《软件体系结构与设计》实习报告

题目： 第 1 次上机实验

班级序号：111171

学生姓名：董安宁

任课教师：尚建嘎

中国地质大学信息工程学院软件工程系

2019 年 9 月

一、实验概况

实验时间：2019 年 9 月 18（周三）晚上 18:30-21:30，共 4 课时

实验地点：未来城校区公教 2-503

实验目的：掌握开发、测试、发布、调用进程间通信的基本方法、工具和流程，理解独立构件体系结构基本原理、结构和特点。掌握使用当今主流云平台来构建独立构件风格软件的相关开发技能。

背景及要求：

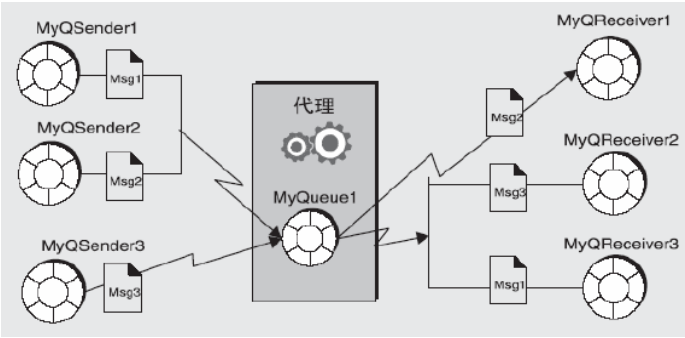
现今，随着软件开发规模的逐渐增大，软件开发的规范准则也随之发生了变化，从最开始只注重程序正确性运行的算法，到现如今注重在整个开发中的架构模式，软件复用性等等质量属性，软件体系结构的规范化与结构化对软件开发的影响越来越大。其中，数据流风格就是软件体系结构风格中一种典型的风格，其高耦合低内聚的特点和构件的独立性使得软件的复用性很高，适用于需要处理源源不断的数据的系统。

[以上说明来自教材以及自己的理解]

现今，越来越多的企业面临着各种各样的数据集成和系统整合的系统需求，掌握开发、测试、发布、调用进程间通信的基本方法、工具和流程显得很重要。在这样的系统需求之下，RPC 中间件技术也应运而生，但由于采用 RPC 同步处理技术，在性能、健壮性、可扩展性上都存在诸多缺点。而基于消息的异步处理模型则采用非阻塞的调用特性，发送者将消息发送给消息服务器，消息服务器在合适的时候再将消息转发给接收者；发送和接收是异步的，发送者无需等待。使用消息中间件作为一个中间层的软件，掌握使用云计算技术来构建独立构件风格的相关技能。

以下题目任意选做一个：

（1）基于 AWS SQS（亚马逊云）或阿里云等简单队列服务的消息中间件，使用 Java，C#或者其他语言分别编写一个发送程序和接收程序（构建两个进程或者程序，一个用于发送消息--发到云端队列，一个用于接收消息--从云端队列订阅下来），实现“点对点”的进程间通信功能。



提示与思考：

1) AWS 相关基本操作，在另外一个文档中，里面有基本的如何获取 AWS key，以及如何建立 AWS 连接。

2) 前端页面简洁明了，用户体验较好，重点在后台通信机制。

3) 这种消息队列服务是基础性的，AWS 作为商业云平台提供了针对 SQS 的高可用性解决方案。如果你基于 Kafka 构建消息队列服务，如何确保其高可用性？

4) 相关链接：

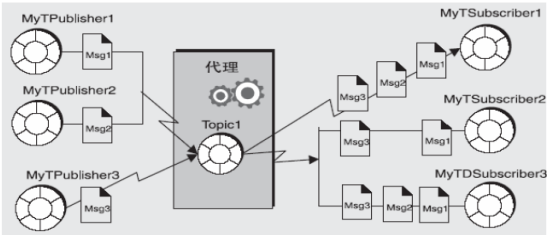
AWS .NET API,你可以在该链接找到你想要的类及相关方法

https://docs.aws.amazon.com/sdkfornet/v3/apidocs/Index.html

Java API:https://docs.aws.amazon.com/zh_cn/AWSJavaSDK/latest/javadoc/index.html

SQS 官方文档链接：https://docs.aws.amazon.com/sqs/index.html#lang/zh_cn

（2）基于 AWS SNS（亚马逊云），或阿里云消息推送服务，使用 Java、C#或者其他语言编写一个发送程序和一个接收程序，实现发布-订阅的选择广播式功能，要求订阅者程序为邮件和 SQS 队列。



