美团云与Python：
使用简单语言构建复杂系统

美团网 云计算开发工程师
陈博
2015-09-19

简介

美团网
meituan.com

- 美团有众多使用Python的产品和项目
  今年第一次来PyCon交流
  最有代表性的

- 美团云： https://mos.meituan.com

- 项目历史：3年
  运维团队孵化 -> 公司内部私有云 -> 公有云产品上线运营

- 本次演讲内容
  不讲云计算，只讲语言和框架；
  介绍为主，配一两个代码分析；
  Pythonic的小故事

简介

美团网 meituan.com

- 为什么用Python?
  OpenStack、团队成员、DevOps
- 自研vs开源项目
- 代码规模
  自研：50W Python
  大量：C / shell
  维护：OpenStack 15W Python
- 功能模块划分
  region controller / host / keystone / glance / swift / api / notify / meter / monitor / vpc / billing / container / baremetal / ssh_relay / vnc / elastic IP / rational database service / cache service / block storage / load balance / big data service / web frontier …
- Python发挥“胶水语言”的特点，将不同架构、不同语言的模块整合到一起

功能模块

美团网 meituan.com

访问控制

WEB

CLI

API

用户管理

目录服务

使用量统计与计费

模板管理

ELB PAAS...

通知报警

资源管理(DB)

任务管理

调度

日志

监控

Network

IP/MAC地址
DNS, 路由与NAT
带宽限制
访问控制

Block Storage

格式转换, 分区,格式化
snapshot, template
migration
IO限制

Hypervisor

KVM
CPU/Memory
使用量限制
migration

Object storage

前端框架

- 前端后台：Django

  目前没有遇到性能问题，因为：

  云计算产品在Web上需要的操作不多；

  部署了多节点分摊压力；

  采用了相对高效的模板库（jinja）进行渲染

- 代码管理与安全隔离：

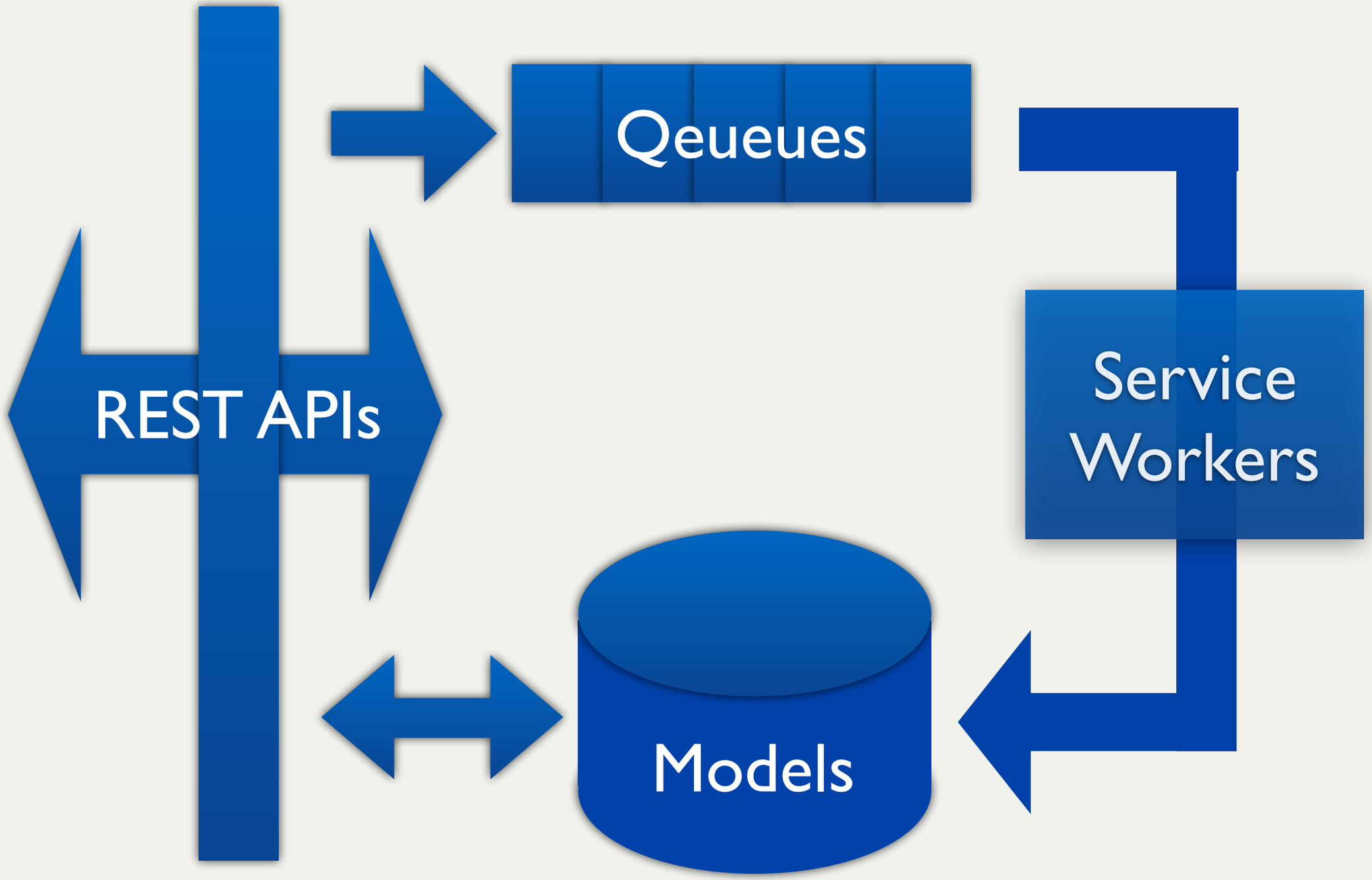
  代码独立管理：不是一套风格

  部署与后端分离，通过API调用，防止Web攻击

- pros & cons：

  社区支持较好；有些特性比较鸡肋

后端Work Flow



消息通信

美团网 meituan.com

- 各个服务之间的通信
  稳定性第一
- 消息队列 vs Web server
  OpenStack采用Rabbitmq；
  Web server更加久经考验，业界对于http协议有着最成熟的经验
- 重度使用了Tornado
  单线程、异步回调、协程、高性能、语言级支持

