Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

设计模式在Web开发中的实践

设计模式

魏继勋　硕士研究生

2020年10月18日

内容提要

- 背景介绍
- 概念回顾
- Web应用场景下的设计模式
- 消息中间件
- 参考资料

内容提要

- 预期收获
  - 1. 回顾实验室项目开发流程、设计模式基本概念
  - 2. 了解web开发中部分设计模式思想以及应用实例
  - 3. 了解消息中间件（异步、解耦、削峰）

背景介绍

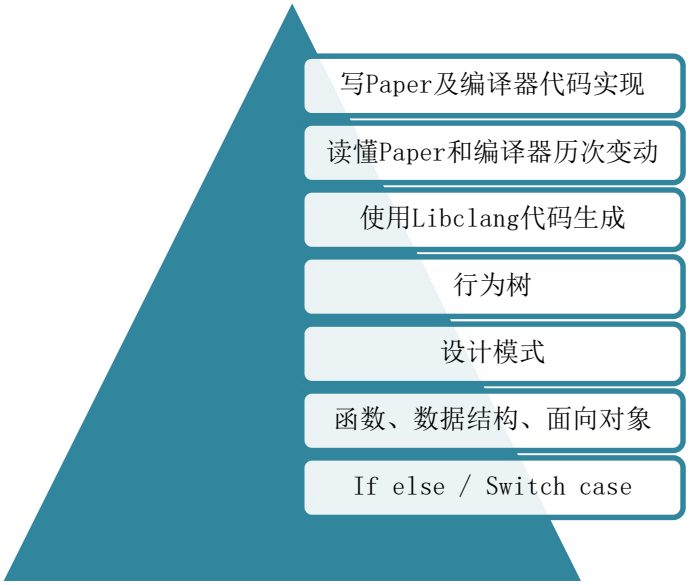
- 软件架构设计时，根据不同的抽象层次可分为三种不同层次的模式
  - 代码模式：利用一个特定的编程语言的特点来实现一个组件的某些特定的方面或关系
  - 设计模式：相互通讯的组件中重复出现的结构
  - 架构模式：软件系统里的基本的结构组织或纲要
- C++实现图书管理系统
  - 代码模式：作者类、图书类
  - 设计模式：图书作者继承关系
  - 架构模式：数据库、UI、Web交互

背景介绍

- 什么是设计模式?
  - 模式是一种可复用的解决方案，可用于解决软件设计中遇到的常见问题
    - 已经验证的解决方案
    - 很容易被复用
    - 富有表达力

背景介绍

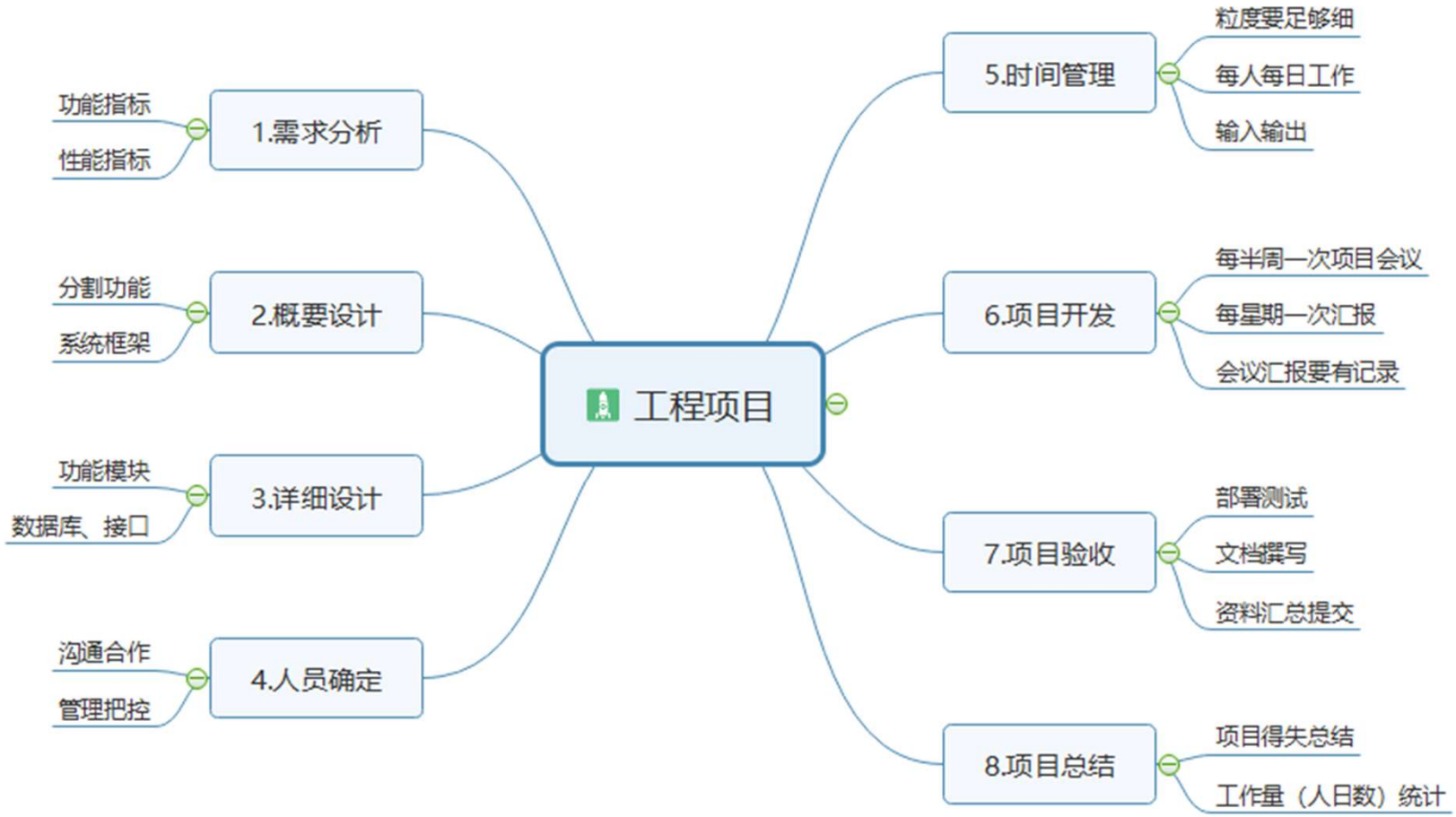
- 为什么要用到设计模式?
  - 增强代码可重用性、易读性、可靠性
  - 经验不足开发人员快速学习软件设计，有经验开发人员问题最佳解决方案
  - 构建大型软件设计大局观



