又拍网架构中的消息 / 任务系统

赵钟秋 belltoy

又拍网程序员

2011.10

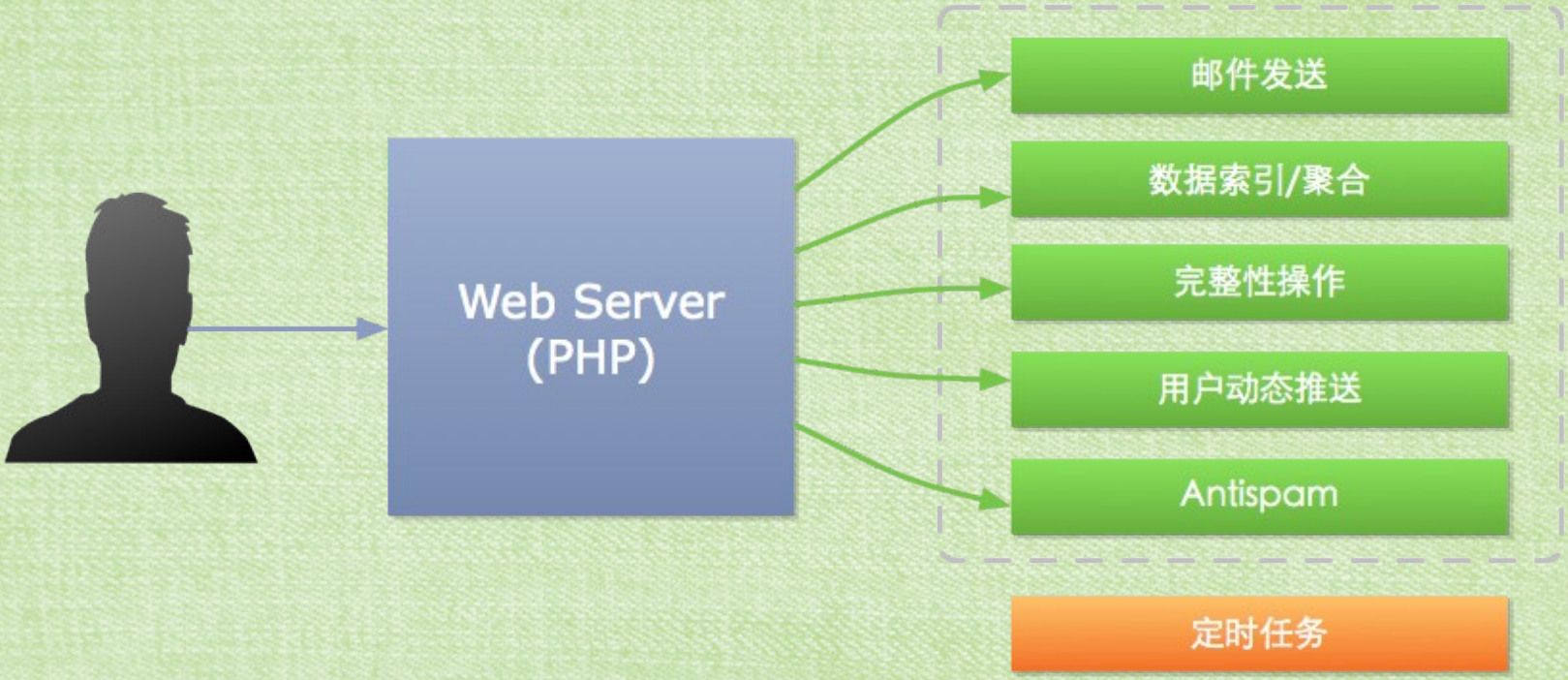
又拍网简介

- 照片分享社区
- 2005 年建站
- 500 万用户
- 超过 3 亿张照片

又拍图片托管简介

- 面向企业用户，提供云存储服务
- 2010 年上线
- 类似 Amazon S3+CloudFront
- 超过 10 亿张图片
- 图片日访问量超过 2 亿次