发布-订阅模式

（3）基于一款开源 JMS 消息中间件（如 Active MQ、Rabbit MQ、kafaka），使用 Java 编写一个发送程序和接收程序，实现点对点和发布-订阅的选择广播式功能，并进行测试。

JMS 选型参考资料：http://blog.csdn.net/oMaverick1/article/details/51331004

要求：

（1）程序应具有 GUI，发送程序和接收程序可选择发送和接收方式；

（2）通过对话框可以输入发送消息，接收结果可显示于对话框中。

（3）报告结果中要有对于“点对点”和“发布-订阅”两种模式的比较分析。

二、实验设计（给出你的实习内容的设计方案，可根据实际情况调整条目）

2.1 系统需求

环境需求：

用户需要在 C:\Users\用户名\.aws 目录下设置 credentials 和 config 文件，用以保存自己的 AWS 账号信息。

功能需求：

实现发布-订阅模型，完成一对多的异步消息发送，使用数据流体系风格提高复用性。

质量需求：

在遇到错误指令或者系统内部发生错误后可以显示出来，不会因此导致程序崩溃。

2.2 架构设计

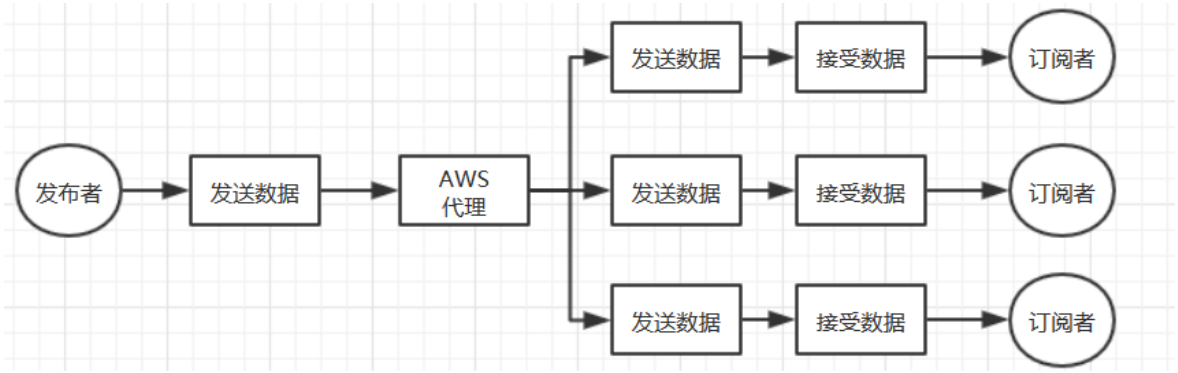


图 1 数据流架构设计

2.3 接口设计

发布者和发送数据功能之间的接口，仅用来传输数据，发布者无需了解数据以怎样的方式发送出去，只需要发送数据。

发送数据和 AWS 代理之间的接口，仅用来传输数据，发送数据的构件无需了解 AWS 代理如何处理数据，只需要发送数据。

AWS 代理和发送构件之间的接口，仅用来传输数据，发送数据的构件无需了解数据如何发送，只需要将数据传输过去，剩下的无需管理。

接受数据和订阅者之间的接口，数据存储在这里，等用户在线的时候推送到用户眼前。

三、实验过程

3.1 软件实现

首先我将该数据流架构分成了三部分，分别是发布者客户端，AWS 代理，以及订阅者客户端。数据从发布者发送，流向 AWS 代理，最终流向订阅者客户段。每个客户端再分别有向 AWS 代理发送数据和向 AWS 接受数据的功能，由两个客户端分别调用。