概念回顾

概念回顾

- 项目开发流程


工程项目
1.需求分析
功能指标
性能指标
2.概要设计
分割功能
系统框架
3.详细设计
功能模块
数据库、接口
4.人员确定
沟通合作
管理把控
5.时间管理
粒度要足够细
每人每日工作
输入输出
6.项目开发
每半周一次项目会议
每星期一次汇报
会议汇报要有记录
7.项目验收
部署测试
文档撰写
资料汇总提交
8.项目总结
项目得失总结
工作量（人日数）统计

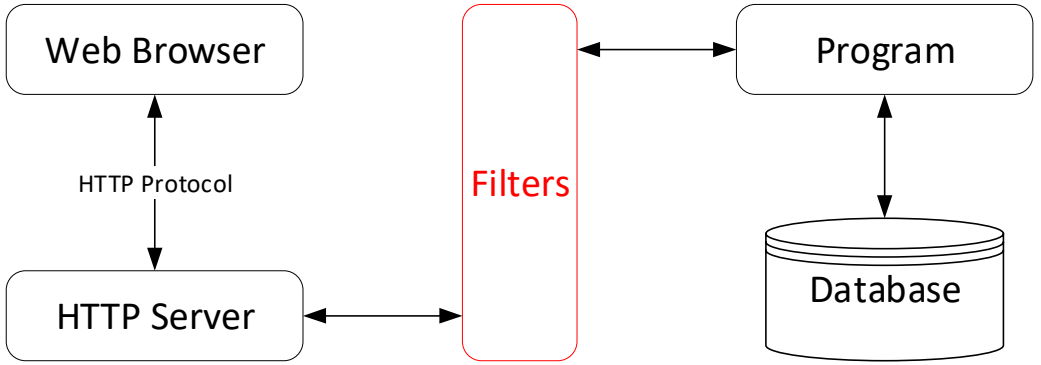
概念回顾

- 设计模式有哪些内容?
  - 4类共33种设计模式
    - 创建型模式
    - 结构型模式
    - 行为型模式
    - J2EE 模式
  - ……

Web应用场景下的设计模式

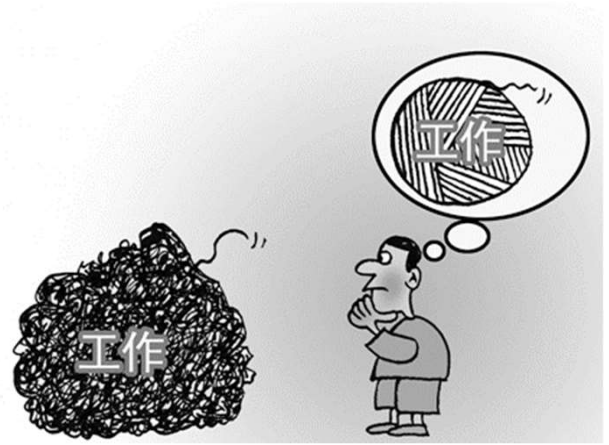
过滤器模式

- 过滤器模式
  - 使用不同的标准来过滤一组对象，通过逻辑运算以解耦的方式把它们连接起来。这种类型的设计模式属于结构型模式，它结合多个标准来获得单一标准。
- Web应用场景
  - 登录验证、权限管理
  - URL统一编码
  - 敏感词汇过滤
  - ……



过滤器模式

- 任务需求
  - 过滤用户所提交文字中敏感内容
    - 实时聊天（短文本）
    - 博客文章（长文本）
- 需求分析
  - 长短文本检索算法一样吗?
  - 如何界定长短文本?
  - 中英文、大小写怎么处理?
  - 是否需要在多个功能模块使用?
  - 数据库是原文存储还是过滤后存储?
  - ……



过滤器模式

- 应用过滤器模式思想
  - 将多个标准结合成单一标准
  - 过滤标准能够随时横向拓展
  - 一个过滤器实体与其他实体独立
  - ……

SimpleFilter
- keywords : set
+ parse(path : string) : set + filter(message : string) : string ……

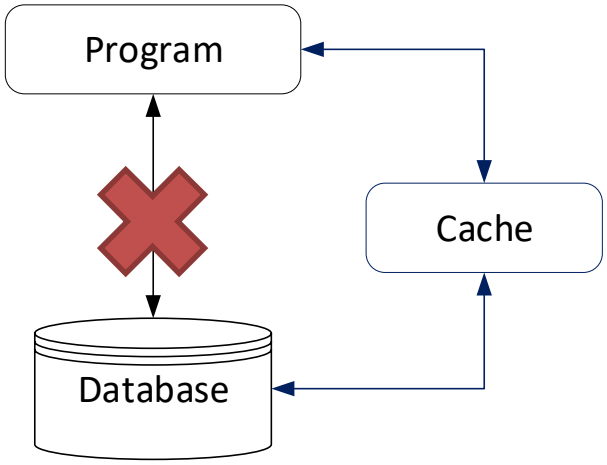
```
01. class SimpleFilter():
02.     '''''Filter Messages from keywords.
03.
04.     very simple filter implementation:
05.
06.     >>> f = NaiveFilter()
07.     >>> f.keywords.add("bad")
08.     >>> f.filter("hello bad boy")
09.     hello *** boy
10.     '''
11.
12.     def __init__(self):
13.         self.keywords = set([])
14.
15.     def parse(self, path):
16.         for keyword in open(path):
17.             self.keywords.add(keyword.strip().decode('utf-8').lower())
18.
19.     def filter(self, message, repl="***"):
20.         message = str(message).lower()
21.         for kw in self.keywords:
22.             message = message.replace(kw, repl)
23.         return message
```

享元模式

- 享元模式
  - 用于减少创建对象的数量，以减少内存占用和提高性能。这种类型的设计模式属于结构型模式，它提供了减少对象数量从而改善应用所需的对象结构的方式。
- Web应用场景
  - 缓存
  - 数据库的数据池
  - HTML模板
  - ……

享元模式

- 优点
  - 减少对象的创建
  - 降低系统的内存
  - 提高效率
- 缺点
  - 提高了系统的复杂度，分离出外部状态和内部状态
  - 外部状态具有固有化的性质，不随着内部状态的变化而变化

