

技术栈

匠人精神，持之以恒！

博客园 首页 新随笔 联系 订阅 管理

随笔 - 364 文章 - 0 评论 - 60 阅读 - 93万

昵称： 大数据老司机
园龄： 5年8个月
粉丝： 243
关注： 1
+加关注

<2024年10月>

日	一	二	三	四	五	六
29	30	1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31	1	2
3	4	5	6	7	8	9

搜索

找找看

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

我的标签

大数据(29)

Hadoop(27)

python(20)

python-pyqt5(9)

Flink(7)

spark(5)

CDH(4)

网络协议(3)

php(2)

Hive(2)

更多

随笔档案

2024年5月(1)

2024年4月(1)

2024年3月(4)

2024年2月(2)

2024年1月(4)

2023年12月(5)

2023年11月(5)

2023年10月(4)

2023年9月(7)

2023年8月(6)

2023年7月(2)

2023年6月(9)

2023年5月(15)

2023年4月(15)

2023年3月(7)

更多

阅读排行榜

1. PyQt5中QWidget设置列宽大小的几种方式(45513)

2. PyQt5-实时刷新页面（QApplication.p

Python 之 WSGI、uWSGI 和 uwsgi 介绍

目录

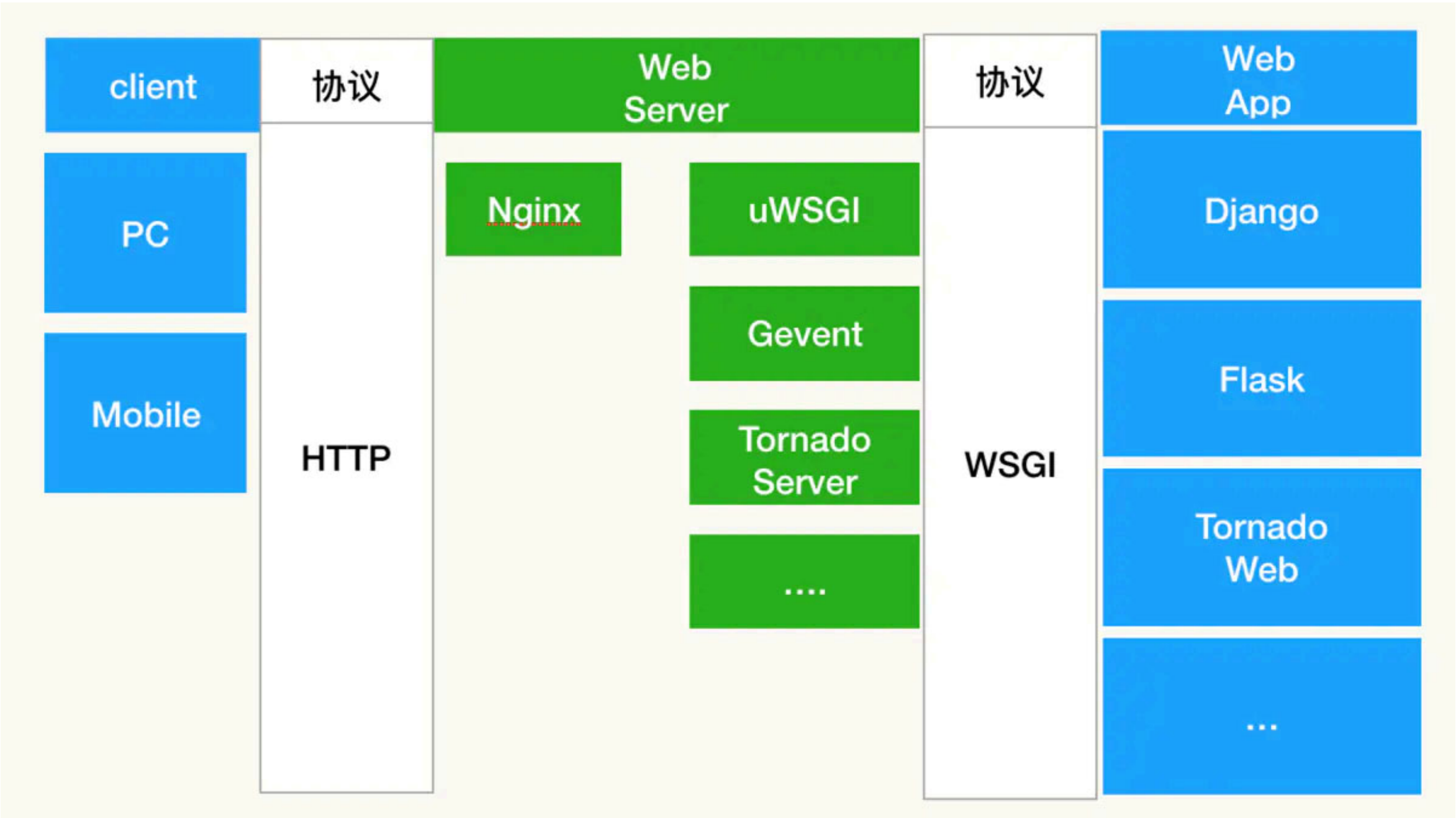
- 一、概述
- 二、安装 uwsgi 模块
 - 1) 配置pip源
 - 2) 安装 uwsgi 模块
- 三、示例演示（uWSGI + Nginx 配置）
 - 1) 安装 nginx
 - 2) 创建 app.py 文件
 - 3) 创建 uWSGI 配置文件
 - 4) 启动 uWSGI
 - 5) 配置 Web 服务器
 - 6) 重启 Web 服务器
 - 7) Nginx upstream 负载均衡
 - 1、轮询（默认）
 - 2、权重（weight）
 - 3、IP散列（ip_hash）
 - 8) http、http-socket 和 socket 区别
 - 9) TCP 与 unix 区别

