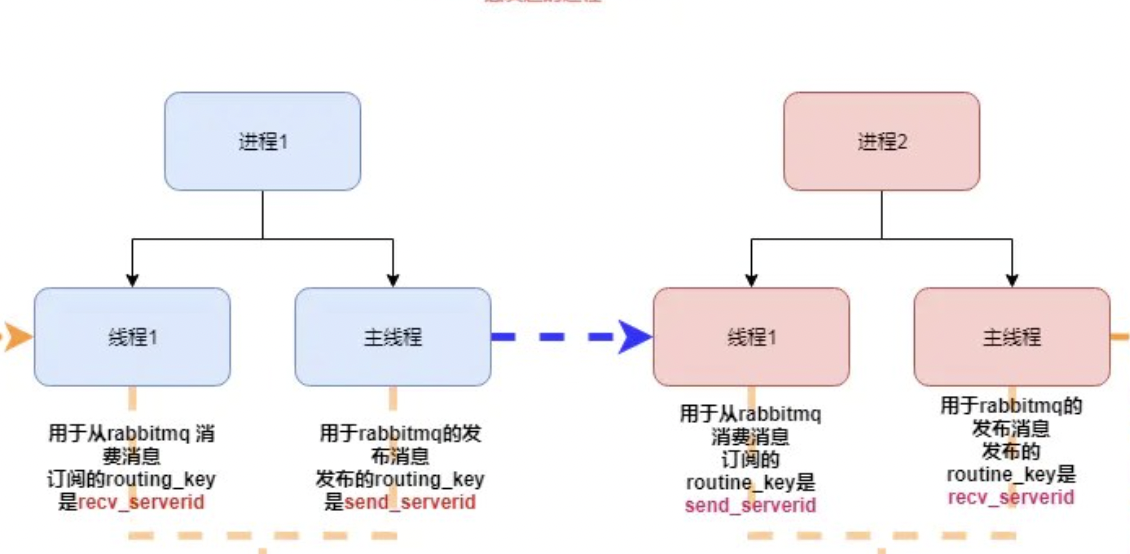
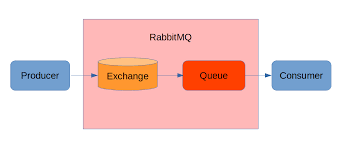
**整体上主要是从前端Jinja的template页面触发事件，然后通过Flask API传递到pika中间件，pika然后传递到rabbitmq**

**Flask** 是一个基于 Python 的轻量级 Web 框架，WSGI 工具箱采用 Werkzeug，模板引擎使用 Jinja2。由于其不依赖于特殊的工具或库，并且没有数据抽象层、表单验证或是其他任何已有多种库可以胜任的功能，从而保持核心简单、易于扩展，而被定义为"微"框架。但是，Flask 可以通过扩展来添加应用功能。并且 Flask 具有自带开发用服务器和 debugger、集成单元测试和 RESTful 请求调度 (request dispatching)、支持 secure cookie 的特点。我们就主要使用Flask的网站部分和wsgi写API部分。

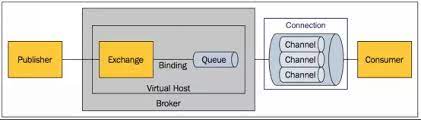


**Pika** 因为技术栈不同，rabbitmq是erlang开发，所以python中使用pika模块来处理与rabbitmq的连接。主要流程是：

1. 建立到rabbitmq的连接
2. 建立通道
3. 声明使用的队列（生产者和消费者都要声明，因为不能确定两者谁先运行）
4. 生产/消费
5. 持续监听/关闭连接

**Jinja2** 是基于 Python 的模版引擎，支持 Unicode，具有集成的沙箱执行环境并支持选择自动转义。Jinja2 拥有强大的自动 HTML 转移系统，可以有效的阻止跨站脚本攻击；通过模版继承机制，对所有模版使用相似布局；通过在第一次加载时将源码转化为 Python 字节码从而加快模版执行时间。我们的网站看得到的页面部分是这块开发。  
  


