# ZTQ 异步任务队列

潘俊勇

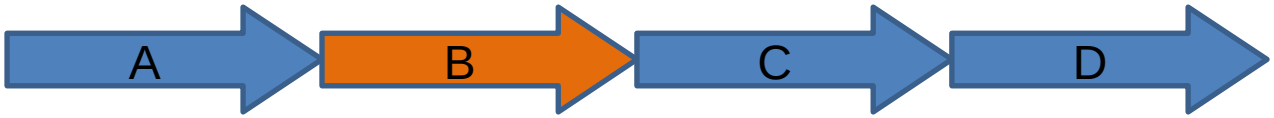
易度云办公
everydo.com

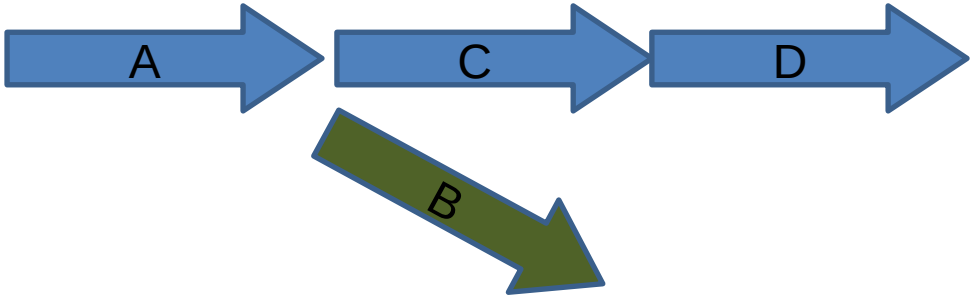
# web 服务中的耗时操作

- 生成 PDF
- 网页抓取
- 游戏数据备份
- 邮件发送
- 短信发送

# 解决之道：异步执行

同步



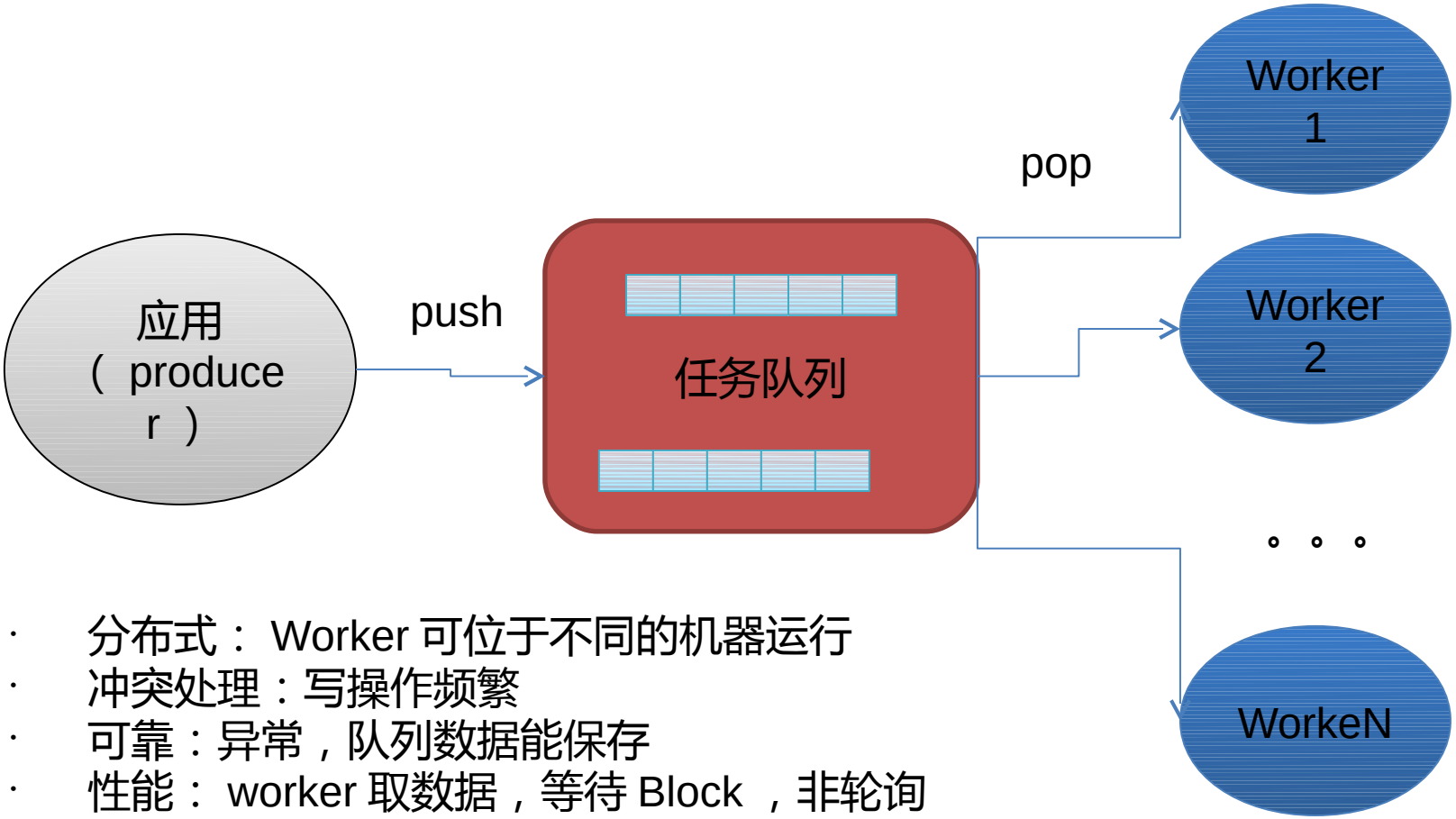




异步：在另外的协程、线程、进程、服务器运行

# 语言级实现

- Scala
- Golang
- erlang

# 异步队列工作原理




- 分布式： Worker 可位于不同的机器运行
- 冲突处理：写操作频繁
- 可靠：异常，队列数据能保存
- 性能： worker 取数据，等待 Block ，非轮询

# 异步队列的更多场合

- 性能优化：尽可能异步
  - 日志记录
  - 消息推送
- 串行化：避免冲突
  xapian 索引只能单写
- 延时 / 定时运行
- 并行计算：
  分割多个任务并行执行

# 队列选型之路

- 数据库方案（ ZODB ： zc.async ）：
  - 轮询查，低效！
  - 频繁写，冲突！
- RabbitMQ ：非常复杂的消息模型
- ZeroMQ ：不支持 Persistent
- Beanstalkd ：需要引入专门的服务器
- Redis ：提供 List ，支持队列

# Redis

- Redis ：
  - 瑞士军刀！
  - 已经用在 Session 、 Cache
- List 直接支持队列：
  - push
  - brpop/plpop ：阻塞式取数据，避免轮询
- Persistent
- Master/Slave

# Redis List: 太简单

- 底层，使用不方便
  - 错误处理
  - 监控
  - 定时执行
  - 不能查找任务
  - 多个 worker 之间的工作调度

# Redis 之上的队列方案

- RedisMQ ：需要另外一个 server
- Resque ： github 之作， Ruby
- Pyres ：
  - Resque 的 Python Clone
  - 使用复杂，不够 pythonic
- Celery ：
  - 目标太大，潜在维护成本