一、概述

WSGI、uWSGI 和 uwsgi 是三个相关的概念，它们是在 Web 应用程序开发中使用的不同的工具和协议。下面是它们的详细介绍：

- WSGI（Web Server Gateway Interface）**：WSGI 是一个 Python Web 应用程序与 Web 服务器之间的**接口规范**，它定义了应用程序和服务器之间的标准接口，使得应用程序可以在不同的 Web 服务器上运行。WSGI 规范规定了应用程序必须实现的接口方法和服务器需要支持的方法。**WSGI 协议**使得不同的 Python Web 框架（例如 Flask、Django 等）能够在不同的 Web 服务器上运行，这些服务器可以是 Apache、Nginx 等。
- uWSGI**：uWSGI 是一个 **Web 服务器**，它是一个用 C 语言编写的 Web 应用程序容器，支持运行 Python、Ruby、Perl 等多种编程语言。uWSGI 服务器可以作为一个独立的应用服务器，也可以与其他 Web 服务器（如 Nginx、Apache）一起使用，通过 WSGI 协议与 Python 应用程序通信。
- uwsgi**：**uwsgi 是一个与 uWSGI 服务器相关的协议**。uwsgi 协议是一种二进制协议，它定义了 uWSGI 服务器与应用程序之间的通信协议。使用 uwsgi 协议，uWSGI 服务器可以与 Python 应用程序通信，而不需要像 CGI 那样启动一个新的进程来处理每个请求。uwsgi 协议允许 uWSGI 服务器与应用程序之间进行双向通信，从而提高了性能。

因此，uWSGI 是一个 Web 服务器，可以通过 WSGI 协议与 Python 应用程序通信，并使用 uwsgi 协议进行通信。WSGI 是 Python Web 应用程序与 Web 服务器之间的接口规范，定义了应用程序和服务器之间的标准接口。而 uwsgi 则是 uWSGI 服务器与应用程序之间的二进制通信协议。



二、安装 uwsgi 模块

uWSGI 是一种 Web 服务器网关接口（Web Server Gateway Interface），它可以用于将 Python Web 应用程序与 Web 服务器（如 Nginx 或 Apache）集成在一起。