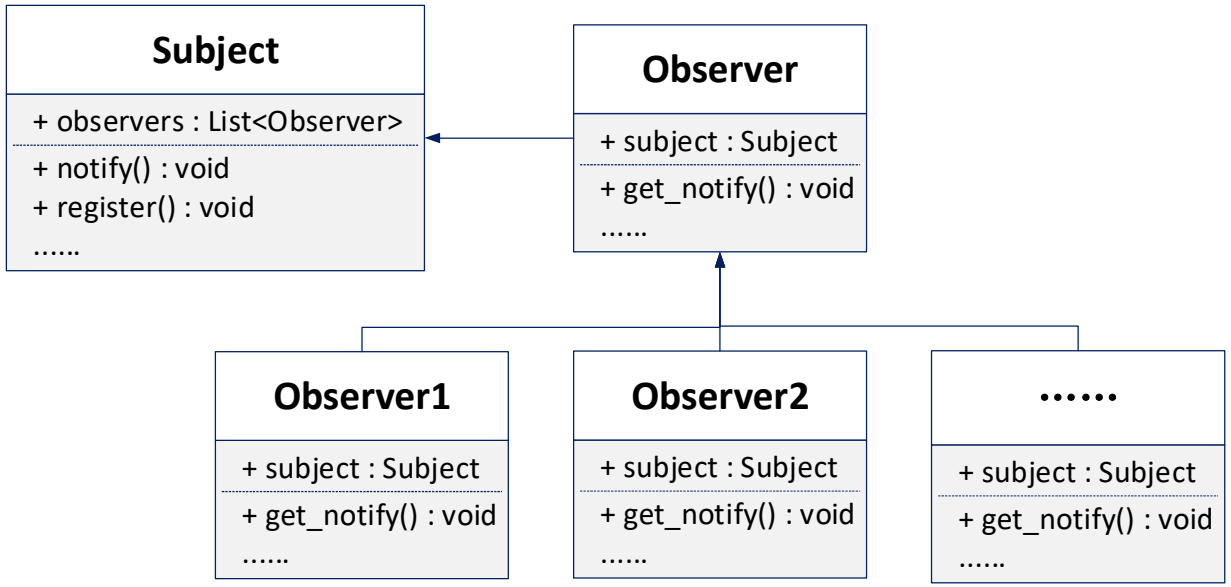


享元模式

- HTML缓存
  - 内存
  - 文件
  - 设备

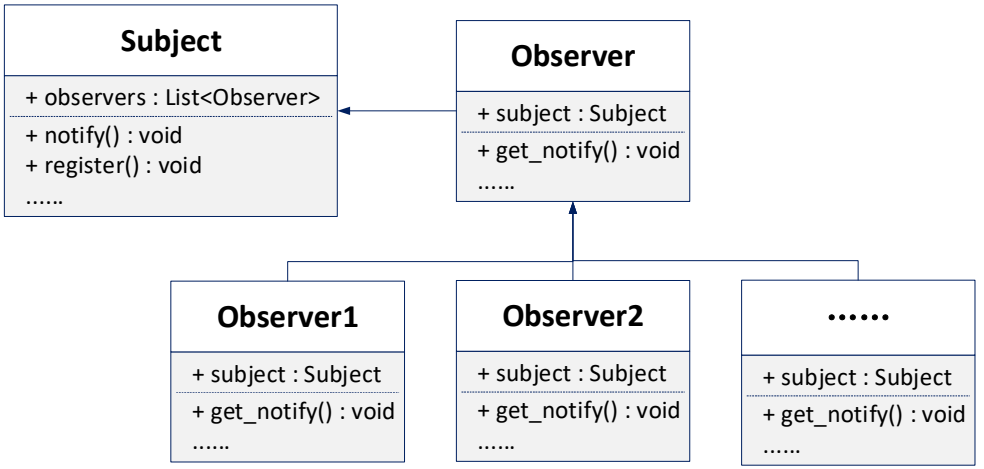
观察者模式

- 观察者模式
  - 在对象之间定义一对多的依赖，当一个对象改变状态，依赖它的对象都会收到通知，并自动更新。是一种行为模式。
- Web应用场景
  - 事件系统
  - 通知订阅发布
  - ……



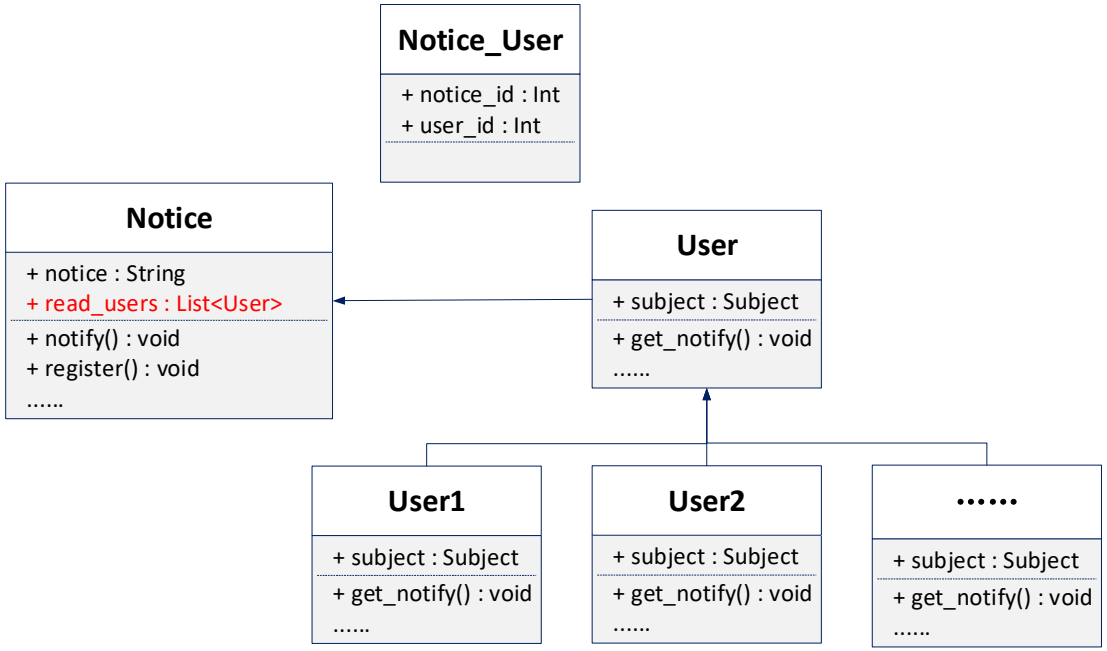
观察者模式

- 观察者模式有两种通知方式
  - 推模型
    - 主题对象向观察者推送主题详细信息，不管其是否需要。
  - 拉模型
    - 主题对象通知观察者时，只传递少量信息。如观察者需要具体信息，由观察者主动到主题对象中获取。
  - 推模型可能会使得观察者对象难以复用，不同观察者get_notify()不同