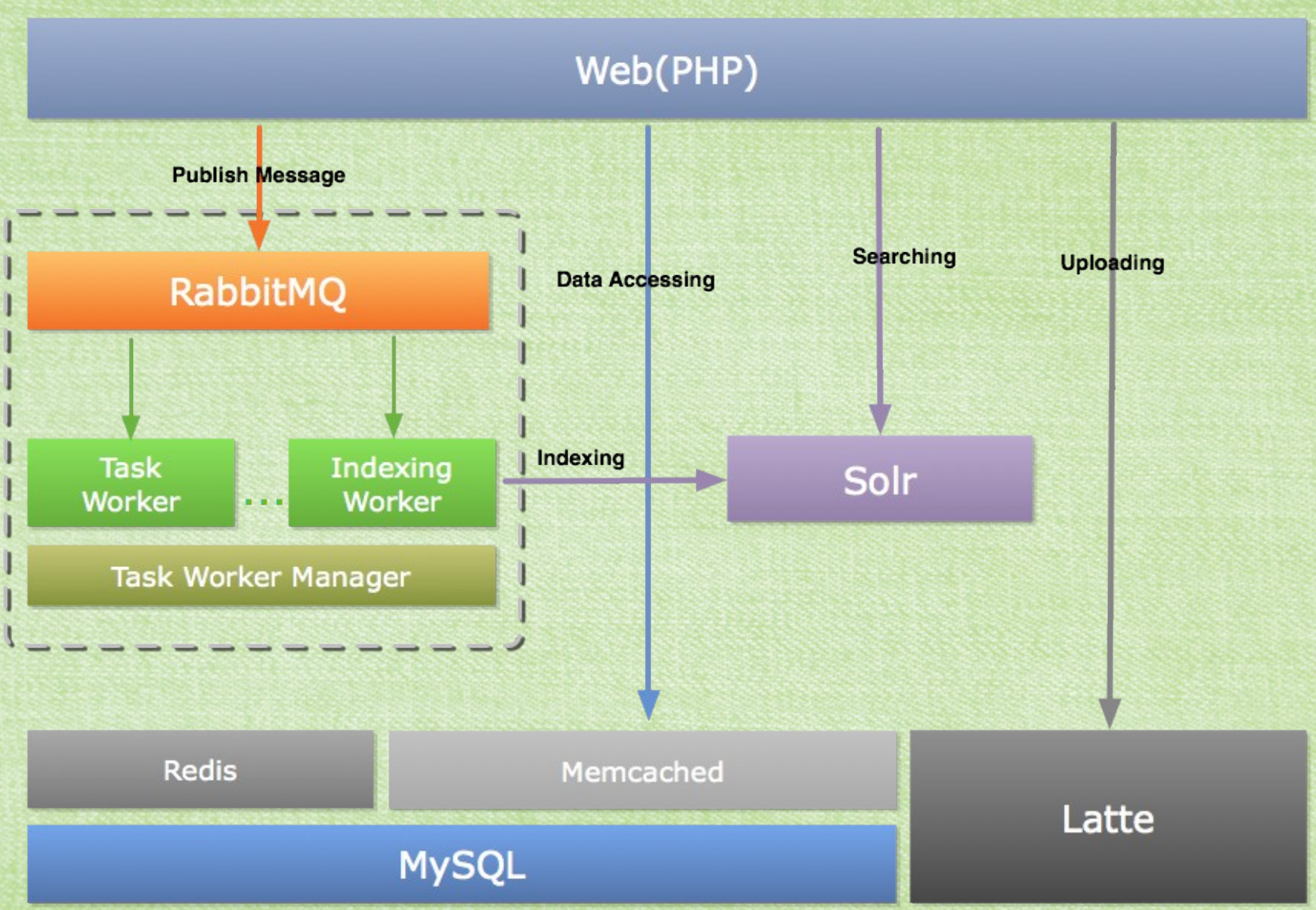
后台任务



后台任务

- 由用户或者定时触发
- 耗时长
- 异步执行

系统架构


Web(PHP)
Publish Message
RabbitMQ
Data Accessing
Searching
Uploading
Task Worker
Indexing Worker
Indexing
Solr
Task Worker Manager
Redis
Memcached
Latte
MySQL

