Upgrade mpcgi 2.9 to 3.0

python coroutine

11 October 2016

邝昌浪
网易游戏计费组

Outlines

- Python web
- Coroutine
- Gevent
- Gunicorn
- Upgrade mpcgi 2.9 to 3.0

Python web--deployments

- cgi

```
fork-and-execute
```

- fastcgi, scgi

```
long-live process
```

- mod_python

```
embed python interpreter into web server(apache only)
```

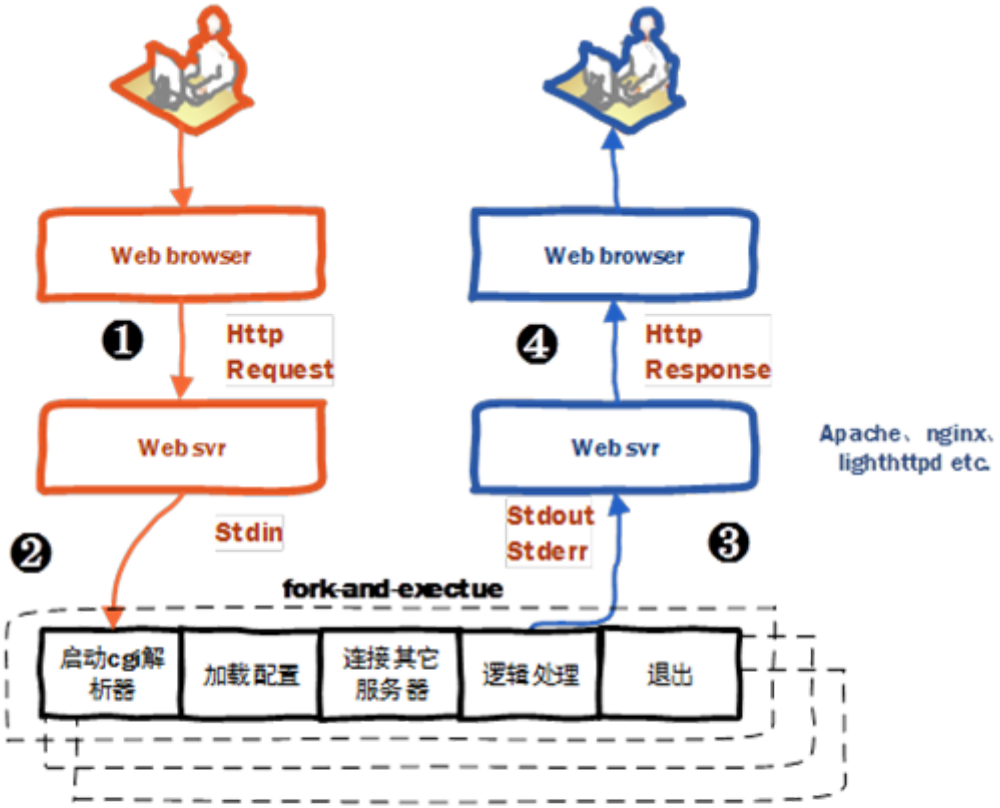
- uwsgi

```
uWSGI server specified
```

- wsgi

```
limited to python
```

Python web--cgi

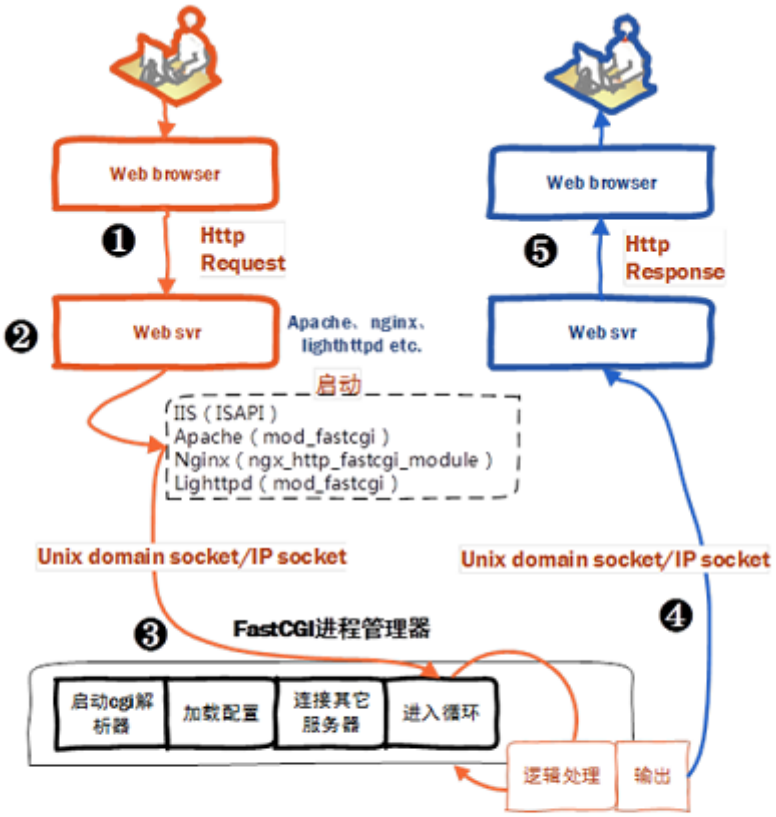


说明：在遇到连接请求(用户请求)

