哈尔滨工业大学

计算学部

2024 年秋季学期

《软件架构与中间件》课程

实验报告

Lab 1：简易消息中间件开发

姓名	学号	联系方式
余昊卿	2023120253	Felixupri@gmail.com
马嘉良	2023120259	1009728013@qq.com

学号：	**2023120253**	**姓名：余昊卿**	
学号：	**2023120259**	**姓名：马嘉良**	

1 实验概述

1.1 实验目的

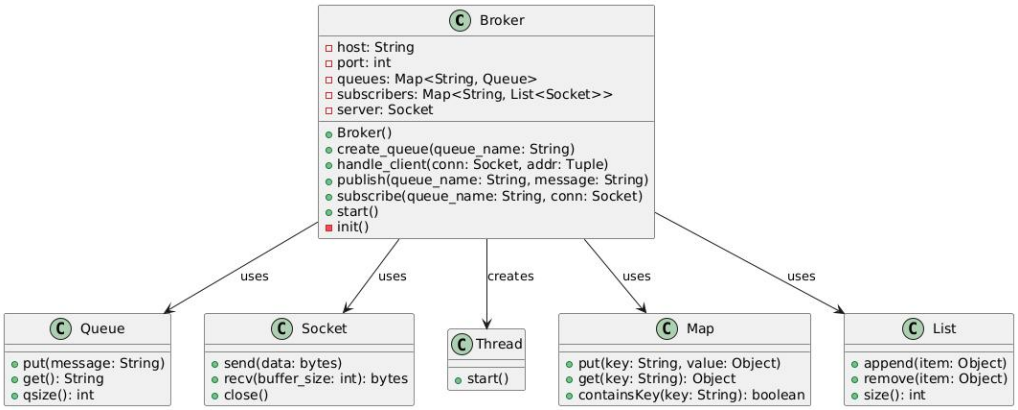
1）学习事件系统型软件架构风格
2）学会使用观测者模式或者订阅发布模式对事件系统进行建模分析
3）能够独立开发简单的消息中间件
4）能够灵活应用所开发的中间件到实际系统

1.2 实验要求

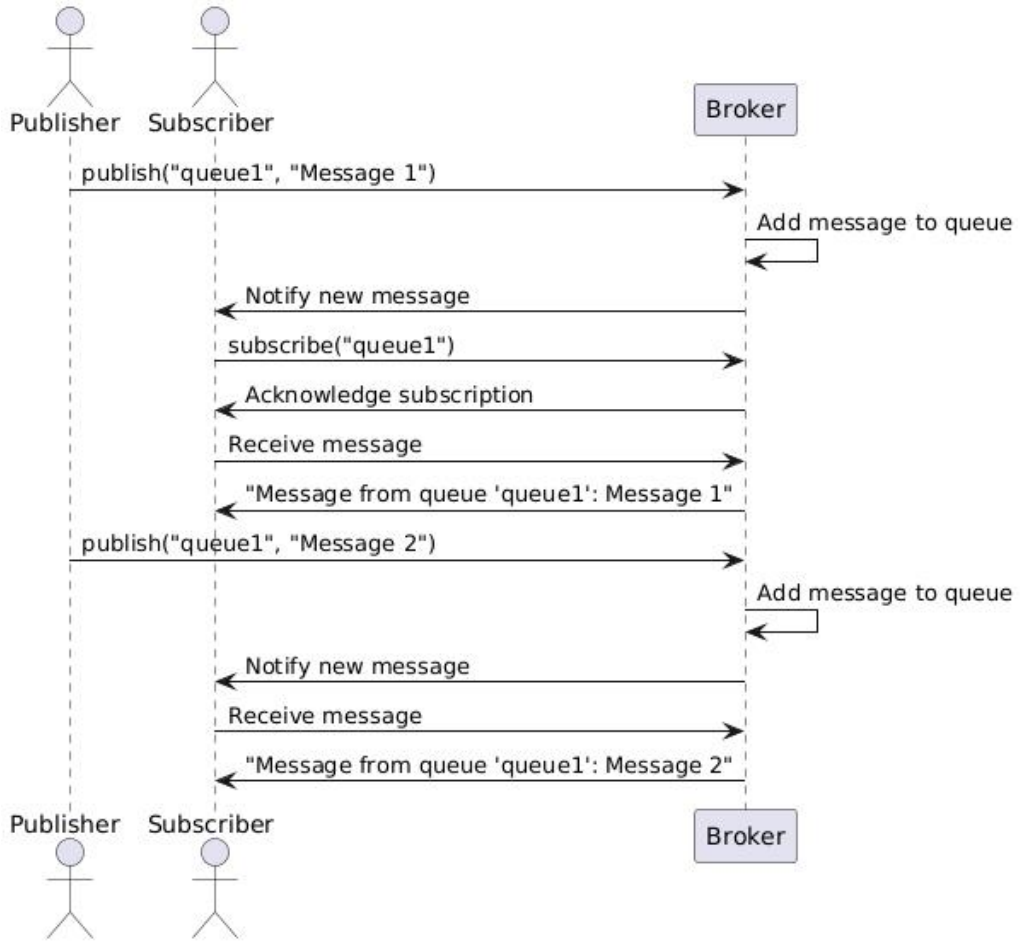
1）2 人结对成组
2）不可借助开源消息中间件框架
3）模拟实现一种简单的消息中间件，能达到对系统功能解耦的目的
4）分析消息中间件的吞吐率等非功能指标
5）应结合《软件过程与工具》课程中进销存系统(或其他实际软件系统)进行分析，给出哪些场景可以运用该消息中间件
6）应该出设计过程和实现细节