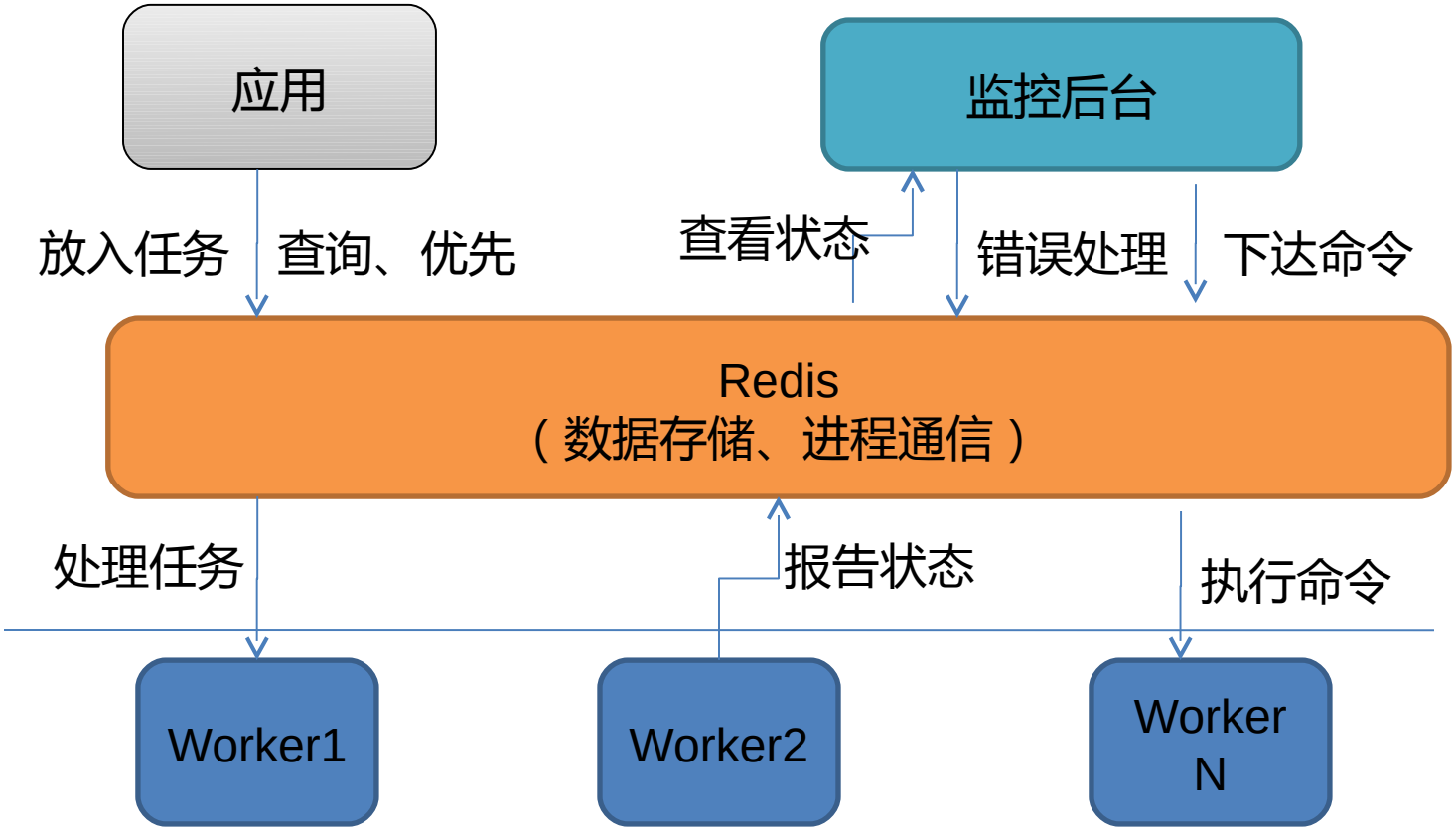
# ZTQ ： Z - Task Queue

- 基于 Redis
- For Python
- 开源
- 来自生产系统
  - 易度云查看
  - 易度云办公
    - 文档转换、索引、日志记录、消息发送、邮件发送、短信发送、垃圾清理、 redis 压缩