任务系统的组成

- 消息分发
- 进程管理
- 工作进程

RabbitMQ

- 开源
- Erlang 实现
- 高级消息队列协议 (AMQP) 实现
- 分布式

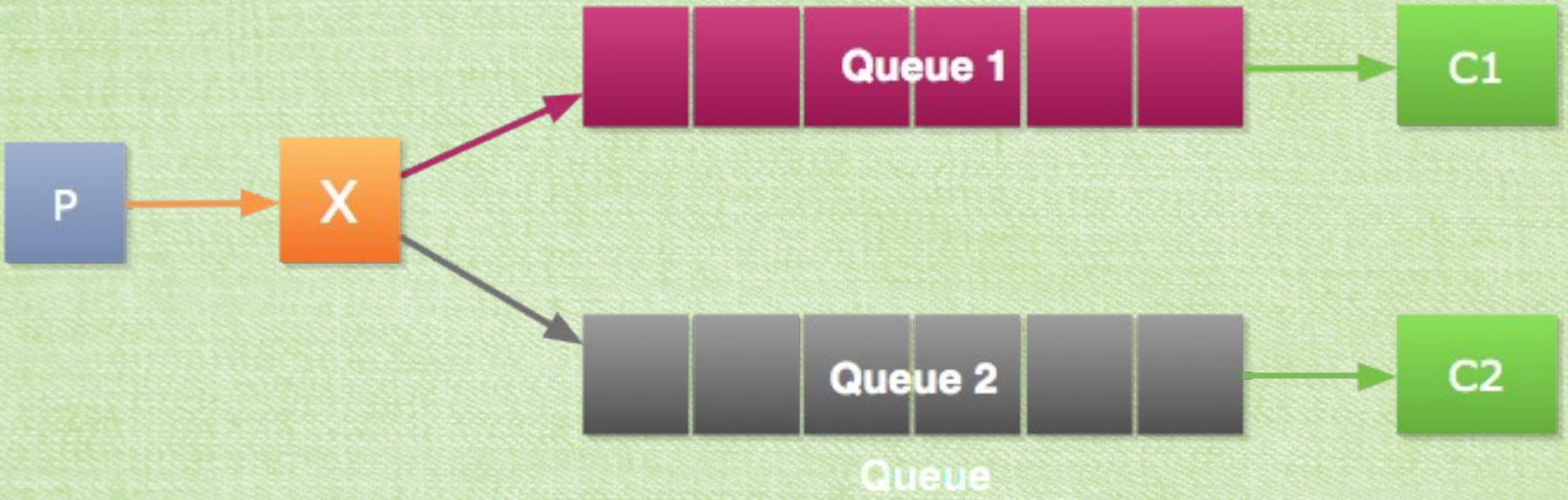
RabbitMQ 组件

- 交换器 (Exchange)
  - direct
  - topic
  - headers
  - fanout
- 队列 (Queue)
- 绑定 (Binding)

RabbitMQ 工作方式

- Work Queues
- Publish/Subscribe
- Routing
- Topics
- RPC

RabbitMQ 工作方式



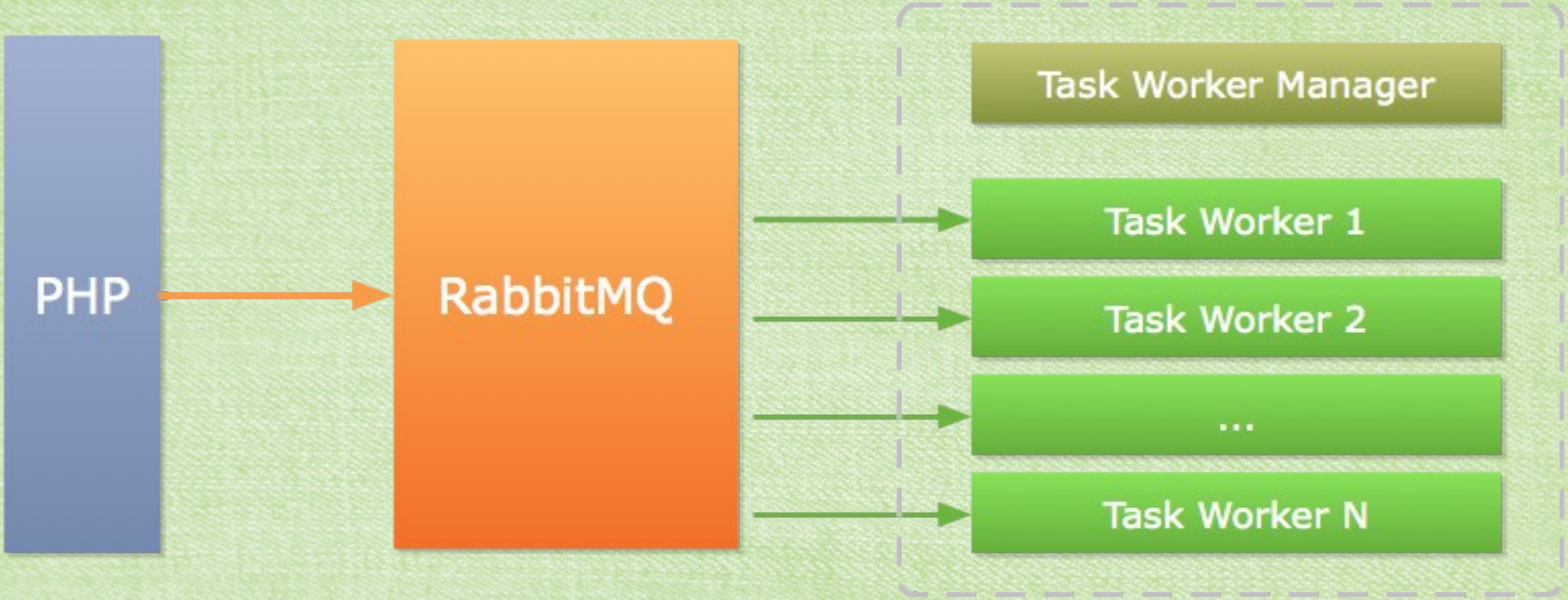
Worker 接收消息

```python
#!/usr/bin/evn python

def consume(cfg):
    exchange_name = 'indexing_exchange'
    queue_name    = 'indexing'
    routing_key   = 'indexing'
    consumer_tag  = 'indexing_msg_consumer'

    chan = get_amqp_channel()
    chan.access_request(vhost)
    chan.queue_declare(queue_name)
    chan.exchange_declare(exchange_name, type='direct')
    chan.queue_bind(queue_name, exchange_name, routing_key)

    chan.basic_consume(queue_name, callback_func, consumer_tag)
    while chan.callbacks:
        chan.wait()

    chan.close()

def callback_func(msg):
    ''' do consume '''
    pass
```

