异步机制赋于 Web 开发的新手段

邹业盛 / yeshengzou@gmail.com / 2012-04-01

目录

什么是异步机制

同步的情况（没有结果不会离开）

►► 顺序，即时结果，等待，阻塞，浪费时间

异步的情况（安排好一切）

►► 并行，非即时，不等待，非阻塞，挤时间 ►
► 并行，非即时，不等待，非阻塞， 挤时间

异步的两种方式：

1. 事件驱动
2. 上下文调度

异步机制作用于 Web 服务器和 Web 开发框架……

在单个上下文环境（进程）中处理大量并发请求的能力

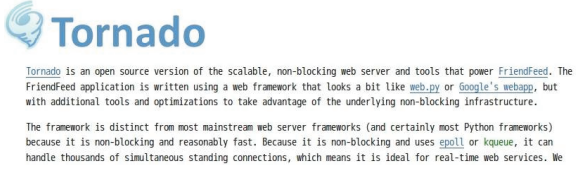
►► 可能性……

使用工具简介

Python

http://www.python.org

Tornado

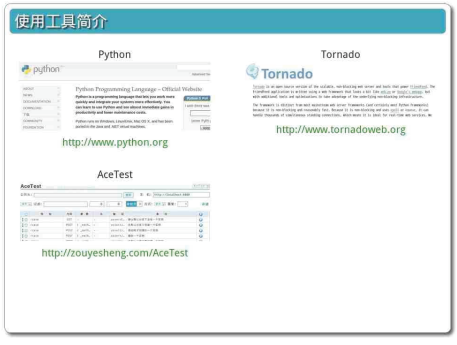


http://www.tornadoweb.org

AceTest

http://zouyesheng.com/AceTest.html

AceSlide



http://zouyesheng.com/AceSlide.html

“全局变量”

多个请求是否可以操作同一个变量……

```
GlobalVar = '' //所谓的全局变量
class GlobalVarHandler(BaseHandler):

    def get(self):
        global GlobalVar
        return self.write(u'现在 v 的值是: ' + GlobalVar)

    def post(self):
        global GlobalVar
        v = self.get_argument('v', '')
        GlobalVar += v
        return self.write('0')

    def delete(self):
        global GlobalVar
        GlobalVar = ''
        return self.write('0')
```

再考虑一个场景：防暴力请求

如果一个 IP 在周期内请求次数过多，则列入黑名单，也可以外部添加黑名单。

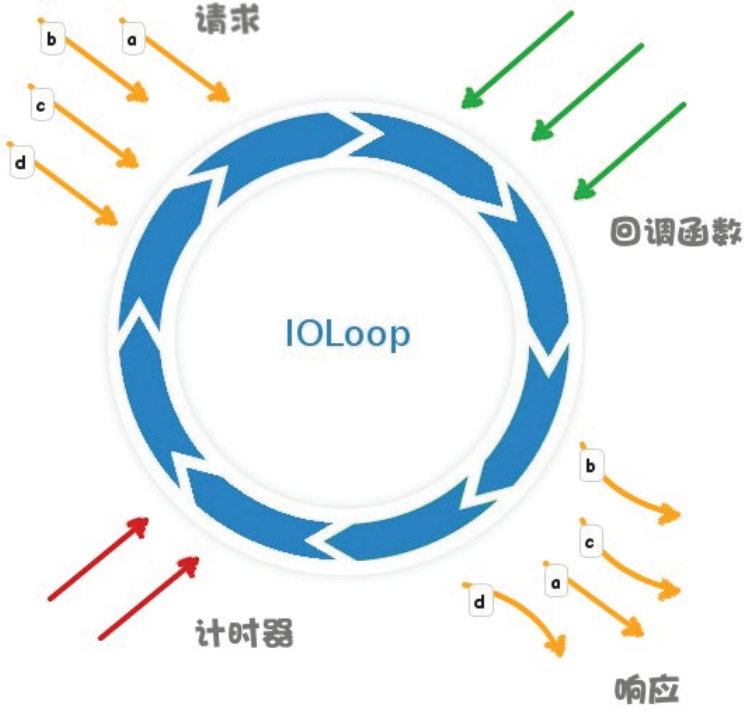
```
class ViolenceHandler(BaseHandler):
    '暴力请求'

    count = {}
    last = 0 //时间间隔

    def get(self):
        ip = self.get_argument('ip', '')
        if not ip:
            return self.write({'result': 1, 'msg': 'need a ip'})

        cls = self.__class__
        now = int(time.time())
        if now - cls.last > 60: //60秒之内
            cls.count = {}
            cls.last = now
        if ip in cls.count:
            if cls.count[ip] > 10: //10次超过不能
                return self.write({'result': 2, 'msg': ''})
            else:
                cls.count[ip] += 1
        else:
            cls.count[ip] = 1
        return self.write({'result': 0, 'msg': cls.count[ip]})
```