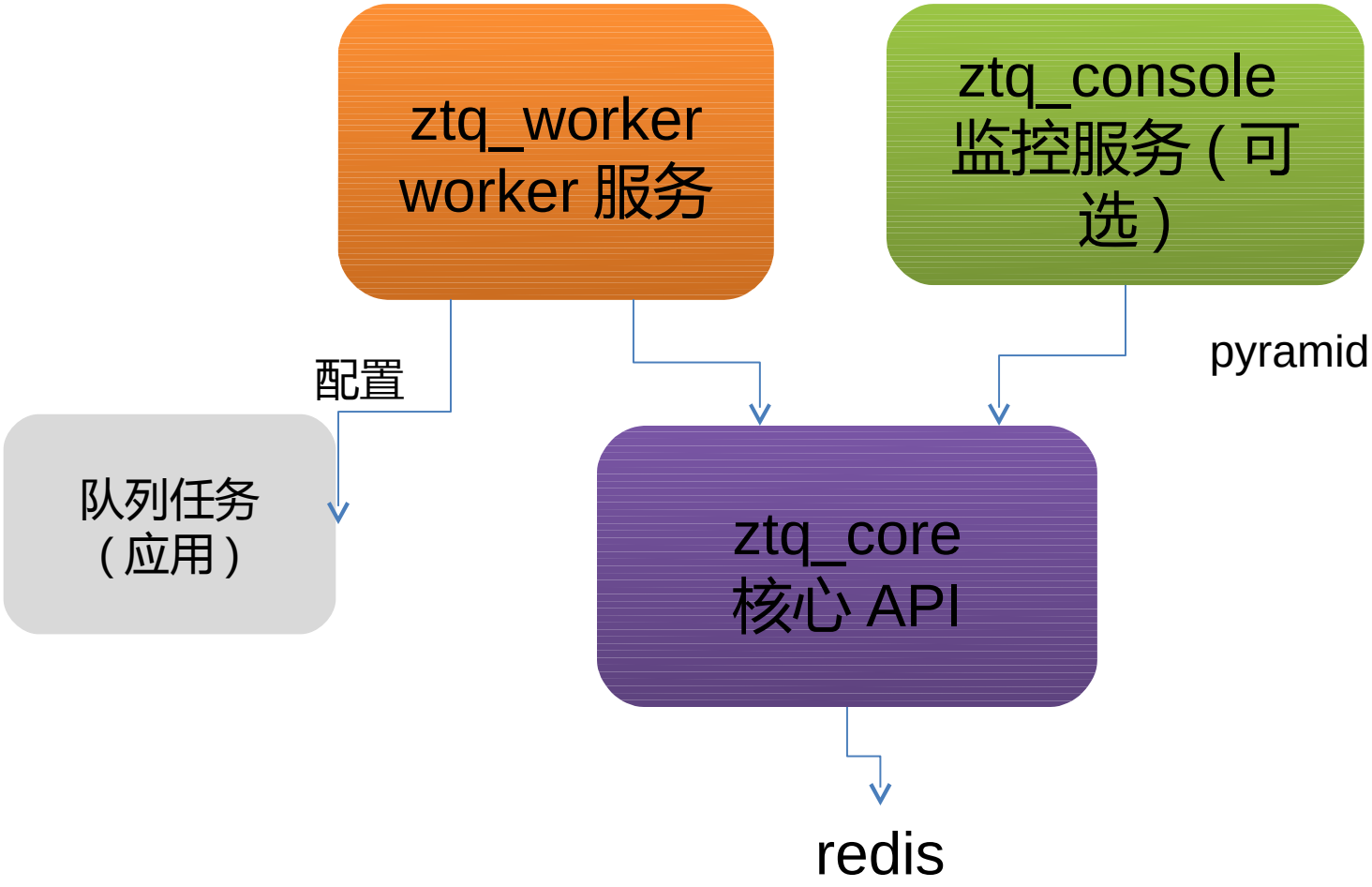
# 设计目标

- 实现简单
- 容易使用
- 可靠
- 可管理：拥塞、出错
- 容易调试
- 灵活调度，高效利用服务器

# 模块关系


应用
监控后台
放入任务
查询、优先
查看状态
错误处理
下达命令
Redis
（数据存储、进程通信）
处理任务
报告状态
执行命令
Worker1
Worker2
Worker
N