- 在使用uWSGI模块时，需要安装 **uwsgi** 模块，并在Python Web应用程序中导入 **uwsgi** 模块，并使用uwsgi模块提供的函数来配置和管理Web应用程序的运行。常见的uwsgi模块函数包括uwsgi.optin()、uwsgi.route()、uwsgi.applications()等。
- 另外，uWSGI模块还提供了一些高级特性，如Master/Worker模式、进程管理、负载均衡、自动扩展等，使得Web应用程序可以更好地适应高并发和大流量的情况。

1) 配置pip源

国内源地址：

- pypi 清华大学源：<https://pypi.tuna.tsinghua.edu.cn/simple>
- pypi 腾讯源：<http://mirrors.cloud.tencent.com/pypi/simple>
- pypi 阿里源：<https://mirrors.aliyun.com/pypi/simple/>

rocessEvents()) (30803)
3. VMware vCenter安装及基本使用(27043)
4. python中socket模块详解(24970)
5. 详解VMware 虚拟机中添加新硬盘的方法(23368)

评论排行榜

1. k8s+Prometheus+Grafana的监控部署(10)
2. 【云原生】Hadoop HA on k8s 环境部署(6)
3. Zabbix4.x 历史数据存储到Elasticsearch7.x(3)
4. 大数据Hadoop之——EFAK和Confluent KSQL简单使用 (kafka listeners 和 advertised.listeners) (2)
5. 【云原生】Prometheus+Grafana on K8s 环境部署(2)

推荐排行榜

1. 详解VMware 虚拟机中添加新硬盘的方法(5)
2. 【云原生】Containerd ctr 和 crictl 客户端命令介绍与实战操作 (nerdctl) (4)
3. 高性能分布式对象存储——MinIO实战操作 (MinIO扩容) (3)
4. wireshark网络抓包详解(3)
5. [云原生] Kubernetes (k8s) 健康检查详解与实战演示 (就绪性探针 和 存活性探针) (2)

最新评论

1. Re:Hadoop on k8s 编排部署进阶篇	老师你好，咨询一个问题： 我想把hdfs持久化从hostpath改成pvc方式，制作好了storageclass，在values.yaml里添加了persistentVolumeClaim。在将ena...
--刚大木	
2. Re:VMware vCenter安装及基本使用	按照教程顺序安装好了，登陆不显示网页提示安装flash插件，装好插件就....登陆页面都没有了，显示连接错误，不知道哪里错了，添加了Windows的一些功能才能成功安装flash插件，不知道是不是安装...
--冰花花	
3. Re:VMware vCenter安装及基本使用	无意中知道了虚拟机，无聊中玩了玩就有点着迷的感觉了
--冰花花	
4. Re:大数据Hadoop之——Hadoop 3.3.4 HA (高可用) 原理与实现 (QJM)	实验非常成功，好文章，良心配置文件
--爪哇终结者	
5. Re:大数据Hadoop之——Spark on Hive 和 Hive on Spark的区别与实现	"由于Hive on MapReduce的缺陷，所以企业里基本上很少使用了。" 请问是有哪些缺陷？
--000A_A000	

```
mkdir ~/.pip/
cat > ~/.pip/pip.conf<<EOF
[global]
index-url = https://repo.huaweicloud.com/repository/pypi/simple
trusted-host = repo.huaweicloud.com
timeout = 120
EOF
```

2) 安装 uwsgi 模块

```
# 安装python3
yum -y install python3

yum -y install gcc-c++ -y
yum -y install python3-devel -y

# 安装 uwsgi flask 模块
pip3 install uwsgi flask

# 查看版本
uwsgi --version
```