**RabbitMq** 消息队列是一种应用程序的通信方法，应用程序可通过读写出入对立的消息进行通信。MQ 是一种消费者-生产者 （Producer-Customer）模式的实现。生产者-消费者模式由生产者、消费者和缓存区三个模块构成。缓存区作为一个中介的存在，生产者将数据放入缓存区，消费者从缓存区取出数据。本系统中，Flask 作为生产者，Salesforce 作为消费者，而 MQ 则是中间的缓存区。应用生产者-消费者模式能够有效的降低两者之间的耦合，减少互相之间的依赖；由于缓存区的存在，消费者无需直接从生产者处获取数据，能够支持并发任务、减少阻塞。



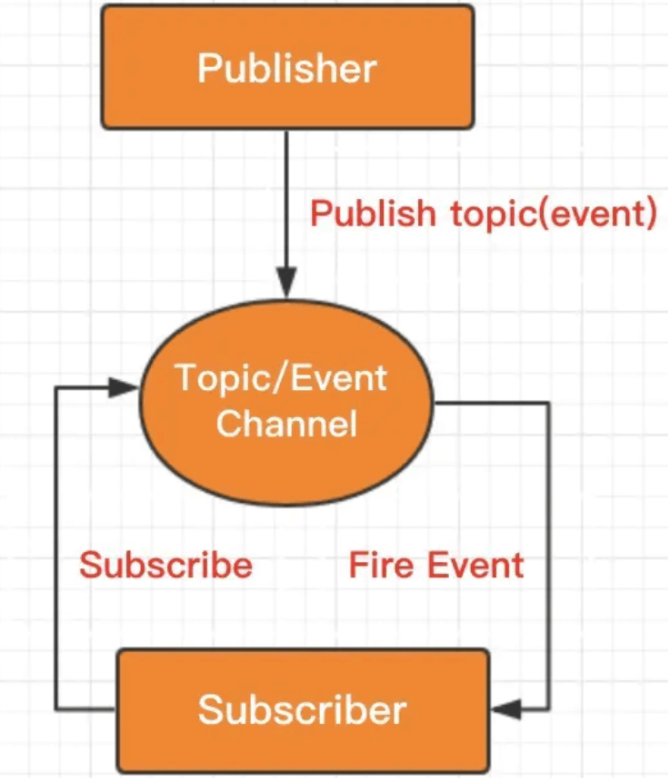
RabbitMQ 则是由 erlang 开发的 AMQP（高级消息队列协议）的开源实现，作为一个消息队列管理工具与 Celery 集成后，负责处理服务器之间的通信任务。RabbitMQ 的使用过程如下：

* 客端连接到消息队列服务器并打开一个 channel（目前我们只使用了一个channel，就是**hello** channel）
* 客户端声明一个 exchange、一个 queue，并分别设置相关属性。
* 客户端使用 routing key 在 exchange 与 queue 之间绑定好关系。
* 客户端投递消息到 exchange，exchange 根据消息的 key 和设置好的 binding，将消息投递到队列中。

RabbitMQ 常用的 Exchange Type 有以下三种：

* Fanout：能够将所有发送到该 exchange 的消息投递到所有与它绑定的队列中。
* Direct：把消息投递到那些 binding key 与 routing key 完全匹配的队列中。
* Topic：将消息路由到 binding key 与 routing key 模式匹配的队列中

我们暂时使用了Topic的type，这样方便我们进行扩展，如果需要，我们不光只有从api端接受到hello channel的topic，我们还可以设置从物联网设备端反馈的消息topic，比如message\_warning topic，代表物联网各种报警和提示消息的队列。

****

**设计模式之订阅者模式** 我们模拟接受rabbitmq消息，等待和展示部分使用订阅者模式实现，基于python多进程（如果像使用多个，只需要开启多个命令行即可）。在这个过程中rabbitmq充当 EventBus(事件总线)，可以理解为订阅事件的集合，它提供订阅、发布、取消等功能。订阅者订阅事件，和发布者发布事件，都通过事件总线进行交互。

将来可以模拟多个物联网设备订阅消息，消费消息，甚至反馈互动。