (nextHandler != null) {
```

```
            nextHandler.handleRequest(request);
        } else {
            System.out.println(request.getName() + " Denied by Manager.");
        }
    }
}

class Director implements LeaveHandler {
    private static final int MAX_DAYS_DIRECTOR_CAN_APPROVE = 10;

    @Override
    public void handleRequest(LeaveRequest request) {
        if (request.getDays() <= MAX_DAYS_DIRECTOR_CAN_APPROVE) {
            System.out.println(request.getName() + " Approved by
Director.");
        } else {
            System.out.println(request.getName() + " Denied by Director.");
        }
    }
}

// 请求类
class LeaveRequest {
    private String name;
    private int days;

    public LeaveRequest(String name, int days) {
        this.name = name;
        this.days = days;
    }

    public String getName() {
        return name;
    }

    public int getDays() {
        return days;
    }
}

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        int n = scanner.nextInt();
        scanner.nextLine();
```

```
        // 组装职责链
        LeaveHandler director = new Director();
        LeaveHandler manager = new Manager(director);
        LeaveHandler supervisor = new Supervisor(manager);

        for (int i = 0; i < n; i++) {
            String[] input = scanner.nextLine().split(" ");
            if (input.length == 2) {
                String name = input[0];
                int days = Integer.parseInt(input[1]);
                LeaveRequest request = new LeaveRequest(name, days);
                supervisor.handleRequest(request);
            } else {
                System.out.println("Invalid input");
                return;
            }
        }
    }
}
```

其他语言版本

Java

使用枚举封装了请求级别的逻辑，方便未来的修改和扩展。

```
import java.util.Scanner;

// 抽象处理器类，定义了责任链的基本结构
abstract class Handler {
    public final static int SUPERVISOR_LEVEL_REQUEST = 1;
    public final static int MANAGER_LEVEL_REQUEST = 2;
    public final static int DIRECTOR_LEVEL_REQUEST = 3;

    private Handler nextHandler;
    private int level = 0;

    // 构造函数，设置处理器的级别
    public Handler(int _level) {
        this.level = _level;
    }

    // 处理请求的方法
    public final Response handleMessage(Request request) {
        if (this.level == request.getRequestLevel()) {
```

```
            return this.response(request);
        } else {
            if (this.nextHandler != null) {
                return this.nextHandler.handleMessage(request);
            } else {
                return new Response("Request denied");
            }
        }
    }

    // 设置下一个处理器
    public void setNext(Handler _handler) {
        this.nextHandler = _handler;
    }

    protected abstract Response response(Request request);
}

// 主管处理
class SupervisorHandler extends Handler {
    public SupervisorHandler() {
        super(Handler.SUPERVISOR_LEVEL_REQUEST);
    }

    @Override
    protected Response response(Request request) {
        System.out.println(request.getName() + " Approved by Supervisor.");
        return new Response("Approved by Supervisor");
    }
}

// 经理处理
class ManagerHandler extends Handler {
    public ManagerHandler() {
        super(Handler.MANAGER_LEVEL_REQUEST);
    }

    @Override
    protected Response response(Request request) {
        System.out.println(request.getName() + " Approved by Manager.");
        return new Response("Approved by Manager");
    }
}

// 董事处理
class DirectorHandler extends Handler {
```

```
    public DirectorHandler() {
        super(Handler.DIRECTOR_LEVEL_REQUEST);
    }

    @Override
    protected Response response(Request request) {
        System.out.println(request.getName() + " Approved by Director.");
        return new Response("Approved by Director");
    }
}

// 请求级别的枚举，定义了不同级别的请假天数范围
enum RequestLevel {
    SUPERVISOR(1, 3),
    MANAGER(4, 5),
    DIRECTOR(6, 10);

    private final int minDays;
    private final int maxDays;

    RequestLevel(int minDays, int maxDays) {
        this.minDays = minDays;
        this.maxDays = maxDays;
    }

    // 根据天数确定请求级别
    public static RequestLevel fromDays(int days) {
        for (RequestLevel level : values()) {
            if (days >= level.minDays && days <= level.maxDays) {
                return level;
            }
        }
        return null;
    }

    public int getValue() {
        return ordinal() + 1;
    }
}

// 请求类，包含请求的详细信息
class Request {
    private String name;
    private int level;
    private int nums;
```