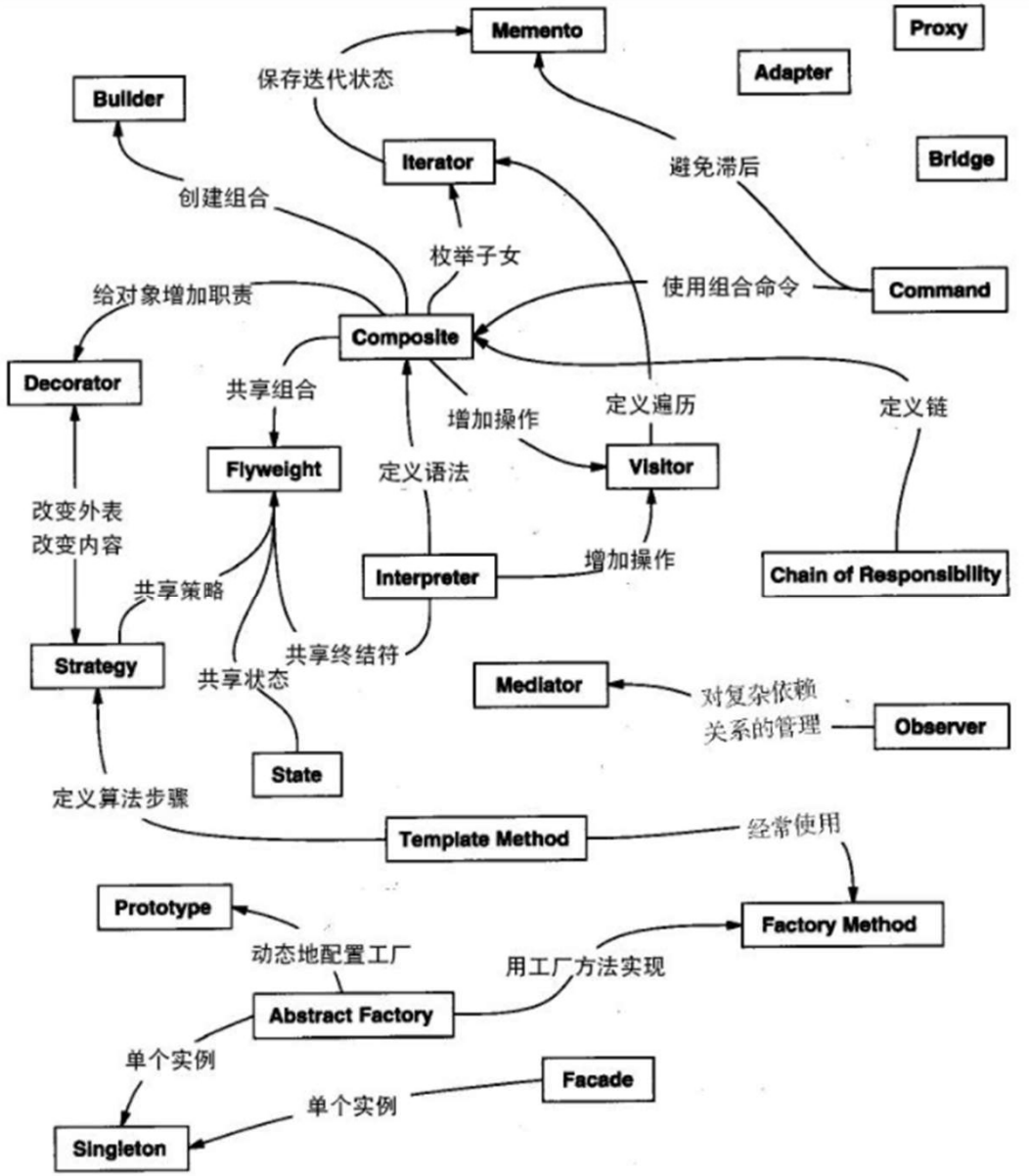


观察者模式

- 任务需求
  - 实现通知公告
    - 管理员发布新的公告
    - 用户上线后收到未读公告提示
    - 用户选择查看未读公告还是忽略
- 一种实现方案
  - 借鉴观察者模式
  - 数据表驱动



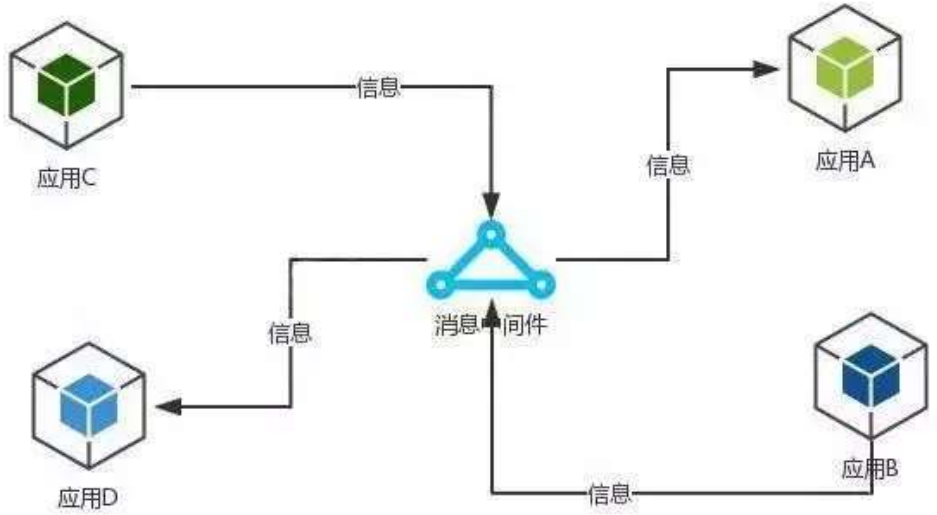
设计模式


Memento
Proxy
保存迭代状态
Adapter
Builder
Iterator
避免滞后
Bridge
创建组合
枚举子女
给对象增加职责
使用组合命令
Command
Composite
Decorator
共享组合
增加操作
定义遍历
定义链
Flyweight
定义语法
Visitor
改变外表
改变内容
共享策略
Interpreter
增加操作
Chain of Responsibility
Strategy
共享终结符
共享状态
Mediator
对复杂依赖
关系的管理
Observer
State
定义算法步骤
Template Method
经常使用
Prototype
Factory Method
动态地配置工厂
用工厂方法实现
Abstract Factory
单个实例
Facade
单个实例
Singleton