客户端：

① 客户端涉及自动登录的步骤，如果没有更新 AWS 的 key 的话会导致登录失败，抛出对话框显示错误：

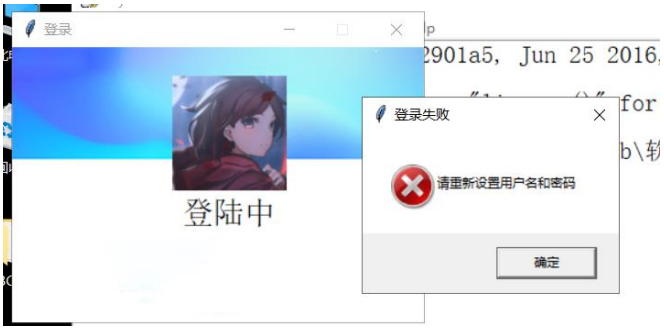


图 2 登陆界面

② 登录成功后会显示用户界面，用户界面有文本输入框，发送按钮，邮箱接受按钮，以及 SQS 接受按钮，布局方式如右图所示：

图 3 用户界面

其中，每个功能都有自己对应的构件来执行不同的操作。构件之间并没有直接的联系，之间调用的关系都是通过字符串的传输等方式来进行调用。当点击发送消息后会开启发送消息线程，消息会发送到 AWS 代理服务器，代理服务器再发送给所有订阅者，其中邮箱订阅能够通过 QQ 的提示直接查询到消息，形成同步发送接受的通信。

AWS 代理

AWS 代理是在 AWS 控制台线上完成的，只需要确定主题，连接订阅者，就可以发送数据了。只要我通过代码访问该 AWS 上的 SNS，传入字符串后，就可以完成分发操作了。