```
    public Request(String name, int nums) {
        this.name = name;
        this.nums = nums;
        RequestLevel requestLevel = RequestLevel.fromDays(nums);
        this.level = (requestLevel != null) ? requestLevel.getValue() : -1;
    }

    public int getRequestLevel() {
        return this.level;
    }

    public String getName() {
        return this.name;
    }

    public int getNums() {
        return this.nums;
    }
}

// 响应类，包含处理结果
class Response {
    private String message;

    public Response(String message) {
        this.message = message;
    }

    public String getMessage() {
        return this.message;
    }
}

// 主类
public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // 创建处理器链
        SupervisorHandler supervisorHandler = new SupervisorHandler();
        ManagerHandler managerHandler = new ManagerHandler();
        DirectorHandler directorHandler = new DirectorHandler();

        // 设置处理器链的顺序
        supervisorHandler.setNext(managerHandler);
        managerHandler.setNext(directorHandler);
```

```
        int n = scanner.nextInt();
        scanner.nextLine();

        // 处理每个请求
        while (n-- > 0) {
            String[] s = scanner.nextLine().split(" ");
            Request request = new Request(s[0], Integer.parseInt(s[1]));
            Response response = supervisorHandler.handleMessage(request);
            if (request.getRequestLevel() == -1) {
                System.out.println(request.getName() + " Denied by
Director.");
            }
        }

        scanner.close();
    }
}
```

C++

```
#include <iostream>
#include <sstream>

class LeaveHandler {
public:
    virtual void handleRequest(const std::string& name, int days) = 0;
};

class Supervisor : public LeaveHandler {
private:
    static const int MAX_DAYS_SUPERVISOR_CAN_APPROVE = 3;
    LeaveHandler* nextHandler;

public:
    Supervisor(LeaveHandler* nextHandler) : nextHandler(nextHandler) {}

    void handleRequest(const std::string& name, int days) override {
        if (days <= MAX_DAYS_SUPERVISOR_CAN_APPROVE) {
            std::cout << name << " Approved by Supervisor." << std::endl;
        } else if (nextHandler != nullptr) {
            nextHandler->handleRequest(name, days);
        } else {
            std::cout << name << " Denied by Supervisor." << std::endl;
        }
    }
```

```
};

class Manager : public LeaveHandler {
private:
    static const int MAX_DAYS_MANAGER_CAN_APPROVE = 7;
    LeaveHandler* nextHandler;

public:
    Manager(LeaveHandler* nextHandler) : nextHandler(nextHandler) {}

    void handleRequest(const std::string& name, int days) override {
        if (days <= MAX_DAYS_MANAGER_CAN_APPROVE) {
            std::cout << name << " Approved by Manager." << std::endl;
        } else if (nextHandler != nullptr) {
            nextHandler->handleRequest(name, days);
        } else {
            std::cout << name << " Denied by Manager." << std::endl;
        }
    }
};

class Director : public LeaveHandler {
private:
    static const int MAX_DAYS_DIRECTOR_CAN_APPROVE = 10;

public:
    void handleRequest(const std::string& name, int days) override {
        if (days <= MAX_DAYS_DIRECTOR_CAN_APPROVE) {
            std::cout << name << " Approved by Director." << std::endl;
        } else {
            std::cout << name << " Denied by Director." << std::endl;
        }
    }
};

class LeaveRequest {
private:
    std::string name;
    int days;

public:
    LeaveRequest(const std::string& name, int days) : name(name),
days(days) {}

    std::string getName() const {
        return name;
```

```
    }

    int getDays() const {
        return days;
    }
};

int main() {
    int n;
    std::cin >> n;
    std::cin.ignore();

    LeaveHandler* director = new Director();
    LeaveHandler* manager = new Manager(director);
    LeaveHandler* supervisor = new Supervisor(manager);

    for (int i = 0; i < n; i++) {
        std::string input;
        std::getline(std::cin, input);
        std::istringstream iss(input);

        std::string name;
        int days;

        if (iss >> name >> days) {
            LeaveRequest request(name, days);
            supervisor->handleRequest(name, days);
        } else {
            std::cout << "Invalid input" << std::endl;
            return 1;
        }
    }

    delete supervisor;
    delete manager;
    delete director;

    return 0;
}
```

Python

```
class LeaveHandler:
    def handle_request(self, name, days):
        pass
```