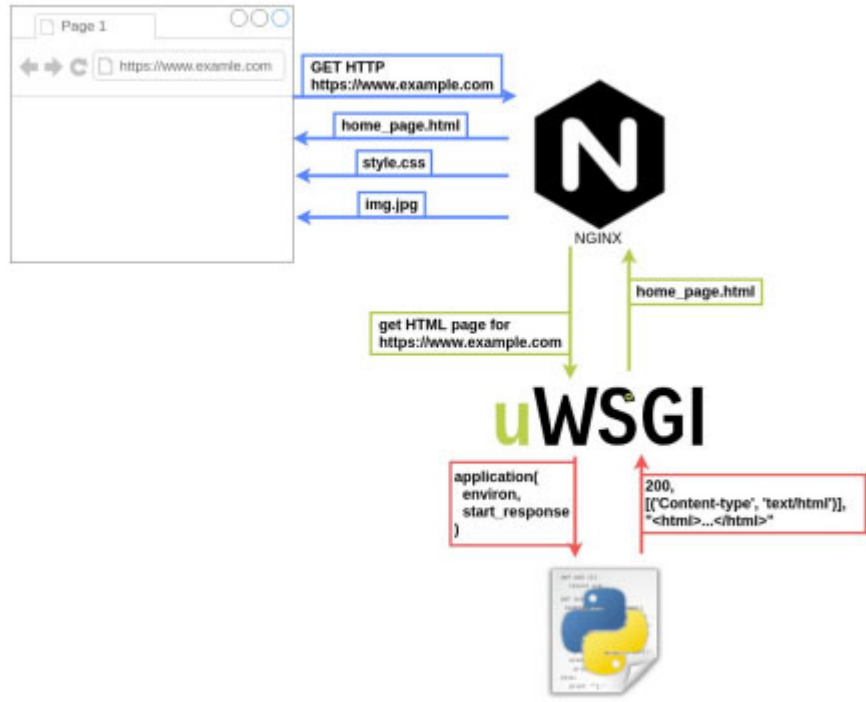
```
Verifying : redhat-rpm-config-9.1.0-88.el7.centos.noarch
Verifying : python3-rpm-macros-3-34.el7.noarch

Installed:
python3-devel.x86_64 0:3.6.8-18.el7

Dependency Installed:
dwz.x86_64 0:0.11-3.el7
python-srpm-macros.noarch 0:3-34.el7
redhat-rpm-config.noarch 0:9.1.0-88.el7.centos
perl-srpm-macros.noarch 0:1-8.el7
python3-rpm-generators.noarch 0:6-2.el7
python3-rpm-macros.noarch 0:3-34.el7
zip.x86_64 0:3.0-11.el7

Complete!
[root@local-168-182-110 ~]# pip3 install uwsgi
WARNING: Running pip install with root privileges is generally not a good idea. Try `pip3 install --user` instead.
Collecting uwsgi
  Downloading https://repo.huaweicloud.com/repository/pypi/packages/24/fd/93851e4a076719199868d4c918cc93a52742e68370188c1c570a6e42a54f/uwsgi-2.0.20.tar.gz (808kB)
    100% |#####| 808kB 16.0MB/s
Installing collected packages: uwsgi
  Running setup.py install for uwsgi ... done
Successfully installed uwsgi-2.0.20
[root@local-168-182-110 ~]# uwsgi --version
2.0.20
[root@local-168-182-110 ~]#
```

三、示例演示 (uWSGI + Nginx 配置)



1) 安装 nginx

```
yum update -y
yum install epel-release
yum -y install nginx
```

2) 创建 app.py 文件

创建一个名为 `app.py` 的文件，添加以下代码：

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, World!'

if __name__ == '__main__':
    app.run()
```

3) 创建 uWSGI 配置文件

创建一个 uWSGI 配置文件，例如 `uwsgi.ini`，其中包含以下信息：

```
[uwsgi]
module = app:app
# 相当于命令下面两行
#wsgi-file = app.py # 项目入口文件
#callable = app # flask应用对象
# 支持http+socket两种方式，这里选用socket，也可以选择http+socket，下面会讲解这三种区别
# http = 127.0.0.1:8000
socket = 0.0.0.0:8000
# 也可以使用socket文件，在nginx配置文件中配置也对应，仅限本机通信，一般也很少使用
# socket = /app/myapp.sock