事件循环机制



计时器

- cron
- javascript
- 化同步为异步，变阻塞为非阻塞

两部分内容：周期循环　延时执行

代替 cron 的好处：

- 减少额外手段，利于维护
- 与运行时相同的上下文环境，交互方便

以抓取网页数据（代理 IP 列表）为例 http://www.sooip.cn/e/web/?type=rss2&classid=1 ：

- 抓取完分页的数据，每一页请求之间间隔 5 秒。
- 定时检查是否有数据更新。
- 对抓取的数据进行可行性检查。
- 检查通过的 IP 资源以服务形式对外供给数据。

```
IL = tornado.ioloop.IOLoop.instance()
RSS_URL = 'http://www.sooip.cn/e/web/?type=rss2&classid=1'
LINK_URL = re.compile('([^<]*?)')
IP_RE = re.compile('(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3} \d{2,5}) HTTP')
IP = []
ALL_URL = []

def fetch_rss(url, callback): //获取RSS页
    AsyncHTTPClient().fetch(url, callback=callback)

def on_rss_fetch(response): //处理RSS页
    global ALL_URL
    ALL_URL = LINK_URL.findall(response.body)
    IL.add_callback(functools.partial(fetch_page,
                                      ALL_URL.pop(), on_page_fetch))

def fetch_page(url, callback): //获取某一内容页
    AsyncHTTPClient().fetch(url, callback=callback)

def on_page_fetch(response): //处理内容页
    global ALL_URL
    global IP
    ip_list = IP_RE.findall(response.body)
    IP.extend(ip_list)
    if ALL_URL:
        IL.add_timeout(datetime.timedelta(seconds=5),
                       functools.partial(fetch_page,
                                         ALL_URL.pop(), on_page_fetch))
    else:
        global checker
        checker.start() //checker是一个定时器
```

缓存

- memcached
- 缓存的维护问题
- 让缓存自己活动起来

一个整站用户在线状态的例子：

- 访问一个链接，传递用户名之后就算是“登陆”。
- 登陆时，服务器端记录一个“会话”。（ 会话作为缓存数据处理 ）
- 可以获取当前登陆的用户。（ 缓存数据直接主动服务于应用 ）
- 1 分钟没有 ping 则自动注销。（ 缓存的维护 ）

```python
class LoginHandler(BaseHandler):
    '处理登陆'

    def get(self):
        name = self.get_argument('name', '')
        if not name:
            return self.write({'result': 1, 'msg': 'need a name'})

        if name in SessionHandler.session:
            IL.remove_timeout(SessionHandler.session[name])
        print 'set session %s ...' % name
        SessionHandler.session[name] = IL.add_timeout(datetime.timedelta(seconds=10),
                      functools.partial(SessionHandler.remove_session, name))
        return self.write({'result': 0, 'msg': ''})


class SessionHandler(BaseHandler):
    '处理会话'

    session = {}

    @classmethod
    def remove_session(cls, name):
        print 'remove session %s ...' % name
        del cls.session[name]

    def get(self):
        return self.write({'result': 0, 'msg': '',
                           'session': self.__class__.session.keys()})
```

代理层

- 所有的请求会在服务器端汇集，并可以延迟返回。
- 针对时间范围内的整体的组合优化。

一个与数据库查询有关的例子：传递一个 ID ，获取数据库中对应的数据。

```
class ProxyHandler(BaseHandler):
    request = {}
    @tornado.web.asynchronous
    def get(self):
        id = self.get_argument('id', '')
        id = int(id)
        cls = self.__class__
        if not id: return self.finish({'result': 1, 'msg': 'need a id'})
        else:
            if id in cls.request:
                cls.request[id].append(self)
            else:
                cls.request[id] = [self]

class SQLTimer(tornado.ioloop.PeriodicCallback):
    def __init__(self, callback_time):
        super(SQLTimer, self).__init__(self.sql, callback_time)

    def sql(self):
        request = ProxyHandler.request.copy()
        ProxyHandler.request = {}
        ids = request.keys()
        sql = 'select id, name from Main where id in (%s)' % (', '.join(['?'] * len(ids)))
        c = CONN.cursor()
        c.execute(sql, ids)
        r = c.fetchall()
        c.close()

        for id, name in r:
            for req in request[id]:
                req.finish({'result': 0, 'name': name})
```

注意的问题

- 阻塞的风险，对编写者的要求。
- 异部风格的代码，维护与调试问题。
- 为利用多核，多实例之间的数据共享与同步问题。

结束

Q & A

邹业盛 / yeshengzou@gmail.com / 2012-04-01