```
class Supervisor(LeaveHandler):
    MAX_DAYS_SUPERVISOR_CAN_APPROVE = 3

    def __init__(self, next_handler=None):
        self.next_handler = next_handler

    def handle_request(self, name, days):
        if days <= self.MAX_DAYS_SUPERVISOR_CAN_APPROVE:
            print(f"{name} Approved by Supervisor.")
        elif self.next_handler:
            self.next_handler.handle_request(name, days)
        else:
            print(f"{name} Denied by Supervisor.")


class Manager(LeaveHandler):
    MAX_DAYS_MANAGER_CAN_APPROVE = 7

    def __init__(self, next_handler=None):
        self.next_handler = next_handler

    def handle_request(self, name, days):
        if days <= self.MAX_DAYS_MANAGER_CAN_APPROVE:
            print(f"{name} Approved by Manager.")
        elif self.next_handler:
            self.next_handler.handle_request(name, days)
        else:
            print(f"{name} Denied by Manager.")


class Director(LeaveHandler):
    MAX_DAYS_DIRECTOR_CAN_APPROVE = 10

    def handle_request(self, name, days):
        if days <= self.MAX_DAYS_DIRECTOR_CAN_APPROVE:
            print(f"{name} Approved by Director.")
        else:
            print(f"{name} Denied by Director.")


class LeaveRequest:
    def __init__(self, name, days):
        self.name = name
        self.days = days

    def get_name(self):
```

```
        return self.name

    def get_days(self):
        return self.days

if __name__ == "__main__":
    n = int(input())

    director = Director()
    manager = Manager(director)
    supervisor = Supervisor(manager)

    for _ in range(n):
        input_data = input().split()
        if len(input_data) == 2:
            name, days = input_data
            days = int(days)
            request = LeaveRequest(name, days)
            supervisor.handle_request(name, days)
        else:
            print("Invalid input")
            exit(1)
```

Go

```
package main

import (
    "fmt"
    "bufio"
    "os"
    "strconv"
    "strings"
)

// 处理者：定义接口
type LeaveHandler interface {
    HandleRequest(request LeaveRequest)
}

// 具体处理者：可以有多个，负责具体处理，这里分为 Supervisor、Manager、Director
type Supervisor struct {
    nextHandler LeaveHandler
}

const maxDaysSupervisorCanApprove = 3
```

```go
func NewSupervisor(nextHandler LeaveHandler) *Supervisor {
    return &Supervisor{nextHandler: nextHandler}
}

func (s *Supervisor) HandleRequest(request LeaveRequest) {
    if request.Days <= maxDaysSupervisorCanApprove {
        fmt.Println(request.Name + " Approved by Supervisor.")
    } else if s.nextHandler != nil {
        s.nextHandler.HandleRequest(request)
    } else {
        fmt.Println(request.Name + " Denied by Supervisor.")
    }
}

type Manager struct {
    nextHandler LeaveHandler
}

const maxDaysManagerCanApprove = 7

func NewManager(nextHandler LeaveHandler) *Manager {
    return &Manager{nextHandler: nextHandler}
}

func (m *Manager) HandleRequest(request LeaveRequest) {
    if request.Days <= maxDaysManagerCanApprove {
        fmt.Println(request.Name + " Approved by Manager.")
    } else if m.nextHandler != nil {
        m.nextHandler.HandleRequest(request)
    } else {
        fmt.Println(request.Name + " Denied by Manager.")
    }
}

type Director struct{}

const maxDaysDirectorCanApprove = 10

func (d *Director) HandleRequest(request LeaveRequest) {
    if request.Days <= maxDaysDirectorCanApprove {
        fmt.Println(request.Name + " Approved by Director.")
    } else {
        fmt.Println(request.Name + " Denied by Director.")
    }
}
```

```go
// 请求类
type LeaveRequest struct {
    Name string
    Days int
}

// 主函数
func main() {
    scanner := bufio.NewScanner(os.Stdin)

    // 读取用户输入
    scanner.Scan()
    var n int
    fmt.Sscanf(scanner.Text(), "%d", &n)

    // 组装职责链
    director := &Director{}
    manager := NewManager(director)
    supervisor := NewSupervisor(manager)

    for i := 0; i < n; i++ {
        scanner.Scan()
        input := strings.Fields(scanner.Text())

        if len(input) == 2 {
            name := input[0]
            days, err := strconv.Atoi(input[1])
            if err != nil {
                fmt.Println("Invalid input")
                return
            }

            request := LeaveRequest{Name: name, Days: days}
            supervisor.HandleRequest(request)
        } else {
            fmt.Println("Invalid input")
            return
        }
    }
}
```