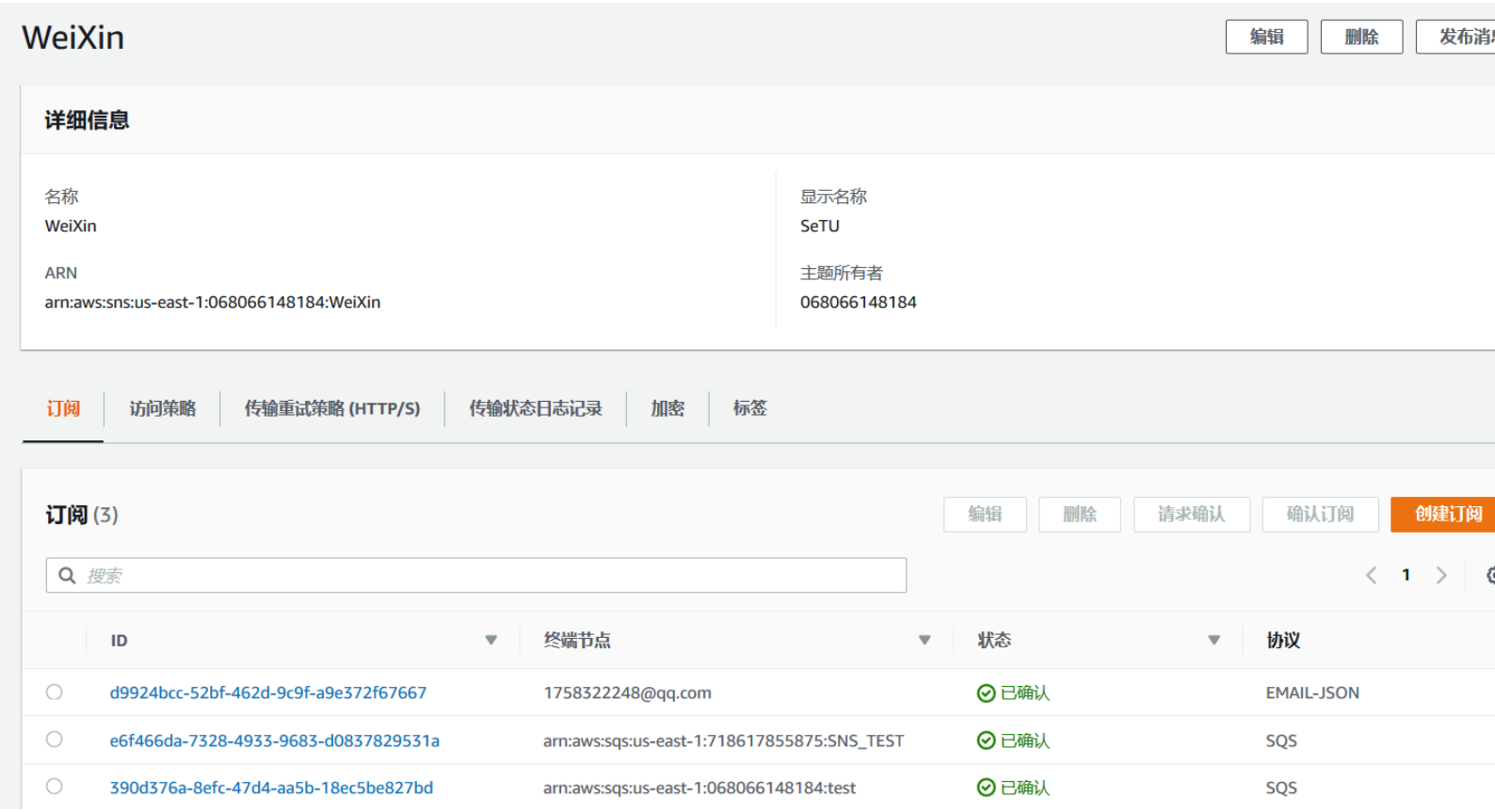


图 4 SNS 界面

SQS 接收端

订阅者（SQS）客户端专门负责接受数据，只需要传入队列的名字就可以查询队列收到的消息，也可以选择删除消息，删除在 SQS 中所查询到的消息。界面如下图所示：

图 5 SQS 队列接受界面

3.2 实验环境

处理器： i7-7700HQ　　操作系统：windows10
开发语言：python　　服务器：AWS
实验场景：宿舍

3.3 实验步骤

① 首先我分别写出了两个界面，分别对两个界面进行调试，如下图所示：

```
if __name__ == '__main__':
    sqsbox()
    User()
    LoginBar()
```

图 6 三个界面所对应的函数

② 每个界面我都分别对界面的布局进行了大量的设置（其实个人感觉这些有些多余，不需要特别炫酷的界面，能用就完事了...）

③ 分别测试了界面的按钮功能以及链接亚马逊后的接受和发送功能，都以打印的方式输出在面板上，供以观察是否出现 bug，如下图所示：

```
==== RESTART: C:\Users\UncleDong\Desktop\lab\软件体系结构\第1次实验指导书\亚马逊课设\亚马逊.py ====
连上了连上了
>>> hi
UncleDong
https://queue.amazonaws.com/068066148184/UncleDong
[]
test
https://queue.amazonaws.com/068066148184/test
[sqs.Message(queue_url='https://queue.amazonaws.com/068066148184/test', receipt_handle='AQEBenwQ8Ig3Db09iJeXxavfrgJg6Ovq98oGLQAtjn4mOHSLb0gUWa4G5ZPvZZ/xkw83QBJO5Gzof7PH4aik7RF4THRbQq+P9Nt1Umk/k2iHNR4MXDj81tac8v2+LqzwQdCt5eo6cyxOp/IJCWOYjdfHHnEGe8XXQF7pevN4FrQZ3bUz/a5M3Ua2WI9teZ4V5Fzmdasl/fhFASv992dfZ3DRdjJ7mGHV9PlxjfXQEy1sK/OzuqY2sAS6YCegniDJG8TnUMRDTI4mBy28UbSFc2u/lSG4QLmU8Dc8WFCYGl3J+ohvdllNCP8mj/O9dy91wJqZngofym7S3AGEGgWF7I9Oo+H2qooS+n9xLDXp1CE9UqPf7IA4oM/RxFJYrNJNTsGV')]
[]
```