1. 先要创建cgi的子进程，然后cgi子进程处理请求，处理完后结束这个子进程。这就是fork-and-execute模式。

2. cgi方式的服务器有多少连接请求就会有多少cgi子进程，每个子进程都需要启动CGI解释器、加载配置、连接其它服务器等初始化工作，这是cgi性能低下的主要原因。当用户请求数量非常多时，会大量挤占系统的资源如内存，CPU时间等，造成效能低下

Python web--fastcgi



说明：在收到连接请求(用户请求)

1. web svr的fastcgi执行环境，通过socket（域socket或ipsocket），将数据传递给fastcgi程序进程。
2. fastcgi程序进程收到请求后，进行对应的逻辑处理
3. 最后将处理结果通过socket（域socket或ipsocket），返回给web svr
4. web svr构建Http Response响应包，返回给浏览器

Python web--fastcgi advantages

- get rid of fork overhead
- binary protocol
- deployed on any separated machines from web server(compared to cgi)

Python web--fastcgi server

- flup

```
python web server support fastcgi
```

```
#!/usr/bin/python
# encoding : utf-8

import os
from flup.server.fcgi import WSGIServer

count = 0
def myapp(environ, start_response):
    global count
    start_response('200 OK', [('Content-Type', 'text/plain')])
    count += 1
    return ['Hello World fastcgi!\nAccess count %d\n Running pid: %d' % (count, os.getpid())]

if __name__  == '__main__':
   WSGIServer(myapp, bindAddress=('127.0.0.1',8080)).run()
```

- spawn-fcgi

```
process manager
```

Python web--mod_python

- embed python interpreter into web server
- processes managed by server

www.onlamp.com/pub/a/python/2003/10/02/mod_python.html

(http://www.onlamp.com/pub/a/python/2003/10/02/mod_python.html)

Python web--uwsgi

- high performance

uwsgi-docs.readthedocs.io/en/latest/ (https://uwsgi-docs.readthedocs.io/en/latest/)

Python web--wsgi

- pep 333 (pep 3333 on python 3.x)
- wsgi is an attempt to get rid of the low level gateways
- simple and universal interface between web servers and web applications or frameworks

```
# coding=utf-8

def application(environ, start_response):
    status = '200 OK'
    response_headers = [('Content-Type', 'text/plain')]
    start_response(status, response_headers)
    return ['Hello world']

# environ:          一个包含请求信息及环境信息的字典，server 端会详细说明
# start_response:   一个接受两个参数`status, response_headers`的方法:
# status:           返回状态码，如http 200、404等
# response_headers: 返回信息头部列表
```