# 注意记得提前创建目录
chdir = /opt/myapp
pidfile=/opt/myapp/myapp.pid
processes = 4
threads = 2
master = true
vacuum = true
py-autoreload = 1
daemonize = /tmp/uwsgi.log
```

uwsgi.ini常用配置参数详解：

- `chdir` =/xxx/xxx # 指定项目目录， 这里写上程序根目录(即 `app.py` 文件所在目录)对应上述目录结构为 `src`

- `home =/xxx/xxx # 指定虚拟环境变量`
- `wsgi-file =xxx # 指定加载WSGI文件`
- `socket =xxx # 指定uwsgi的客户端将要连接的socket的路径（使用UNIX socket的情况）或者地址（使用网络地址的情况）。#socket协议，用于和nginx通讯,端口可配置成别的端口；如果有nginx在uwsgi之前作为代理的话应该配socket 如：socket=0.0.0.0:5000。当然也可以使用 http-socket #而如果客户端请求不经过(不搭建)Nginx代理服务器,服务请求直接到uwsgi服务器的话那么就配http。如：http=0.0.0.0:5000;IP和端口与项目启动文件app.py中一致; 127.0.0.1虽然是表示本地IP，但想要在网络上访问必须设置host=0.0.0.0才不受IP限制。`
- `callable =app # 这个 app 指的是 flask 项目启动程序中定义的 flask name 的名字，我的启动程序是 app.py，里面定义的 flask 的名字是 app。`
- `module =mysite.wsgi # 加载一个WSGI模块,这里加载mysite/wsgi.py这个模块`
- ``master=true # 指定启动主进程`
- ``processes=4 # 设置工作进程的数量`
- `threads =2 # 设置每个工作进程的线程数`
- `vacuum =true # 当服务器退出时自动删除unix socket文件和pid文件`
- `logfile-chmod =644 # 指定日志文件的权限`
- `daemonize =%(chdir)/xxx.log # 进程在后台运行，并将日志打印到指定文件`
- `pidfile =%(chdir)/xxx.pid # 在失去权限前，将主进程pid写到指定的文件`
- `uid =xxx # uWSGI服务器运行时的用户id`
- `gid =xxx # uWSGI服务器运行时的用户组id`
- `procname-prefix-spaced =xxx # 指定工作进程名称的前缀`
- `chdir =/xxx/xxx # 指定项目目录，这里写上程序根目录(即 app.py 文件所在目录)对应上述目录结构为 /opt/uwsgi/`
- `listen = 120 # 设置socket的监听队列大小（默认：100）`

4) 启动 uWSGI

在命令行中启动 uWSGI:

```
uwsgi --ini uwsgi.ini
###或者
uwsgi uwsgi.ini

### 重启
uwsgi --reload /opt/myapp/myapp.pid
###关闭
uwsgi --stop /opt/myapp/myapp.pid
```

```
detected number of CPU cores: 4
current working directory: /etc/nginx
detected binary path: /usr/local/bin/uwsgi
!!! no internal routing support, rebuild with pcre support !!!
uWSGI running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
chdir() to /opt/myapp
your processes number limit is 7736
your memory page size is 4096 bytes
detected max file descriptor number: 1024
lock engine: pthread robust mutexes
thunder lock: disabled (you can enable it with --thunder-lock)
uwsgi socket 0 bound to TCP address 127.0.0.1:8000 fd 3
uWSGI running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
Python version: 3.6.8 (default, Nov 16 2020, 16:55:22) [GCC 4.8.5 20150623 (Red Hat 4.8.5-44)]
Python main interpreter initialized at 0xedd780
uWSGI running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
python threads support enabled
your server socket listen backlog is limited to 100 connections
your mercy for graceful operations on workers is 60 seconds
mapped 416720 bytes (406 KB) for 8 cores
*** Operational MODE: preforking+threaded ***
failed to open python file myapp.wsgi
unable to load app 0 (mountpoint='') (callable not found or import error)
*** no app loaded. going in full dynamic mode ***
uWSGI running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 52731)
spawned uWSGI worker 1 (pid: 52732, cores: 2)
spawned uWSGI worker 2 (pid: 52733, cores: 2)
spawned uWSGI worker 3 (pid: 52734, cores: 2)
spawned uWSGI worker 4 (pid: 52735, cores: 2)
Python auto-reloader enabled
[root@local-168-182-110 nginx]#
[root@local-168-182-110 nginx]#
[root@local-168-182-110 nginx]# netstat -tnlp|grep 8000
tcp        0      0 127.0.0.1:8000      0.0.0.0:*           LISTEN      52731/uwsgi
[root@local-168-182-110 nginx]# █
```

【温馨提示】其实也可以通过一条命令带上对应的参数即可启动，但是不推荐，测试可以。一般使用配置文件的方式启动服务。

使用http协议启动uwsgi的命令为:

```
uwsgi --http :8000 --ini uwsgi_conf.ini -d ./uwsgi.log --pidfile=uwsgi.pid
```

- `--http` 指定用5800端口启动http协议
- `--ini` 指定上述的启动配置文件
- `-d` 指定uwsgi的log，方便我们调试
- `--pidfile` 将启动的进程号写到 `uwsgi.pid` 文件中，方便我们在需要停止服务器时kill掉。

5) 配置 Web 服务器

将 Web 服务器配置为反向代理 uWSGI，例如，在 Nginx 中，可以使用以下配置文件：

```
# vi /etc/nginx/conf.d/myapp.conf
server {
    listen 8080;
    server_name myapp.com;
    location / {
        include uwsgi_params;
        uwsgi_pass 127.0.0.1:8000;
    }
}
```

其中， `uwsgi_params` 文件包含以下内容：

```
uwsgi_param QUERY_STRING      $query_string;
uwsgi_param REQUEST_METHOD    $request_method;
uwsgi_param CONTENT_TYPE      $content_type;
uwsgi_param CONTENT_LENGTH    $content_length;

uwsgi_param REQUEST_URI       $request_uri;
uwsgi_param PATH_INFO          $document_uri;
uwsgi_param DOCUMENT_ROOT     $document_root;
uwsgi_param SERVER_PROTOCOL   $server_protocol;
uwsgi_param REQUEST_SCHEME    $scheme;
uwsgi_param HTTPS              $https if_not_empty;

uwsgi_param REMOTE_ADDR       $remote_addr;
uwsgi_param REMOTE_PORT       $remote_port;
uwsgi_param SERVER_PORT       $server_port;
uwsgi_param SERVER_NAME       $server_name;
```

```
[root@local-168-182-110 ~]# cd /etc/nginx/
[root@local-168-182-110 nginx]# ll
total 68
drwxr-xr-x 2 root root  6 Nov 11 00:58 conf.d
drwxr-xr-x 2 root root  6 Nov 11 00:58 default.d
-rw-r--r-- 1 root root 1077 Nov 11 00:58 fastcgi.conf
-rw-r--r-- 1 root root 1077 Nov 11 00:58 fastcgi.conf.default
-rw-r--r-- 1 root root 1007 Nov 11 00:58 fastcgi_params
-rw-r--r-- 1 root root 1007 Nov 11 00:58 fastcgi_params.default
-rw-r--r-- 1 root root 2837 Nov 11 00:58 koi-utf
-rw-r--r-- 1 root root 2223 Nov 11 00:58 koi-win
-rw-r--r-- 1 root root 5231 Nov 11 00:58 mime.types
-rw-r--r-- 1 root root 5231 Nov 11 00:58 mime.types.default
-rw-r--r-- 1 root root 2336 Nov 11 00:58 nginx.conf
-rw-r--r-- 1 root root 2656 Nov 11 00:58 nginx.conf.default
-rw-r--r-- 1 root root  636 Nov 11 00:58 scgi_params
-rw-r--r-- 1 root root  636 Nov 11 00:58 scgi_params.default
-rw-r--r-- 1 root root  664 Nov 11 00:58 uwsgi_params
-rw-r--r-- 1 root root  664 Nov 11 00:58 uwsgi_params.default
-rw-r--r-- 1 root root 3610 Nov 11 00:58 win-utf
[root@local-168-182-110 nginx]# cat uwsgi_params