2 设计方案

中间件采用订阅发布模式，类图如下：



基于 socket 编程实现了服务器和客户端之间的通信。采用单例模式，Broker 和同名的消息队列只会创建一个实例。订阅者可以订阅若干消息队列，发布者发布新消息时，中间件会通知订阅者，时序图如下：



3 程序实现

3.1 开发环境与工具

开发语言：Python 3.10
开发环境：Windows 11

3.2 实现要点说明

Broker 采用单例模式实现，程序运行过程中仅创建一个 Broker。

```
    def __new__(cls, *args, **kwargs):
        if not cls._instance:
            cls._instance = super(Broker, cls).__new__(cls)
            cls._instance.init()
        return cls._instance
```

Broker 的主要接口实现如下：

```
def publish(self, queue_name, message):
    if queue_name in self.queues:
        self.queues[queue_name].put(message)
        print(f"Message published to queue '{queue_name}': {message}")

        for subscriber in self.subscribers[queue_name]:
            try:
                subscriber.send(f"Message from queue '{queue_name}': {message}".encode())
            except:
                self.subscribers[queue_name].remove(subscriber)
    else:
        print(f"Queue '{queue_name}' does not exist.")

def subscribe(self, queue_name, conn):
    if queue_name in self.queues:
        self.subscribers[queue_name].append(conn)
        print(f"New subscriber to queue '{queue_name}'")
    else:
        print(f"Queue '{queue_name}' does not exist.")
```

Subscriber 启动一个线程用于接收消息：

```
def subscribe(self, queue_name):

    def listen_for_messages(sock):
        while True:
            data = sock.recv(1024).decode()
            if data:
                print(f"{self.name} received: {data}")

    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect((self.host, self.port))
        command = f"SUBSCRIBE|{queue_name}"
        s.send(command.encode())

        # 启动线程用于接收消息
        threading.Thread(
            target=listen_for_messages,
            args=(s,),
            daemon=True
        ).start()

        # 保持连接
        while True:
            pass
```

4 中间件测试与应用

4.1 中间件测试与验证

运行 Broker，并创建队列 Q1 和 Q2。

```
$ python broker.py --queue Q1 --queue Q2
Queue 'Q1' created.
Queue 'Q2' created.
Broker started at localhost:5555
```

创建 Subscriber A，订阅 Q1 的消息。

```
$ python subscriber.py --name A --queue Q1
```

创建 Subscriber B，订阅 Q2 的消息。

```
$ python subscriber.py --name B --queue Q2
```

订阅成功后运行 Broker 的终端会显示相应的信息：

```
$ python broker.py --queue Q1 --queue Q2
Queue 'Q1' created.
Queue 'Q2' created.
Broker started at localhost:5555
Connection from ('127.0.0.1', 3985) established.
New subscriber to queue 'Q1'
Connection from ('127.0.0.1', 4013) established.
New subscriber to queue 'Q2'
```

向 Q1 发布消息：

```
$ python publisher.py --queue Q1 --message "hello Q1"
```

运行 Broker 的终端和运行 Subscriber 的终端会显示相应的消息，Subscriber A 接收到了来自 Q1 的消息。

```
$ python broker.py --queue Q1 --queue Q2
Queue 'Q1' created.
Queue 'Q2' created.
Broker started at localhost:5555
Connection from ('127.0.0.1', 3985) established.
New subscriber to queue 'Q1'
Connection from ('127.0.0.1', 4013) established.
New subscriber to queue 'Q2'
Connection from ('127.0.0.1', 4437) established.
Message published to queue 'Q1': hello Q1
```

```
$ python subscriber.py --name A --queue Q1
Subscriber A received: Message from queue 'Q1': hello Q1
```

