gunicorn 工作原理

原

江浸月 发布于 2013/01/31 17:35 字数 1339 阅读 5305 收藏 11 点赞 2 评论 1

python gunicorn flask web

##gunicorn工作原理

Gunicorn“绿色独角兽”是一个被广泛使用的高性能的Python WSGI UNIX HTTP服务器，移植自Ruby的独角兽（Unicorn ）项目,使用pre-fork worker模式，具有使用非常简单，轻量级的资源消耗，以及高性能等特点。

Gunicorn 服务器作为wsgi app的容器，能够与各种Web框架兼容（flask，django等）,得益于gevent等技术，使用Gunicorn能够在基本不改变wsgi app代码的前提下，大幅度提高wsgi app的性能。

###总体结构

gunicorn pre-fork worker模型中有一个管理进程以及几个的工作进程。管理进程:master，工作进程:worker。（以下代码中为了方面理解，均去除了一些干扰代码）

master通过pre-fork的方式创建多个worker:

```
def spawn_worker(self):
    self.worker_age += 1
    #创建worker。请注意这里的app 对象并不是真正的wsgi app对象，而是gunicorn的app
    #对象。gunicorn的app对象负责import我们自己写的wsgi app对象。
    worker = self.worker_class(self.worker_age, self.pid, self.LISTENERS,
                               self.app, self.timeout / 2.0,
                               self.cfg, self.log)
    pid = os.fork()
    if pid != 0:  #父进程，返回后继续创建其他worker，没worker后进入到自己的消息循环
        self.WORKERS[pid] = worker
        return pid
```

```
    # Process Child
    worker_pid = os.getpid()
    try:
        ..........
        worker.init_process() #子进程，初始化woker，进入worker的消息循环，
        sys.exit(0)
    except SystemExit:
        raise
    ............
```

在worker.init_process()函数中，worker中gunicorn的app对象会去import 我们的wsgi app。也就是说，每个woker子进程都会单独去实例化我们的wsgi app对象。每个worker中的swgi app对象是相互独立、互不干扰的。

manager维护数量固定的worker：

```
def manage_workers(self):
        if len(self.WORKERS.keys()) < self.num_workers:
            self.spawn_workers()
        while len(workers) > self.num_workers:
            (pid, _) = workers.pop(0)
            self.kill_worker(pid, signal.SIGQUIT)
```

创建完所有的worker后，worker和master各自进入自己的消息循环。 master的事件循环就是收收信号，管理管理worker进程，而worker进程的事件循环就是监听网络事件并处理（如新建连接，断开连接，处理请求发送响应等等），所以真正的连接最终是连到了worker进程上的。（注：有关这种多进程模型的详细介绍，可以参考 http://blog.csdn.net/largetalk/article/details/7939080）

###worker

woker有很多种，包括：ggevent、geventlet、gtornado等等。这里主要分析ggevent。

每个ggevent worker启动的时候会启动多个server对象：worker首先为每个listener创建一个server对象（注：为什么是一组listener,因为gunicorn可以绑定一组地址,每个地址对于一个listener），每个server对象都有运行在一个单独的gevent pool对象中。真正等待链接和处理链接的操作是在server对象中进行的。

```
    #为每个listener创建server对象。
    for s in self.sockets:
        pool = Pool(self.worker_connections) #创建gevent pool
        if self.server_class is not None:
           #创建server对象
            server = self.server_class(
                s, application=self.wsgi, spawn=pool, log=self.log,
                handler_class=self.wsgi_handler, **ssl_args)
        ............
```

```
            server.start() #启动server，开始等待链接，服务链接
            servers.append(server)
            .........
```

上面代码中的server_class实际上是一个gevent的WSGI SERVER的子类：

```
class PyWSGIServer(pywsgi.WSGIServer):
    base_env = BASE_WSGI_ENV
```

需要注意的是构造PyWSGIServer的参数：

```
 self.server_class(
                s, application=self.wsgi, spawn=pool, log=self.log,
                handler_class=self.wsgi_handler, **ssl_args)
```

这些参数中s是server用来监听链接的套接字。spawn是gevent的协程池。application即是我们的wsgi app（通俗点讲就是你用 flask 或者 django写成的app），我们的app就是通过这种方式交给gunicorn的woker去跑的。 handler_class是gevent的pywsgi.WSGIHandler子类。

当所有server对象创建完毕后，worker需要定时通知manager，否则会被认为是挂掉了。

```
    while self.alive:
            self.notify()
            .......
```

这个地方的notify机制设计的比较有趣，每个worker有个与之对应的tmp file，每次notify的时候去操作一下这个tmp file（比如通过os.fchmod），这个tmp file的last update的时间戳就会更新。而manager则通过检查每个worker对应的temp file的last update的时间戳，来判断这个进程是否是挂掉的。

###WSGI SERVER

真正等待链接和处理链接的操作是在gevent的WSGIServer 和 WSGIHandler中进行的。 最后再来看一下gevent的WSGIServer 和 WSGIHandler的主要实现:

WSGIServer 的start函数里面调用start_accepting来处理到来的链接。在start_accepting里面得到接收到的套接字后调用do_handle来处理套接字：

```
def do_handle(self, *args):
    spawn = self._spawn
    spawn(self._handle, *args)
```

可以看出，WSGIServer 实际上是创建一个协程去处理该套接字，也就是说在WSGIServer 中，一个协程单独负责一个HTTP链接。协程中运行的self._handle函数实际上是调用了WSGIHandler的

handle函数来不断处理http 请求：

```
def handle(self):
    try:
        while self.socket is not None:
            result = self.handle_one_request()#处理HTTP请求
            if result is None:
                break
            if result is True:
                continue
            self.status, response_body = result
            self.socket.sendall(response_body)#发送回应报文
          ..............
```

在handle函数的循环内部，handle_one_request函数首先读取HTTP 请求，初始化WSGI环境，然后最终调用run_application函数来处理请求：

```
def run_application(self):
    self.result = self.application(self.environ, self.start_response)
    self.process_result()
```

在这个地方才真正的调用了我们的 app。

总结：gunicorn 会启动一组 worker进程，所有worker进程公用一组listener，在每个worker中为每个listener建立一个wsgi server。每当有HTTP链接到来时，wsgi server创建一个协程来处理该链接，协程处理该链接的时候，先初始化WSGI环境，然后调用用户提供的app对象去处理HTTP请求。

江浸月

粉丝 39 博文 9 码字总数 18652 作品 2

深圳 程序员