消息中间件

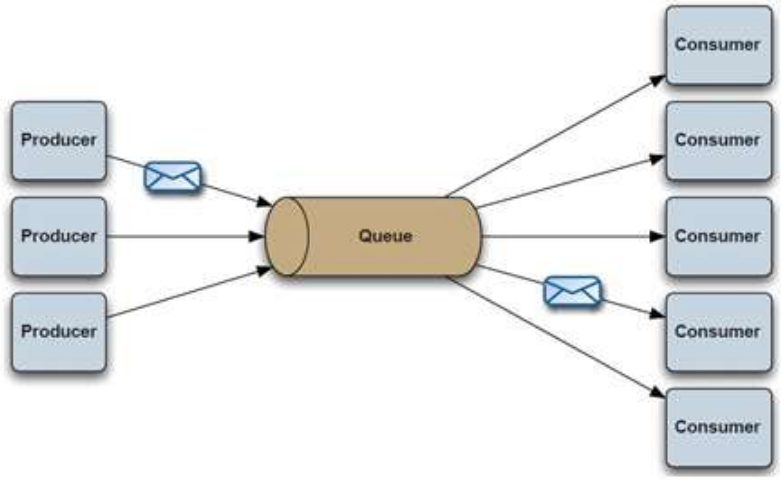
消息中间件

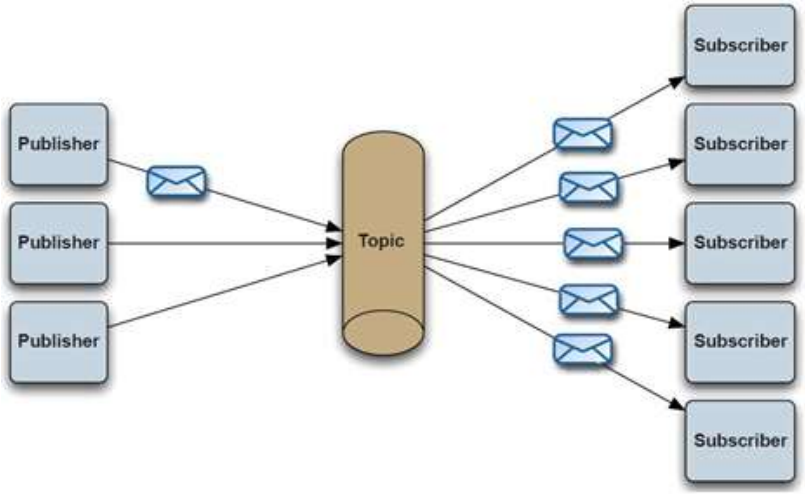
- 消息队列中间件
  - 利用高效可靠的消息传递机制进行与平台无关的数据交流，并基于数据通信来进行分布式系统的集成。
    - 提供消息传递和消息排队模型
    - 在分布式环境下提供应用解耦、弹性伸缩、冗余存储、流量削峰、异步通信、数据同步等等功能



消息中间件

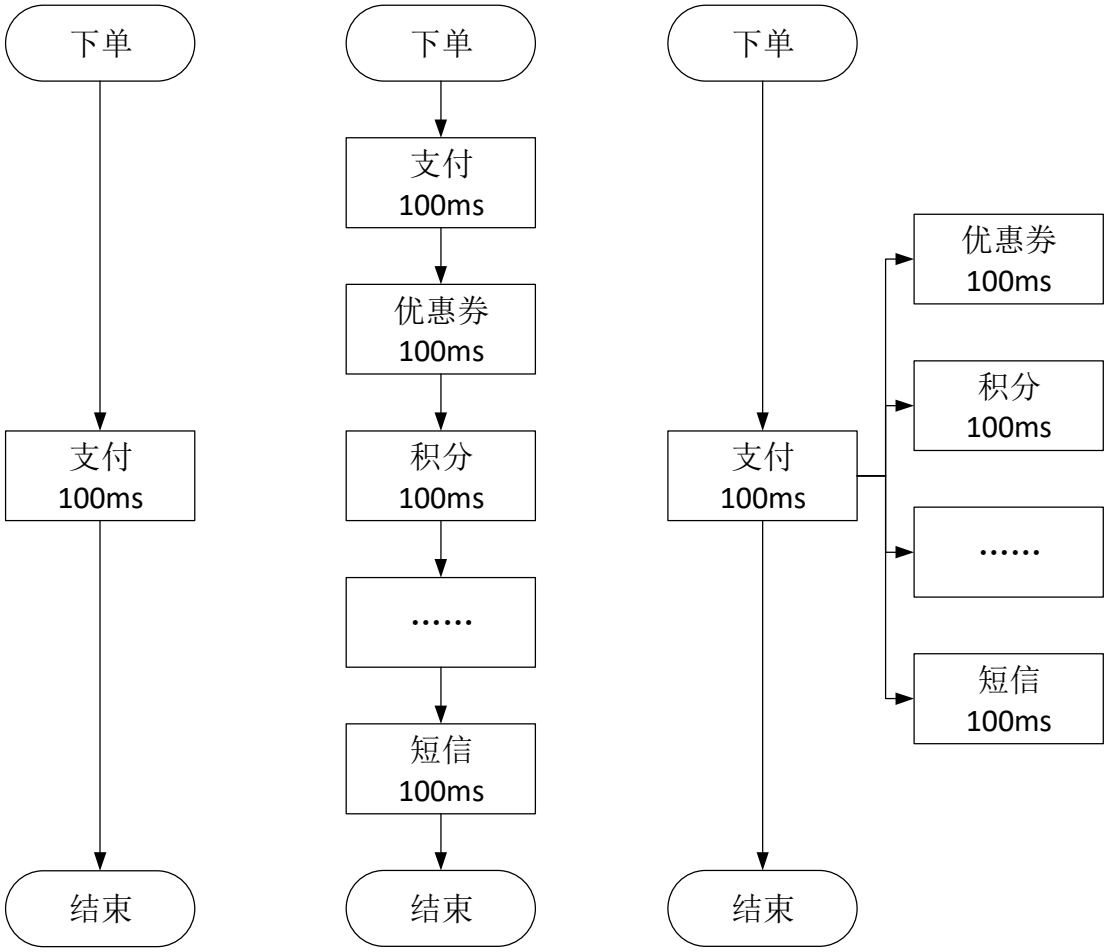
- 常用的消息队列中间件
  - RabbitMQ
  - ActiveMQ
  - RocketMQ
  - Redis（不可靠）
  - ……
- 消息模式
  - 点对点模式
  - 发布订阅模式