uwsgi_param QUERY_STRING      $query_string;
uwsgi_param REQUEST_METHOD    $request_method;
uwsgi_param CONTENT_TYPE      $content_type;
uwsgi_param CONTENT_LENGTH    $content_length;

uwsgi_param REQUEST_URI       $request_uri;
uwsgi_param PATH_INFO          $document_uri;
uwsgi_param DOCUMENT_ROOT     $document_root;
uwsgi_param SERVER_PROTOCOL   $server_protocol;
uwsgi_param REQUEST_SCHEME    $scheme;
uwsgi_param HTTPS              $https if_not_empty;

uwsgi_param REMOTE_ADDR       $remote_addr;
uwsgi_param REMOTE_PORT       $remote_port;
uwsgi_param SERVER_PORT       $server_port;
uwsgi_param SERVER_NAME       $server_name;
[root@local-168-182-110 nginx]# pwd
/etc/nginx
[root@local-168-182-110 nginx]# █
```

【特别注意】 `uwsgi_params` 在nginx conf文件夹下自带，uwsgi_pass一定要跟uwsgi_conf.ini中写的地址完全一致。

6) 重启 Web 服务器

重启 Web 服务器以使配置生效。

```
# 重启
systemctl restart nginx

# 如果是之前nginx:服务已经存在，只是修改了配置，可建议使用reload加载
nginx -t && nginx -s reload
# 或者
systemctl reload nginx
```

访问（浏览器访问，curl访问也行）

```
your mercy for graceful operations on workers is 60 seconds
mapped 416720 bytes (406 KB) for 8 cores
*** Operational MODE: preforking+threaded ***
WSGI app 0 (mountpoint='') ready in 0 seconds on interpreter 0x26968e0 pid: 78981 (default app)
uwsgi running as root, you can use --uid/--gid/--chroot options
*** WARNING: you are running uWSGI as root !!! (use the --uid flag) ***
*** uWSGI is running in multiple interpreter mode ***
spawned uWSGI master process (pid: 78981)
spawned uWSGI worker 1 (pid: 79034, cores: 2)
spawned uWSGI worker 2 (pid: 79035, cores: 2)
spawned uWSGI worker 3 (pid: 79038, cores: 2)
spawned uWSGI worker 4 (pid: 79041, cores: 2)
Python auto-reloader enabled
[pid: 79034|app: 0|req: 1/1] 127.0.0.1 ( ) {32 vars in 351 bytes} [Fri Feb 24 21:37:37 2023] HEAD / => generated 0 bytes in 2 msecs (HTTP/1.1 200) 2 headers in 79 bytes (0 switches on core 0)
[root@local-168-182-110 myapp]# curl -I localhost:8080
HTTP/1.1 200 OK
Server: nginx/1.20.1
Date: Fri, 24 Feb 2023 13:38:09 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 13
Connection: keep-alive

[root@local-168-182-110 myapp]# █
```

7) Nginx upstream 负载均衡

Nginx上游（upstream）是指一组后端服务器，Nginx可以与其通信并将客户端请求转发到这些服务器。换句话说，上游服务器是Nginx代理请求的后端服务器。

Nginx的upstream支持5种 分配方式，其中 **轮询（默认）**、**权重**、**IP散列**这三种为Nginx原生支持的分配方式， **fair** 和 **url_hash** 为第三方支持的分配方式。

1、轮询（默认）

轮询是 **upstream** 的**默认分配方式**，即每个请求按照时间顺序轮流分配到不同的后端服务器，如果某个后端服务器 down 掉后，能自动剔除。

```
upstream backend {
    server 192.168.182.110:8000;
    server 192.168.182.111:8000;
}
```

2、权重（weight）

轮询的加强版，既可以指定轮询比率，**weight 和访问几率成正比**，主要应用于后端服务器异质的场景下。