# 组成包


ztq_worker
worker 服务
ztq_console
监控服务 ( 可选 )
配置
pyramid
队列任务
( 应用 )
ztq_core
核心 API
redis

# 安装

pip install ztq_core

pip install ztq_worker

# 首先：定义队列任务

```
#  ztq_demo/tasks.py

import time

from ztq_core import async


@async                                                    #
使用默认队列 default

def send(body):

        print ‘START: ‘, body
```

# 接下来：运行 worker

```
# 运行： bin/ztq_worker worker.ini


[server]
host = localhost
port = 6379
db = 0
alias = w01
active_config = false
```

# 最后：测试异步运行

```
import ztq_core
from ztq_demo.tasks import send


# 设置 Redis 连接
ztq_core.setup_redis('default', 'localhost', 6379, 0)
```

send('hello, world')

好，喘口气

小窥下监控后台

# 安装运行

- pip install ztq_console
- bin/pserve app.ini

# 当前 worker 状态


ZTQ队列监控后台
Powered by Everydo.com
工作状态
工作队列
错误清单
工作历史
系统日志
下面是各个工作端的当前工作情况:
主机名称
IP
CPU 占用
内存占用
启动时间
上报时间
状态
操作
当前工作
w01
w01
9.9%
共 1001M
已用 72.3%
2012-10-19 16:06:09
2012-10-19 16:06:10
运转
刷新
停止
在进程 Thread-4 上: start the job 已经运行了 3
详细信息
线程信息
192.168.1.116
w01
10.4%
共 1001M
已用 74.0%
2012-10-19 13:21:47
2012-10-19 13:22:00
空闲
启用
删除

# 队列情况


ZTQ队列监控后台
Powered by Everydo.com
工作状态
工作队列
错误清单
工作历史
系统日志
转换队列
队列 队列方向 工作端 任务数 任务首个时间 错误数 错误首个时间
openoffice-0 首 192.168.8.9:0 192.168.8.9:0 0 31 2012-10-18 13:26:40
transform-0 首 192.168.8.9:0 0 9 2012-08-06 13:46:31
clock-0 首 192.168.8.9:0 0 6 2012-10-18 04:00:45
frs-0 首 192.168.8.9:0 0 2 2012-08-10 15:54:10
msg-0 首 192.168.8.9:0 0 0
xapian_full-1 首 192.168.8.9:0 0 0

# 错误处理


ZTQ队列监控后台
Powered by Everydo.com
工作状态
工作队列
错误清单
工作历史
系统日志
最近1 ~ 20条错误清单如下：
所属队列
工作端
开始时间
结束时间
出错类型
操作
openoffice-0
192.168.8.9
2012-10-19 16:43:47
2012-10-19 16:43:59
未知
详细信息
重做
删除
Traceback (most recent call last):
File "/opt/hg/cloudviewer/ztq_worker/ztq_worker/job_thread.py", line 125, in start_job
self.run_task(*task['args'], **task['kw'])
File "/opt/hg/workonline/zopen.frs4wsgi/src/zopen/frs4wsgi/wsgi2frs/__init__.py", line 32, in transform_for_index_job
transform_engine.convert(path, source_mime_type, {'vpath': vpath}, 'text/plain')
File "/opt/hg/workonline/zopen.frs4wsgi/src/zopen/frs4wsgi/engine/engine.py", line 328, in convert
raise Exception('\r\n----------\r\n'.join(errors))
Exception: Traceback (most recent call last):
File "/opt/hg/workonline/zopen.frs4wsgi/src/zopen/frs4wsgi/engine/engine.py", line 308, in convert

# 队列执行日志


ZTQ队列监控后台
Powered by Everydo.com
工作状态
工作队列
错误清单
工作历史
系统日志
16:08:42
16:08:52
16:08:57
ob
success
详细信息
成功
2012-10-19 16:08:42
w01
send
2012-10-19 16:08:47
2012-10-19 16:08:52
start the job
success
详细信息
出错
2012-10-19 16:08:47
w01
send2
2012-10-19 16:08:47
2012-10-19 16:08:47
start the job
Traceback (most recent call last):
File "/home/panjy/hg/ztq/lib/python2.7/site-packages/ztq_worker-0.1.1dev-py2.7.egg/ztq_worker/job_thread.py", line 128, in start_job
self.run_task(*task['args'], **task['kw'])
File "build/bdist.linux-i686/egg/ztq_core/demo.py", line 15, in send2
raise Exception('connection error')
Exception: connection error
详细信息
成功
2012-10-19 16:08:47
w01
fail_callback
2012-10-19 16:08:47
2012-10-19 16:08:47
start the job