Python web--wsgi

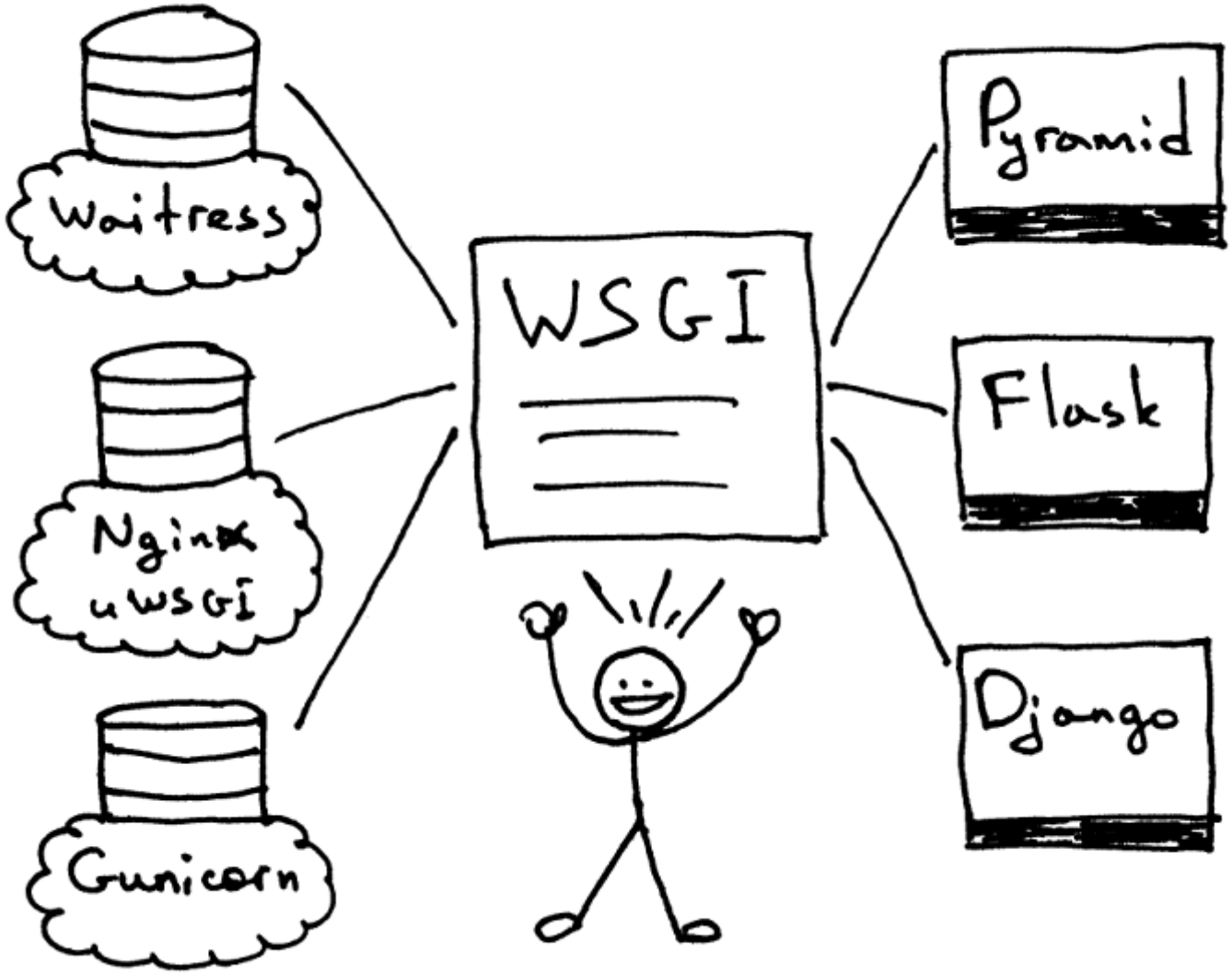
- server

```
- handle connections
- process(thread) management
- ...

* gunicorn
* mod_wsgi
* flup
* ...
```

- framework(app)

```
provides simple api for programmer, such as:
- url mapping
- template management
- session management
- ...

* webpy
* flask
* django
* ...
```

Python web--wsgi



Python web--wsgi benefits

- easy to code
- easy to migrate application from one server to another
- easy to add middleware
- ...

Python web--summary

- Process

```
can not make full use of multiple CPUs
```

- Multi-threads

```
GIL
```

- Multi-processes or Multi-threads

```
CPU and memory overhead
```

- Processes pool or threads pool

```
better one
```

Other choices?

```
coroutine
```

Coroutine--What is coroutine

Coroutines are computer program components that generalize subroutines for nonpreemptive multitasking, by allowing multiple entry points for suspending and resuming execution at certain locations.

- user-defined threads
- scheduled by user

Programming languages with native support:

```
- go
- lua
- ruby
- javascript(since 1.7)
- python 3.5(explicit support)
- ...
```

Coroutine--coroutines vs threads

- Threads:

```
- 8k stack memory on create(default)
- more context switch time(kernel space)
- lock on global access
+ scheduled on multi-cpus
```

- Coroutines:

```
+ lower memory cost
+ less context switch time(user space)
+ lock free
- single cpu only
```

Coroutine--context switch

- Consider:

```
1. two jobs running on a single cpu
2. each takes 10 seconds cpu calculation.
3. cpu context switch each second, each takes 0.1 seconds
4. coroutine context switch takes 0.1 seconds(even less)
```

- threads:

```
takes 10 + 0.1*20 + 10 = 22 seconds
```

- coroutines:

```
takes 10 + 0.1 + 10 = 20.1 seconds
```

Coroutine--Python coroutine

- python 2.5+

```
- keyword yield
- greenlet
```

```
def fib():
    first, second = 0, 1
    yield first

    while True:
        yield first + second
        first, second = second, first+second

if __name__ == '__main__':
    g = fib()
    for i in xrange(50):
        print g.next()
```

- python 3.5+

```
explicit support
```

Coroutine--greenlet

The greenlet package is a spin-off of Stackless, a version of CPython that supports micro-threads called "tasklets".