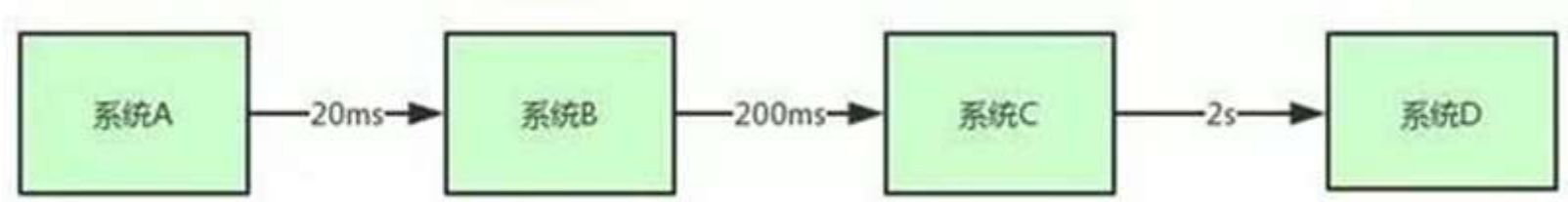
消息中间件

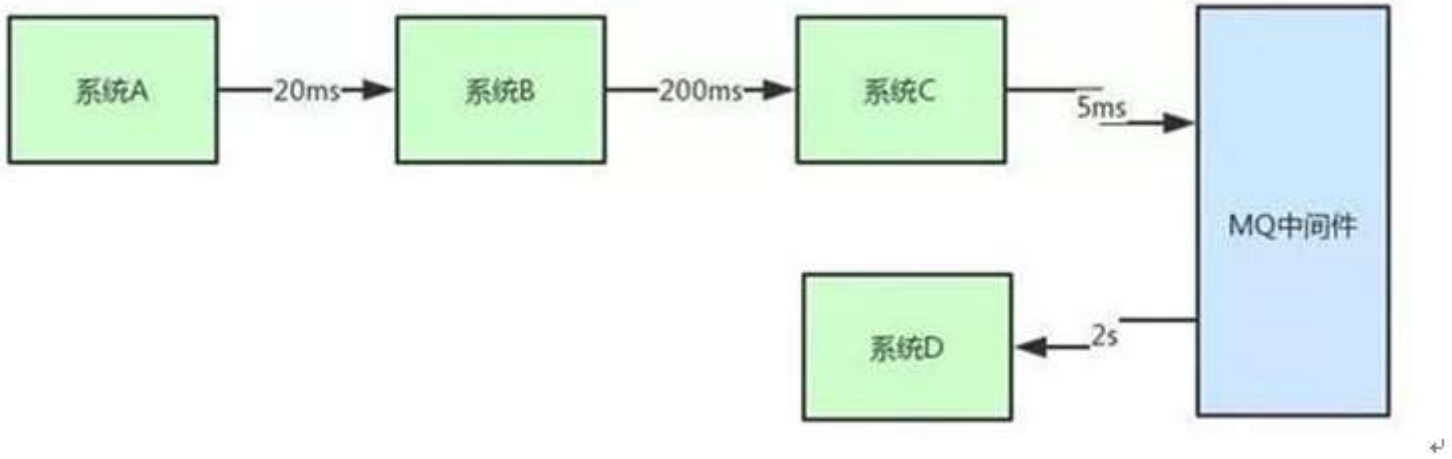
- 异步
- 解耦
- 消峰



消息中间件

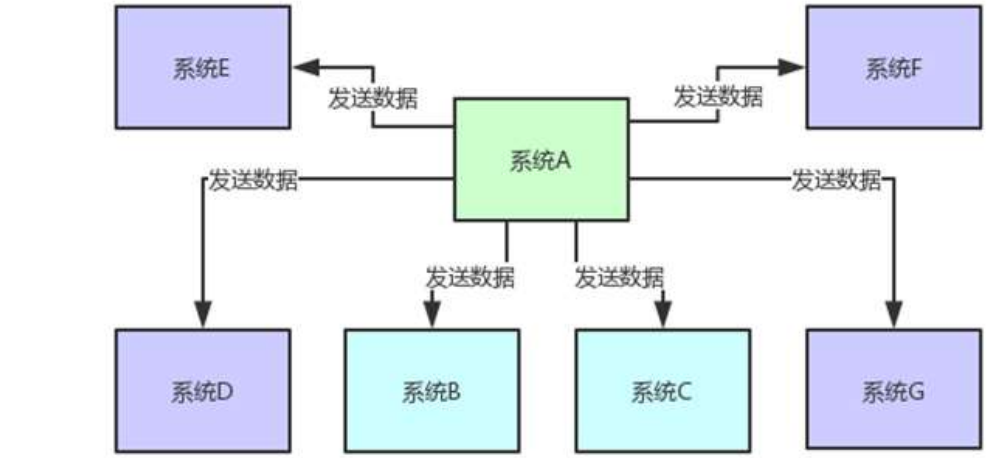
- 异步

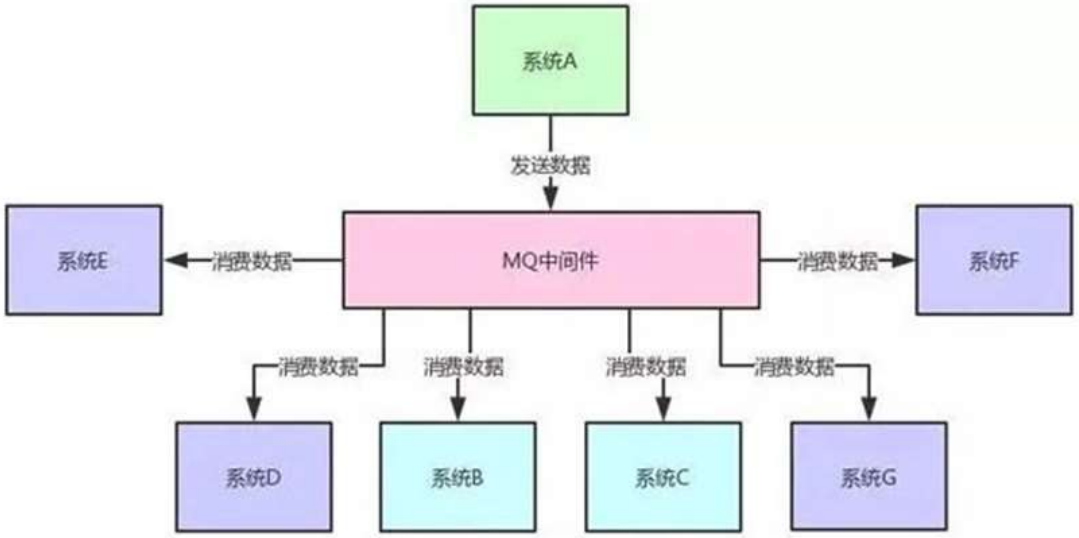

系统A
20ms
系统B
200ms
系统C
2s
系统D


系统A
20ms
系统B
200ms
系统C
5ms
MQ中间件
系统D
2s

消息中间件

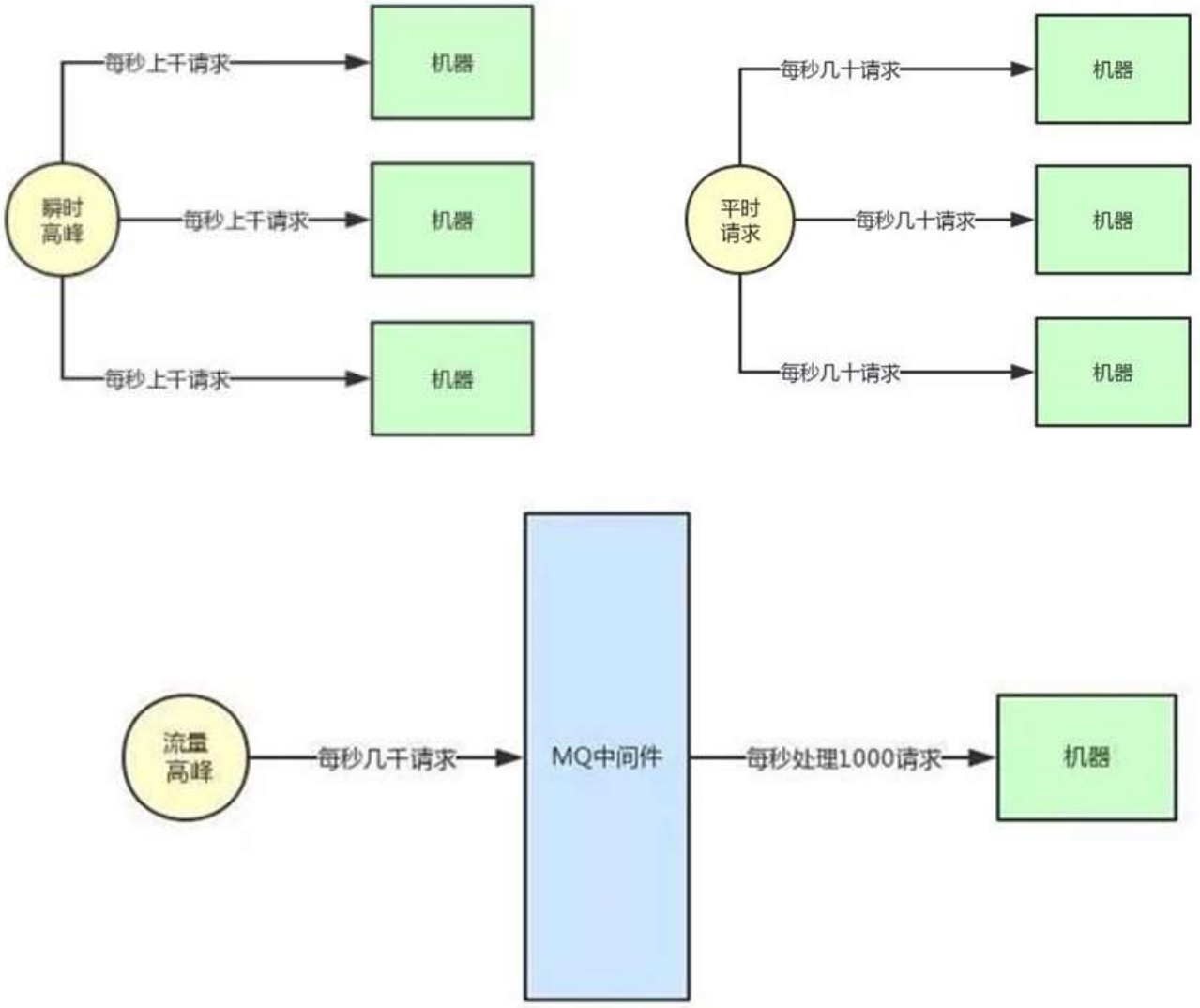
- 解耦


系统E
发送数据
系统F
发送数据
系统A
发送数据
发送数据
发送数据
发送数据
系统D
系统B
系统C
系统G


系统A
发送数据
系统E
消费数据
MQ中间件
消费数据
系统F
消费数据
消费数据
消费数据
消费数据
系统D
系统B
系统C
系统G

消息中间件

- 消峰

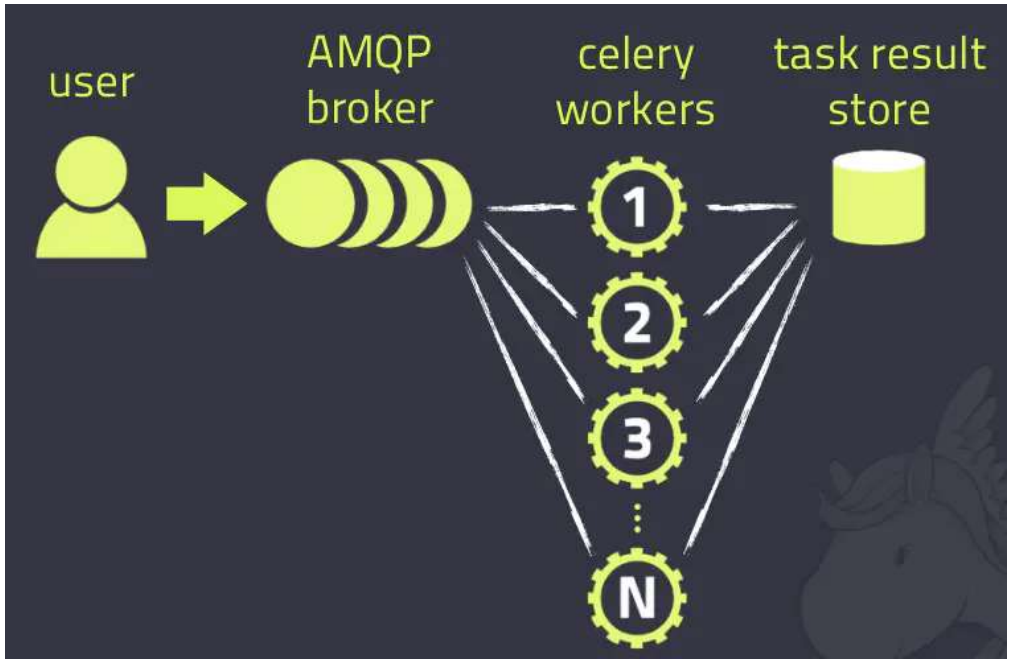

瞬时
高峰
每秒上千请求
每秒上千请求
每秒上千请求
机器
机器
机器
平时
请求
每秒几十请求
每秒几十请求
每秒几十请求
机器
机器
机器
流量
高峰
每秒几千请求
MQ中间件
每秒处理1000请求
机器

消息中间件

- 缺点
  - 增加系统复杂性
    - 需要考虑重复、消息丢失、消息执行顺序
  - 数据一致性
    - 分布式系统中动态数据不会完全一致
  - 系统可用性
    - 中间件宕机导致系统崩溃

消息中间件

- Python中分布式任务调度
  - Celery + RabbitMQ
    - 异步任务调用
    - 分布式任务调度



消息中间件

- 安装RabbitMQ

```
01. #  安装
02. sudo apt-get install rabbitmq-server
03. # 添加用户
04. sudo rabbitmqctl add_user myuser mypassword
05. # 添加vhost
06. sudo rabbitmqctl add_vhost myvhost
07. # 为user设置tag，administrator，monitoring，policymaker，management
08. sudo rabbitmqctl set_user_tags myuser mytag
09. # 权限
10. sudo rabbitmqctl set_permissions -p myvhost myuser ".*" ".*" ".*"
11. # celery