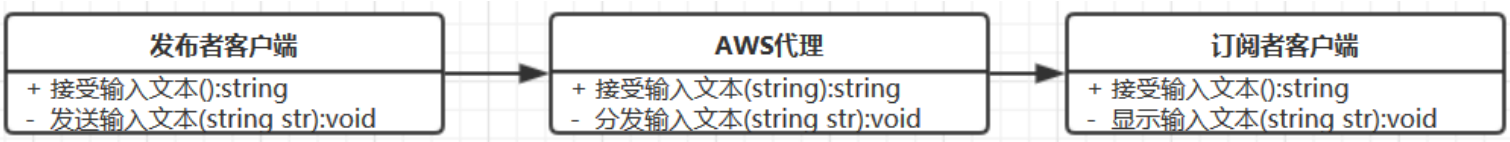
图 7 打印出信息

④ 最后分别调试完毕后，将模块通过数据的发送链接起来，再进行微小的错误调试，就完成了。

四、实验评价

4.1 实验结果

本次实验完成了题目 1 和题目 2 的内容，实现了基于管道数据流风格的点对点消息发送和发布订阅者一对多的消息发布，在设计构件的时候尽量实现了构件的独立性，减少与其他构件之间的耦合程度，仅通过数据（即消息字符串）的传输来串联起整个系统。我所设计的系统类图如下所示：



4.2 结果分析

本次实习基本上完成了目标，实现了点对点和发布订阅的异步通信，和学长交流后发现实际中这是一个将异步通信做成同步通信的过程，当发布者发送消息的时候，如果用户刚好在线，那就会收到发布者的消息，如果不在线，也可以等到自己上线后从接收端接受发布给自己的消息。本次实习的订阅者都需要去主动接受消息，实际中应该会有更完善的机制来通知订阅者消息的到达。这就是本次实习略有局限的地方了。

五、总结

由于第一次以体系结构的方式去思考实习，在代码风格上不免还是有比较混乱的地方，在我写报告的时候回顾代码，发现很多构件之间的关系并没有我所设想的那样独立，还是会存在一些为了方便而胡乱传参的现象，这种方式也导致了各个构件之间的关系变得更加的紧密，功能更加的臃肿，不利于构件的复用。在写完后尝试修改的时候发现果然如老师所说的，修改代码结构混乱的代码就是一场灾难…最终还是放弃了这个想法。

后来经过反思，我认为在开始写代码之前最重要的是构思好软件总体的框架，如 UML 图等分析的图示都应在最开始的时候就画出来，清晰地表示每个组件之间的联系和功能。同时自己最好要设想好组件之间链接的函数都有哪些输入参数和返回值，不能只有个大概思路，要详细到参数个数和类型，这样才能保证自己在写的时候不会东拼西凑，导致代码的结构变得臃肿不堪。因此在写下一次实习的时候，我就打算先写好对于这个系统的分析，设想好所有接口的链接关系，将所有的情况尽可能的考虑到后，再动手写代码，应该会起到事半功倍的效果。

本实习在实际中的应用场景为通讯，如 QQ 聊天，以及广播推送，如微信公众号。本次实习中所用到的方式是用户通过点击按钮主动获取信息，不过实际中应该会有更好的解决方案来提示用户消息的到达。

参考文献：

[1] boto3 官方文档 https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
[2] SQS 官方文档链接：https://docs.aws.amazon.com/sqs/index.html#lang/zh_cn
[3] SNS 官方文档链接：https://docs.aws.amazon.com/sns/index.html#lang/zh_cn4

附件：

重要组件伪代码如下

```
Def SendMessage(String message):
    Boto3.publish('ARN',message)

Def GetMessageFromText():
    return Text.getstring()

Def GetMessageFromAws():
    Return Boto3.getmessage()

Def ShowMessage(String message):
    Text.setMessage(message)
```

Github 链接：https://github.com/UncoDong/SAInCug