数据库

美团网 meituan.com

- 使用ORM：SQLAlchemy

  功能丰富；文档不友好，有些小坑；唯一能用的几个Python ORM中算比较好的；关闭auto-commit

- 数据库查询：缺少对异步的原生支持

  Web server的异步框架与ORM的同步查询矛盾；
  网上能找到开源的数据库异步驱动（不敢用）

- 定制Tornado，支持线程

  由于GIL的存在，协程在Python中的地位非常重要，但复杂项目仍不可缺少线程；

- 线程级别的数据库事务锁、对象锁

源码剖析：Tornado

美团网 meituan.com

```
3  def finish(self, chunk=None):
4      """Finishes this response, ending the HTTP request."""
5
6      if not self._headers_written:
7          if (self._status_code == 200 and
8              self.request.method in ("GET", "HEAD") and
9                  "Etag" not in self._headers):
10             self.set_etag_header()
11             if self.check_etag_header():
12                 self._write_buffer = []
13                 self.set_status(304)
14         if self._status_code == 304:
15             assert not self._write_buffer, "Cannot send body with 304"
16             self._clear_headers_for_304()
17         elif "Content-Length" not in self._headers:
18             content_length = sum(len(part) for part in self._write_buffer)
19             self.set_header("Content-Length", content_length)
20
21     # Sets a callback that will be run when the connection is closed
22     self.request.connection.set_close_callback(None)
23
24     # Flushes the current output buffer to the network
25     self.flush(include_footers=True)
26
27     # Finishes the request. Close the socket connection
28     self.request.finish()
29
30     self._finished = True
31     self.on_finish()
32
33
34 def on_finish(self):
35     """Called after the end of a request.
36
37     Override this method to perform cleanup, logging, etc.
38     It is called after the response has been sent to the client.
39     """
40     pass
```

源码剖析：美团云

美团网 meituan.com

```
10
11  class BaseDBSyncHandler(BaseHandler):
12      # BaseHandler -> tornado.web.RequestHandler
13
14      @tornado.web.asynchronous
15      def process(self):
16          sync_req = self._sync_request_class_(self)
17          # A typical task queue model, insert itself into queue
18          get_sync_request_manager().request(sync_req)
19
20      def on_finish(self):
21          clean_db_session()
22          super(BaseDBSyncHandler, self).on_finish()  # Tornado's on_finish
23
24
25  class Worker(BaseWorker):
26      # BaseWorker -> threading.Thread
27      def worker_finish(self):
28          IOLoop.instance().add_callback(self.request.finish)
29
30
31  class SyncDBRequest(SyncRequest):
32
33      def finish(self):
34          self.output_header()
35          self.__handler.write(self.resp)
36          self.__handler.finish()
```

延伸阅读

美团网 meituan.com

- Python 3 asyncio

  This module provides infrastructure for writing single-threaded concurrent code using coroutines, multiplexing I/O access over sockets and other resources, running network clients and servers, and other related primitives.

  asyncio的核心是一个event loop，功能上包含了coroutine、future、task的抽象；性能优秀（superior to nodejs）

  十分火爆，众多第三方库添加支持；asyncio.org，试图重写一切

- database + asyncio ?

  泼冷水，SQLAlchemy作者：使用关系型数据就应该是线程模型；性能下降
  http://techspot.zzzeek.org/2015/02/15/asynchronous-python-and-databases/

  表支持，Python社区人士：除了关系型数据库以外还有Redis，Memcache，RPC这些东西；一个操作里完成所有查询并自动收集结果是个很cool的事
  http://www.onebigfluke.com/2015/02/asyncio-is-for-composition.html

REST API

美团网 meituan.com

- REST API

  4种http操作对应资源的增删查改

  GET /class_name/object_id

- 资源的从属关系在云计算业务上能够非常清楚地反映

  云也是一种资源，特别是对于IAAS

  DELETE /zones/zone_id/hosts/host_id

  POST /storages/storage_id/disks

- http client的一些定制

  httplib2 -> Tornado async client

  future

其他

- Python包管理

  测试环境：单机版云平台，一键安装；
  仍然使用pip：离线源码包，指定版本；
  依赖链：pip show查询，提前在脚本里写好顺序

- subprocess.check_output

  收集程序输出结果，经常使用；
  2.6版本只有check_call和call；
  自己实现

其他

- Python 2.6，踩过的坑

  CentOS 6.x 默认自带2.6版本；
  Dict Comprehension 语法只在2.7以上支持；
  非异常，编译器直接报错，导致服务不可用

  d = {key: value for (key, value) in iterable}

争议问题

美团网 meituan.com

- 性能

  很慢，非IO，纯内存操作；

  不敢开线程；

  多实例，分布式锁，有开发难度；

  对比go coroutine、java thread，go map vs python dict

- 内存泄露?

  自研的模块基本没遇到过，用apache+mod_wsgi跑Keystone服务的时候遇到过几次

- 动态语言特性

  提升开发效率的同时也带来危险性，保持良好编码习惯；

  typo / assert

- IDE vs Text Editor

美团网

meituan.com

谢谢！