12. broker_url = 'amqp://myuser:mypassword@localhost:5672/myvhost'
```

消息中间件

- 使用Celery

```
01. from celery import Celery
02.
03. app = Celery('tasks', broker='pyamqp://test:123456@10.15.7.120:5672/test')
04.
05. @app.task
06. def add(x, y):
07.     return x + y
```

消息中间件

- 使用Celery
  - celery -A tasks worker --loglevel=INFO

```
终端    问题    输出    调试控制台
(venv) close@DESKTOP-MH12AKS:~/code/Python$ celery -A tasks worker --loglevel=INFO

 -------------- celery@DESKTOP-MH12AKS v5.0.0 (singularity)
--- ***** -----
-- ******* ---- Linux-4.19.128-microsoft-standard-x86_64-with-glibc2.29 2020-10-18 16:17:02
- *** --- * ---
- ** ---------- [config]
- ** ---------- .> app:         tasks:0x7fee8735a7c0
- ** ---------- .> transport:   amqp://test:**@10.15.7.120:5672/test
- ** ---------- .> results:     disabled://
- *** --- * --- .> concurrency: 4 (prefork)
-- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
--- ***** -----
 -------------- [queues]
                .> celery           exchange=celery(direct) key=celery


[tasks]
  . tasks.add

[2020-10-18 16:17:02,493: INFO/MainProcess] Connected to amqp://test:**@10.15.7.120:5672/test
[2020-10-18 16:17:02,508: INFO/MainProcess] mingle: searching for neighbors
[2020-10-18 16:17:03,565: INFO/MainProcess] mingle: all alone