- Micro-threads with no implicit scheduling
- Implemented in C(stack switch implemented in ASM)
- Lightweight
- Only one can run at a time
- Cooperative

Coroutine--greenlet example

- Organized in a tree structure

```
- every greenlet has a "parent" greenlet, except main
- when a greenlet dies, control is switched to its parent
```

```
from greenlet import greenlet

def test1():
    print 12
    gr2.switch()
    print 34

def test2():
    print 56
    gr1.switch()
    print 78

gr1 = greenlet(test1)
gr2 = greenlet(test2)
gr1.switch()
```

Coroutine--greenlet stack switch

```
/***********************************************************
A PyGreenlet is a range of C stack addresses that must be
saved and restored in such a way that the full range of the
stack contains valid data when we switch to it.

Stack layout for a greenlet:

               |     ^^^       |
               |  older data   |
               |               |
  stack_stop . |_______________|
        .      |               |
        .      | greenlet data |
        .      |   in stack    |
        .    * |_______________| . .  _____________  stack_copy + stack_saved
        .      |               |     |             |
        .      |     data      |     |greenlet data|
        .      |   unrelated   |     |    saved    |
        .      |      to       |     |   in heap   |
 stack_start . |     this      | . . |_____________| stack_copy
               |   greenlet    |
               |               |
               |  newer data   |
               |     vvv       |
***********************************************************/
```

Gevent

gevent is a coroutine -based Python networking library that uses greenlet to provide a high-level synchronous API on top of the libev event loop.

- Fast event loop based on libev (epoll on Linux, kqueue on FreeBSD).
- Lightweight execution units based on greenlet.

Gevent--example

```
import gevent

def foo():
    print('Running in foo')
    gevent.sleep(0)
    print('Explicit context switch to foo again')

def bar():
    print('Explicit context to bar')
    gevent.sleep(0)
    print('Implicit context switch back to bar')

gevent.joinall([
    gevent.spawn(foo),
    gevent.spawn(bar),
])
```

Gevent--monkey patch

- Gevent switch greenlets implicitly when:

```
- io
- sleep
- signal
```

```
def patch_all(socket=True, dns=True, time=True, select=True, thread=True, os=True,
              ssl=True, httplib=False, subprocess=True, sys=False,
              aggressive=True, Event=False, builtins=True, signal=True)
"""
       Do all of the default monkey patching
       (calls every other applicable function in this module).
"""
```

Gunicorn

- Gunicorn 'Green Unicorn' is a Python WSGI HTTP Server for UNIX.
- pre-fork worker model
- compatible with various web frameworks
- simply implemented, light on server resources, and fairly speedy

Upgrade mpcgi 2.9 to 3.0--architecture of 2.9

- nginx
- flup(fastcgi) + webpy

```
upstream h99mp {
        server 127.0.0.0:19660 weight=1 fail_timeout=10 max_fails=1;
        server 127.0.0.0:19661 weight=1 fail_timeout=10 max_fails=1;
        server 127.0.0.0:19662 weight=1 fail_timeout=10 max_fails=1;
        keepalive 16;
}

server {
        ; ...
        location ~ ^/(mp|mpapi)/  {
            fastcgi_pass    h99mp;
            #fastcgi_pass unix:/tmp/nginx_fpy.sock;
            fastcgi_index  index.py;
            #fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_name;
            include        fastcgi_params;
        }
}
```

Upgrade mpcgi 2.9 to 3.0--new to 3.0

- nginx
- gunicorn(master+workers)
- gevent + webpy

```
upstream g0mp {
        server 127.0.0.0:19660 weight=1 fail_timeout=10 max_fails=1;
        keepalive 16;
}

server{
        ;...
        location ~ ^/(admin|mp|mpapi)/ {
                keepalive_timeout  80;
                proxy_read_timeout 80;
                proxy_set_header X-Real-IP       $remote_addr;

                proxy_pass   http://g0mp;
        }
}
```

Upgrade mpcgi 2.9 to 3.0--codes changed

- Do not use any blocking packages, such as Mysqldb

```
replace it with pymysql
```

- Mysqldb

```
- implemented in C
- follow pep 249(Python Database API Specification v2.0)
```

- pymysql

```
- implemented in python
- follow pep 249(Python Database API Specification v2.0)
```

gitlab.game.netease.com/billing_dev/mp_interface/merge_requests/2/diffs?view=parallel

(https://gitlab.game.netease.com/billing_dev/mp_interface/merge_requests/2/diffs?view=parallel)

Upgrade mpcgi 2.9 to 3.0--problem raise

- logging same file in different processes

```
- use socket logging handler
- use file lock on write
```

Upgrade mpcgi 2.9 to 3.0--performance

- flexible deployment via config(sync, gevent)
- better process management
- graceful reload
- lower system load

More--to be continue...

- Python 3

```
-async
-await
```

Thank you

邝昌浪
网易游戏计费组
kcln1687@corp.netease.com (mailto:kcln1687@corp.netease.com)