PHP 发送消息

```php
<?php
$mq_conn = amqp_connection_open($host, $port);
$res = amqp_login($mq_conn, $user, $passwd, $vhost);
$mq_chan = 1;
$res = amqp_channel_open($mq_conn, $mq_chan);
$exchange = 'indexing_exchange';
$routing_key = 'indexing';
amqp_basic_publish($mq_conn, $mq_chan,
    $exchange, $routing_key, $message_body, false, false, $options);
```

基于 RabbitMQ 的实现



一个行为触发多个任务

- 添加在原来的任务中
- 发送多个不同的消息
- 采用 Publish/Subscribe 方式

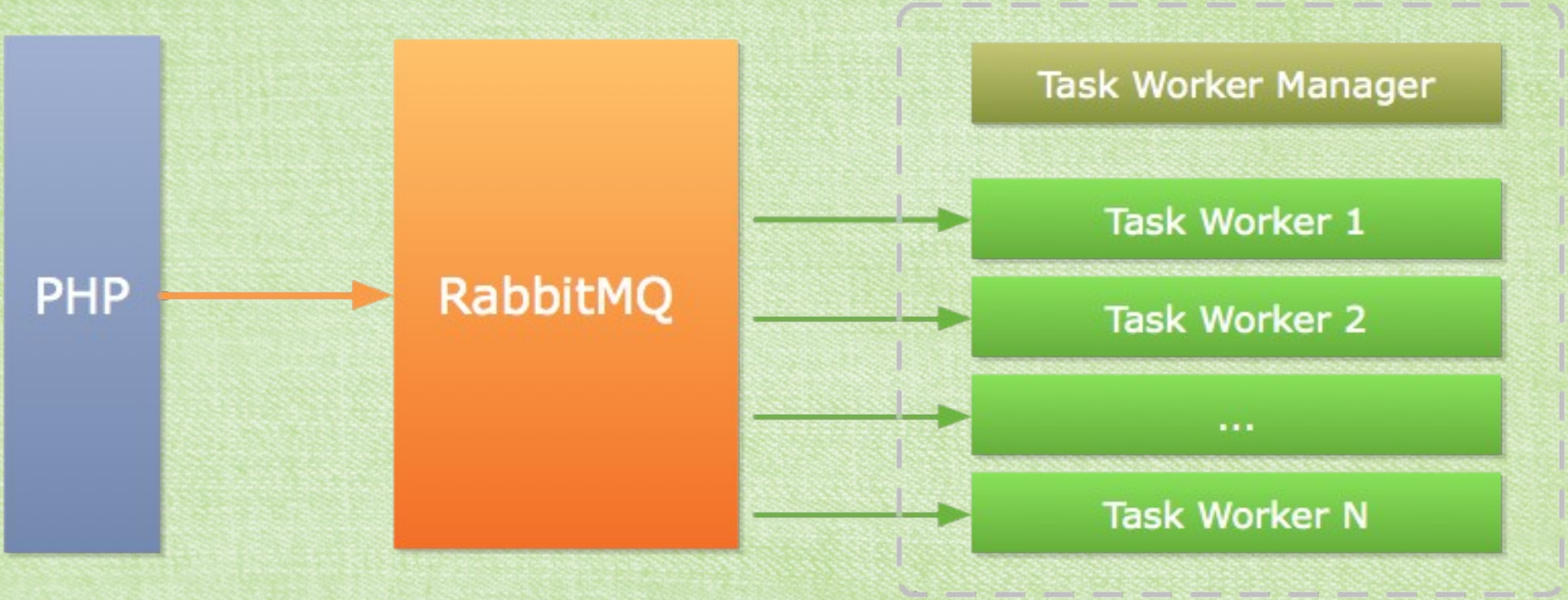
问题出现

- 访问量增加，工作进程增加
- 业务复杂，消息类型增加
- 不能动态配置任务
- 代码经常更新，系统频繁启停
- 可能中断正在执行的任务
- 大量工作进程驻留在内存