# Worker 运行日志


ZTQ队列监控后台
Powered by Everydo.com
工作状态
工作队列
错误清单
工作历史
系统日志
最近1 ~ 20条服务器运行如下：
工作端 服务器IP 报告时间 状态
w01 192.168.1.22 2012-10-19 16:06:09 reboot
w01 192.168.1.22 2012-10-19 15:53:20 reboot
w01 192.168.1.22 2012-10-19 15:01:41 reboot
w01 192.168.1.22 2012-10-19 14:57:55 reboot
w01 192.168.1.22 2012-10-19 14:11:28 reboot
w01 192.168.1.22 2012-10-19 14:02:41 reboot
w01 w01 2012-10-19 14:00:07 reboot
w01 w01 2012-10-19 13:30:29 reboot
w01 w01 2012-10-19 13:29:45 reboot

更多特性。。。

# 抢占式执行

```
#  后插入先执行
#  如果任务已经在队列，会优先
send (body,   ztq_first=True)
```

# Ping: 探测任务状态

# running: 运行； queue: 排队中；

# error: 出错； none: 不存在

ztq_core.ping_task(send, body)

# ztq_first 存在就优先 ; ztq_run 不存在就运行

ztq_core.ping_task(send, body,

# 事务： transaction

```
import transaction

ztq_core.enable_transaction(True)

send_mail(from1, to1, body1)
send_mail(from2, to2, body2)
transaction.commit()
```

# Cron ：定时任务

```
from async import async
import redis_wrap
from ztq_core import has_cron, add_cron


@async(queue='clock-0')
def bgrewriteaof():
    """  将 redis 的 AOF 文件压缩    """
    redis = redis_wrap.get_redis()
```

# 任务串行： callback

```
from ztq_core import prepare_task

callback = prepare_task(send, body)

send_mail(body,
                ztq_callback=callback)
```

# 多级串行

```
from ztq_core import prepare_task


callback1 = prepare_task(send, body)


callback2 = prepare_task(send2, body, ztq_c
allback=callback1)
```

send (body,  ztq_callback=callback2)

# 异常处理： fcallback

```
from ztq_core import prepare_task


@async(queue='mail')
def fail_callback(return_code, return_msg):
        print return_code, return_msg


fcallback = prepare_task(send2)
```

# 进度回调： pcallback

```
from ztq_core import prepare_task

pcallback = prepare_task(send2, body)

send_mail(body,  ztq_pcallback=pcallback)
```

# 抛出进度信息

```
import ztq_worker


@async(queue='xxx')
def doc2pdf(filename):
        ...
    # 可被进度回调函数调用
    ztq_worker.report_progress(page=2)
```

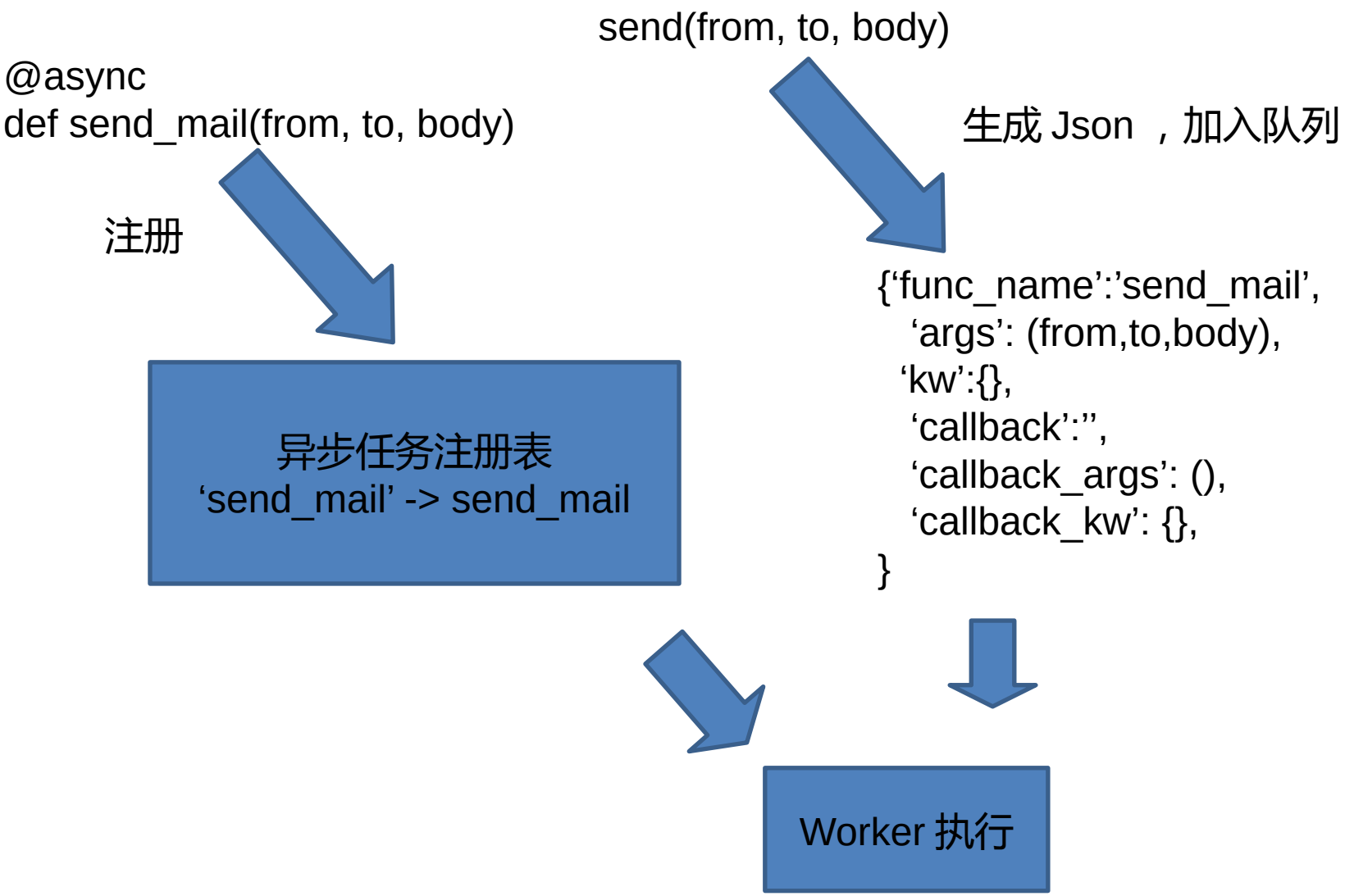
# 拥塞：批处理加速

```
# 为提升性能，需要多个 xapian 索引操作，一次性提交数据库


@async(queue='xapian')
def index(data):
    pass


def do_commit():
```