[2020-10-18 16:17:03,608: INFO/MainProcess] celery@DESKTOP-MH12AKS ready.
```

消息中间件

- 使用Celery
  - add.delay(4, 4)

```
(venv) wjx@wjx-virtual-machine:~/Test$ ls
__pycache__  tasks.py  venv
(venv) wjx@wjx-virtual-machine:~/Test$ python
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from tasks import add
>>> add.delay(4, 4)
<AsyncResult: de857a35-d1da-455e-9785-a0ecad408345>
>>>
```

```
[tasks]
  . tasks.add

[2020-10-18 16:17:02,493: INFO/MainProcess] Connected to amqp://test:**@10.15.7.120:5672/test
[2020-10-18 16:17:02,508: INFO/MainProcess] mingle: searching for neighbors
[2020-10-18 16:17:03,565: INFO/MainProcess] mingle: all alone
[2020-10-18 16:17:03,608: INFO/MainProcess] celery@DESKTOP-MH12AKS ready.
[2020-10-18 16:20:22,989: INFO/MainProcess] Received task: tasks.add[de857a35-d1da-455e-9785-a0ecad408345]
[2020-10-18 16:20:22,990: INFO/ForkPoolWorker-2] Task tasks.add[de857a35-d1da-455e-9785-a0ecad408345] succeeded in 0.0001622999999995045s: 8
```

参考资料

- 《Design Patterns - Elements of Reusable Object-Oriented Software》
- RabbitMQ：https://www.rabbitmq.com/
- Celery：https://docs.celeryproject.org/en/stable/index.html

道德经

知人者智，自知者明。
胜人者有力，自胜者
强。知足者富。强行
者有志。不失其所者
久。死而不亡者，寿。

谢谢！