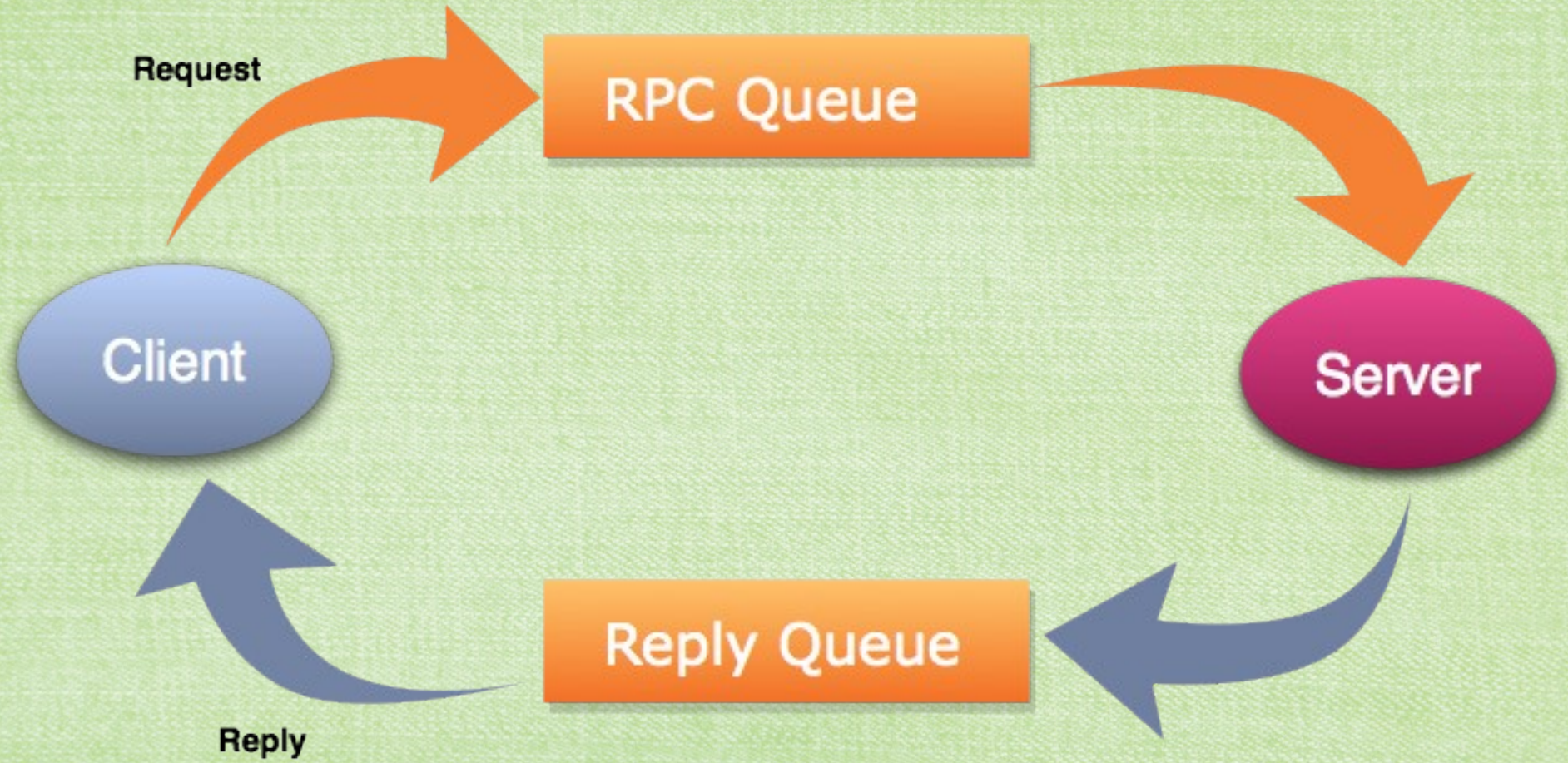
我们的需求

- 使用 Publish/Subscribe 方式
- 简单灵活的配置
- 动态更新代码
- 方便的进程管理
- 空闲时，释放资源
- RPC

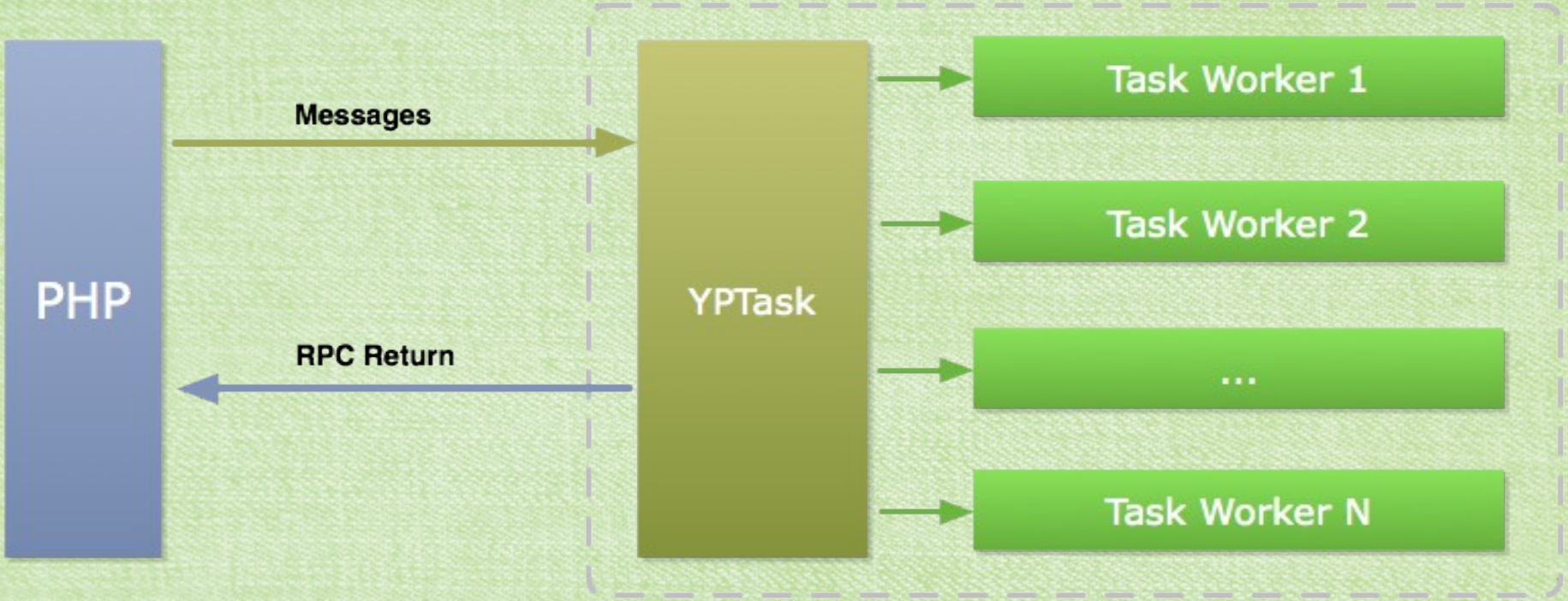
基于 RabbitMQ 的实现



RabbitMQ 的 RPC 实现方式



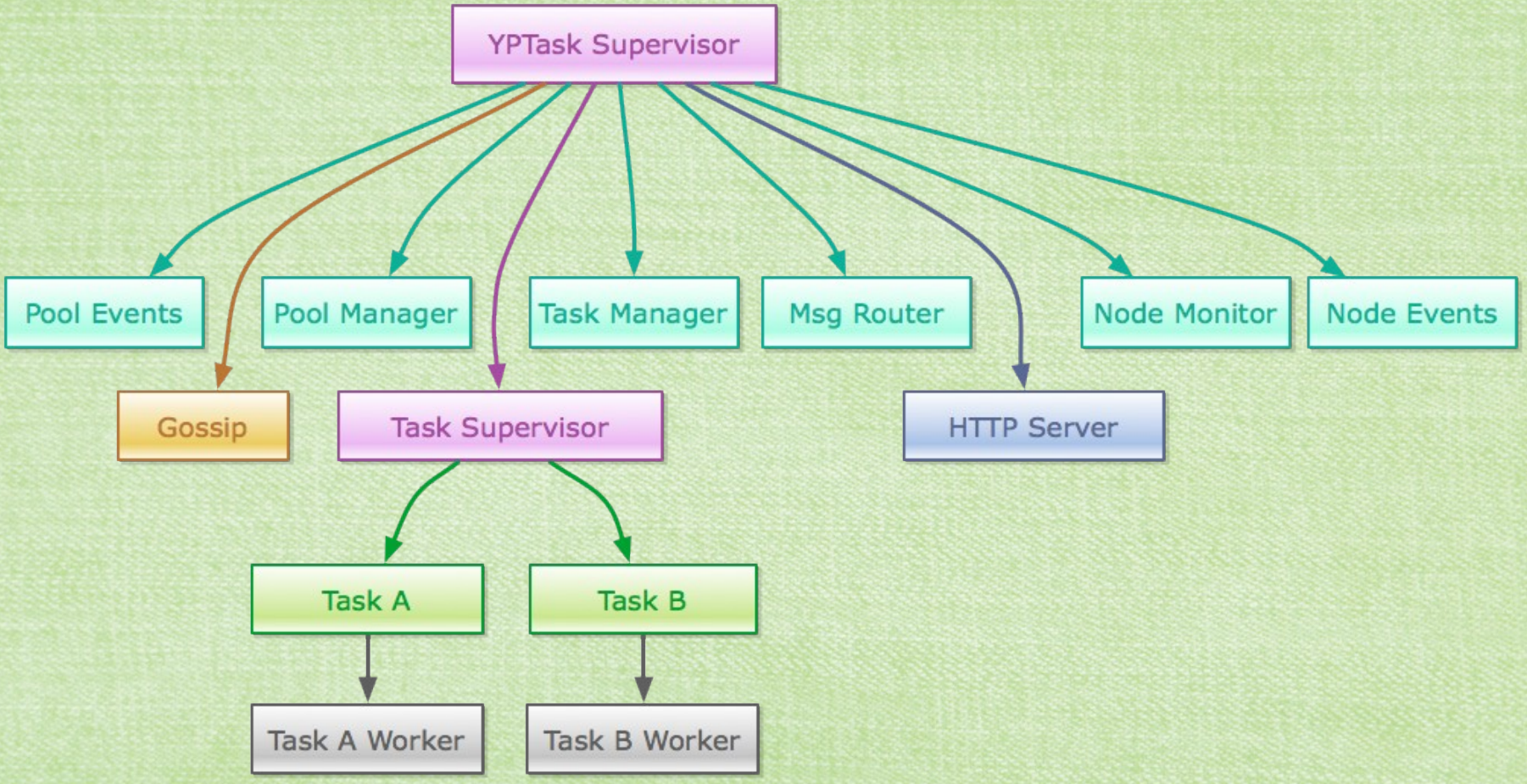
基于 YPTask 的实现



为什么使用 Erlang 实现

- 原生的分布式支持
- 支持代码动态更新
- 成熟的 OTP 方案
- 进程管理方便
- 我们熟悉 Erlang 开发

YPTask 系统内部进程树



分布式

- Erlang 原生的支持
- 动态增删节点
- 去中心化
- 同步各节点的配置

消息分发