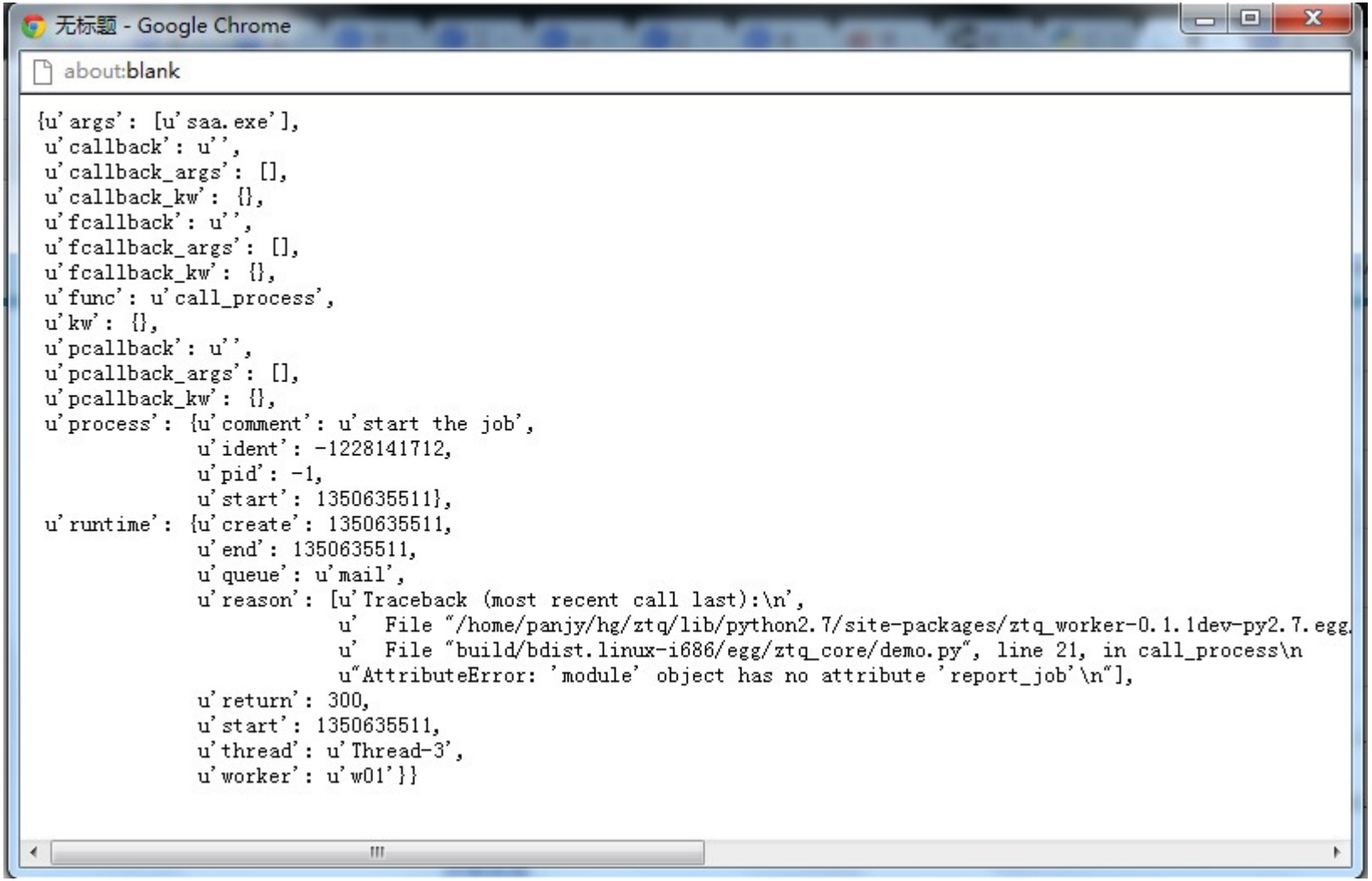
# 内部原理

# 更多血淋淋的细节

# 任务的序列化


send(from, to, body)
@async
def send_mail(from, to, body)
生成 Json ，加入队列
注册
{‘func_name’:’send_mail’,
‘args’: (from,to,body),
‘kw’:{},
‘callback’:’’,
‘callback_args’: (),
‘callback_kw’: {},
}
异步任务注册表
‘send_mail’ -> send_mail
Worker 执行