向 Q2 发布消息：

```
$ python publisher.py --queue Q2 --message "hello Q2"
```

Subscriber B 接收到了来自 Q2 的消息。

```
$ python broker.py --queue Q1 --queue Q2
Queue 'Q1' created.
Queue 'Q2' created.
Broker started at localhost:5555
Connection from ('127.0.0.1', 3985) established.
New subscriber to queue 'Q1'
Connection from ('127.0.0.1', 4013) established.
New subscriber to queue 'Q2'
Connection from ('127.0.0.1', 4437) established.
Message published to queue 'Q1': hello Q1
Connection from ('127.0.0.1', 4681) established.
Message published to queue 'Q2': hello Q2
```

```
$ python subscriber.py --name B --queue Q2
Subscriber B received: Message from queue 'Q2': hello Q2
```

测试消息中间件的吞吐量

可以通过测试发布 1000 条消息的耗时来衡量消息中间件的吞吐量，测试结果如下：

```
$ py test.py
Published 1000 messages in 1.0257 seconds.
Throughput: 974.95 messages/second
```

4.2 实际系统中的运用

电商网站的订单处理

当用户在电商平台上下单时，系统需要执行一系列操作，例如生成订单、处理支付、更新库存、发送订单确认邮件等。这些操作如果同步执行，会导致用户下单时延迟过长。通过中间件订单生成操作可以立即返回，而支付处理、库存更新等操作可以异步地推送到消息队列中，后台任务逐步处理这些任务。使用中间件降低了用户操作的延迟，提升用户体验，同时可以通过水平扩展消费者来处理更多订单。

图片或视频的批量处理

对于在线图片分享网站，用户上传图片后，需要对图片进行压缩、打水印等处理。这些处理任务较为耗时，且不适合在上传请求时同步完成。通过使用中间件，图片上传服务可以立即返回，然后异步处理这些任务。使用中间件可以让任务处理更具弹性，可以通过增加或减少消费者来调节并发量和资源使用，保证任务处理的平稳性。

新闻推送系统

在新闻门户网站中，当有新的新闻发布时，用户可能订阅了不同类别的新闻推送。通过中间件的发布/订阅模型，可以方便地将新闻发布到不同的消费者。发布/订阅模型使得生产者和消费者解耦，生产者不需要知道哪些用户订阅了特定类型的新闻，中间件负责消息的路由和分发。

5 结对开发过程记录

(1) 角色切换与任务分工

表 1-1 结对开发角色与任务分工

日期	时间	驾驶员	领航员	本段时间的任务
9/25	16:20-17:10	余昊卿	马嘉良	讨论中间件的应用场景与实现功能，初步完成整体架构设计。

9/26	15:26-18:37	马嘉良	余昊卿	实现中间件的核心功能
9/27	13:00-16:30	马嘉良	余昊卿	进一步完善中间件
10/3	18:20-23:56	余昊卿	马嘉良	测试中间件，撰写实验报告。

(2) 工作日志

由领航员负责记录，记录结对开发期间的遇到的问题、两人如何通过交流合作解决每个问题的。

表 1-2 结对开发工作日志

日期/时间	问题描述	最终解决方法	交流过程
9/25	中间件的接口设计	参考 RabbitMQ 进行设计	分别阅读了常见中间件如 Kafka 等的文档，后经过交流确定方案。
9/27	如何实现服务器和客户端之间的通信	基于 Socket 编程实现	考虑到基于 Socket 编程有较高的灵活性，于是确定该方案。

(3) 结对开发工作现场照片

请其他同学帮助拍摄结对开发现场照片至少 2 张或讨论记录截屏。

结对开发现场照片 1

结对开发现场照片 2

6 实验总结

通过本次实验，进一步了解了事件系统型软件架构风格及其应用场景，学习了观测者模式和订阅发布模式的原理，并对事件系统进行建模与分析。通过事件驱动的方式可以实现系统的高效解耦，提高系统的灵活性。通过开发一个简单的消息中间件，实现中间件的基本功能，增强了对事件驱动架构的理解。目前简易中间件的性能有待提高，系统的健壮性仍有不足。我们会在未来的学习与实践过程中继续学习相关知识，逐步完善该系统。

7 教师评语	