- 一种消息类型对应一个消息队列
- 一个任务可以接收多种消息类型
- 一个任务可以动态配置工作进程的数量

定时器

- 类似 crontab 的定时器配置
- 基于消息
- 集群中一种定时器只有一个实例
- 错误处理

API

- Publisher: JSON-RPC
- Subscriber: BERT-RPC

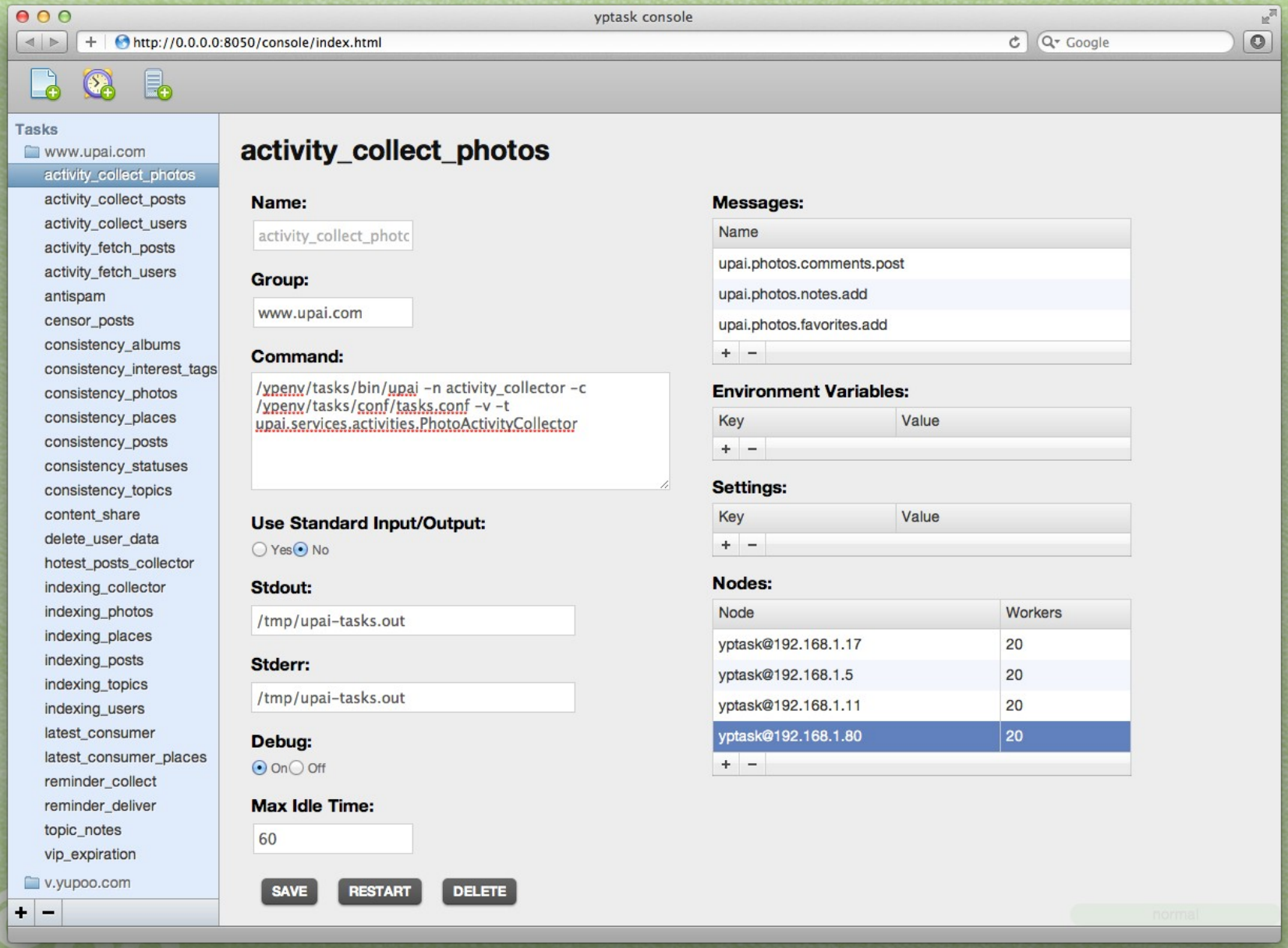
RPC

- 客户端、服务端协议均为 RPC
- Erlang 内部使用 Cast/Call
- 使得更进一步拆分 Web 成为可能

动态配置

- 把消息转发的配置工作统一到 YPTask 中
  - 只要指定消息的名字和消息内容
- 动态配置、动态更新
  - 修改配置之后只要重新启动相应的工作进程

YPTask 系统配置界面



新的消息收发实例

```php
<?php

yptask_cast('upai.photos.upload', $msg);