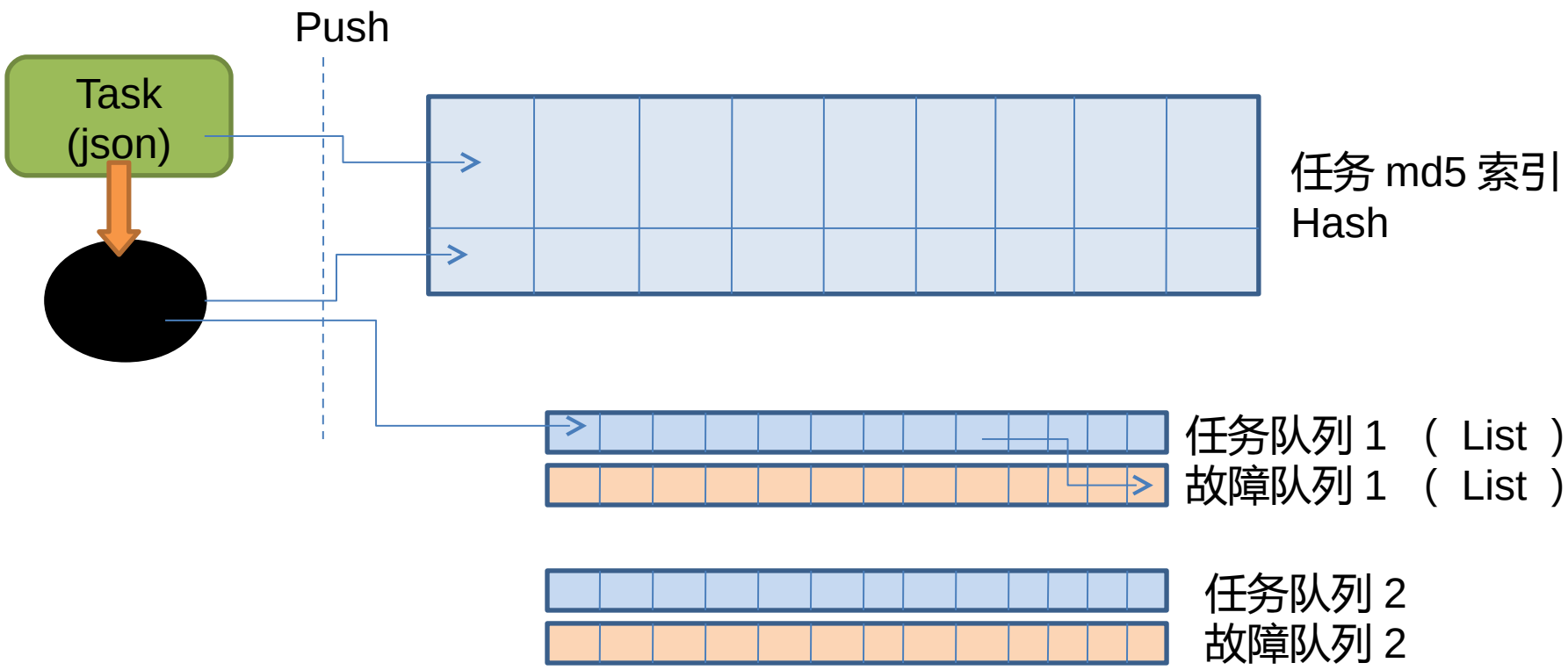
# 完整的任务信息


无标题 - Google Chrome
about:blank
{u'args': [u'saa.exe'],
u'callback': u'',
u'callback_args': [],
u'callback_kw': {},
u'fcallback': u'',
u'fcallback_args': [],
u'fcallback_kw': {},
u'func': u'call_process',
u'kw': {},
u'pcallback': u'',
u'pcallback_args': [],
u'pcallback_kw': {},
u'process': {u'comment': u'start the job',
u'ident': -1228141712,
u'pid': -1,
u'start': 1350635511},
u'runtime': {u'create': 1350635511,
u'end': 1350635511,
u'queue': u'mail',
u'reason': [u'Traceback (most recent call last):\n',
u' File "/home/panjy/hg/ztq/lib/python2.7/site-packages/ztq_worker-0.1.1dev-py2.7.egg.
u' File "build/bdist.linux-i686/egg/ztq_core/demo.py", line 21, in call_process\n
u"AttributeError: 'module' object has no attribute 'report_job'\n"],
u'return': 300,
u'start': 1350635511,
u'thread': u'Thread-3',
u'worker': u'w01'}}

# 任务 ID ？

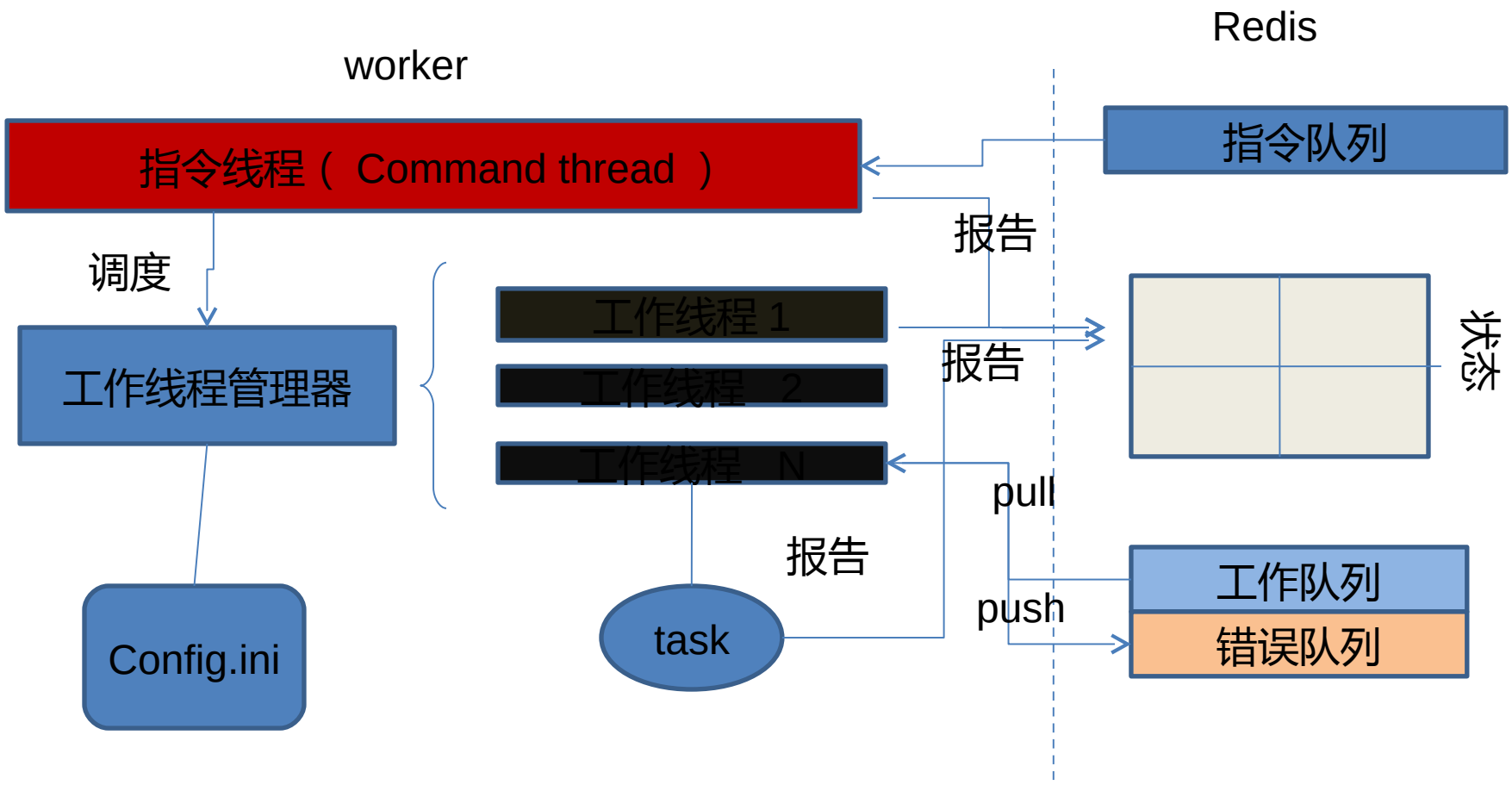
- 直接根据任务 JSON ，生成 MD5 ，作为 ID
  - 方便查询任务是否已经存在
  - 避免出现重复的任务
- 也是问题：不可插入完全相同的任务！
- 附加一个参数，来区分

# 任务队列的 Redis 存储设计




任务队列和错误队列一一对应，方便管理

# Worker 模型




worker 指令线程：工作机状态报告：线程工作调度；杀死 / 取消进程

Task 可报告工作进程的 pid ，监控后台可下指令杀死卡死的进程

# 智能调度脚本

- 管理上百台 worker 服务器？
  - 工作是否饱和程度
  - 自动调整工作安排
- 是可能的！
  - 读取 worker 的 CPU 、内存情况
  - 根据闲忙，调整任务的分配

# TODO

- 延时执行
- 支持协程，用于下载
- 优化监控后台的代码
- 改进 cron
- 需要 TestCase

# 总结

- Redis ：分布式计算的通信中心
- 感谢 PyCONChina ，让我们有时间开源

# 项目信息

- Github ：
  https://github.com/everydo/ztq

- 主要作者
  - 徐陶哲
    http://weibo.com/xutaozhe
  - 潘俊勇
    http://weibo.com/panjunyong