```

```python
#!/usr/bin/env python

class IndexingMessageCollector(BaseTask):
    def handle_photos_upload(self, msg):
        pass

    def handle_photos_delete(self, msg):
        pass

class AntispamMessageConsumer(BaseTask):
    def handle_photos_upload(self, msg):
        pass

    def handle_something_else(self, msg):
        pass
```

迁移

- 仍然采用 Python 实现工作进程
- 简化消息发布方式，一个事件只发一条消息
- 统一配置管理，业务代码专注于业务逻辑
- 再一次做拆分

YPTask 特点

- 简单的配置
- 灵活的消息分发
- 支持大量的任务
- 能够动态更新代码，不间断运行
- 支持异构的后台任务
- 支持 RPC

TODO

- 消息持久化
- 多种语言支持
- 多种客户端协议支持
- 优化配置管理界面
- 开放源代码

一些经验

- 拆分业务逻辑，让 web 只处理最少的事情
- 拆分系统架构，利用已经实现的工具
- 利用成熟的方案，能够带来高效的实现
- 让处理业务的代码只处理业务，其它事交给别人去做

加入我们

job@yupoo-inc.com

谢谢


QCon
北京站·2012年4月18~20日
www.qconbeijing.com（11月启动）
QCon杭州站官网和资料
www.qconhangzhou.com
全球企业开发大会
INTERNATIONAL
SOFTWARE DEVELOPMENT
CONFERENCE