```
upstream backend {
    server 192.168.182.110:8000 weight=1;
    server 192.168.182.111:8000 weight=2;
}
```

3、IP散列（ip_hash）

每个请求按照访问 Ip（即Nginx的前置服务器或客户端IP）的 hash结果分配，这样**每个访客会固定访问一个后端服务器**，可以解决 session 一致问题。

```
upstream backend {
    ip_hash;
    server 192.168.182.110:8000 weight=1;
    server 192.168.182.111:8000 weight=2;
}
```

先在另外一个节点上再起一个uWSGI服务，将上面示例配置修改：

```
# vi /etc/nginx/conf.d/myapp.conf
upstream backend {
    server 192.168.182.110:8000;
    server 192.168.182.111:8000;
}

server {
    listen 8080;
    server_name myapp.com;
    location / {
        include uwsgi_params;
        uwsgi_pass backend;
    }
}
```

192.168.182.110 节点 **app.py**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, World 192.168.182.110!\n'

if __name__ == '__main__':
    app.run()
```

192.168.182.111 节点 **app.py**

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello, World 192.168.182.111!\n'

if __name__ == '__main__':
    app.run()
```

验证

```
curl 127.0.0.1:8080
```



```
[root@local-168-182-110 ~]#  
[root@local-168-182-110 ~]#  
[root@local-168-182-110 ~]# cat /etc/nginx/conf.d/myapp.conf  
upstream backend {  
    server 192.168.182.110:8000;  
    server 192.168.182.111:8000;  
}  
  
server {  
    listen 8080;  
    server_name myapp.com;  
    location / {  
        include uwsgi_params;  
        uwsgi_pass backend;  
    }  
}  
[root@local-168-182-110 ~]#  
[root@local-168-182-110 ~]#  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.111!  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.111!  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.110!  
[root@local-168-182-110 ~]#  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.111!  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.110!  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.110!  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.110!  
[root@local-168-182-110 ~]# curl 127.0.0.1:8080  
Hello, World 192.168.182.110!  
[root@local-168-182-110 ~]#
```

从上图可知，请求轮询调度，这才是企业一般想要的效果，负载均衡。

8) http、http-socket 和 socket 区别

- `http` 和 `http-socket` 的区别在于，如果我们想**直接将uwsgi用作服务器**（例如Apache和nginx那样）直接暴露在公网那么就使用 `http` ；
- 如果有单独的服务器（例如Apache或者nginx），由服务器将请求转发给uwsgi处理，并且使用 `http` 协议，那么此时使用 `http-socket` 。
- `http`：自己会产生一个http进程(可以认为与nginx同一层)负责路由 `http` 请求给 `worker`，`http` 进程和 `worker` 之间使用的是uwsgi协议。
- `http-socket`：不会产生http进程,一般用于在前端webserver不支持 `uwsgi` 而仅支持http时使用,他产生的worker使用的是http协议。
- 因此，`http` 一般是作为独立部署的选项; `http-socket` 在前端webserver不支持uwsgi时使用, 如果前端webserver支持uwsgi, 则直接使用socket即可(tcp or unix)。

【1】 `socket` 示例 (uwsgi.ini)：

```
[uwsgi]  
module = app:app  
#socket = 127.0.0.1:8000  
socket = 0.0.0.0:8000  
  
chdir = /opt/myapp  
pidfile=/opt/myapp/myapp.pid  
processes = 4  
threads = 2  
master = true  
vacuum = true  
py-autoreload = 1  
daemonize = /tmp/uwsgi.log
```

nginx配置

```
upstream backend {  
    server 192.168.182.110:8000;  
    server 192.168.182.111:8000;  
}  
  
server {  
    listen 8080;  
    server_name myapp.com;  
    location / {  
        include uwsgi_params;  
        uwsgi_pass backend;  
    }  
}
```

【2】 `http` 示例 (uwsgi.ini)：

```
[uwsgi]  
module = app:app  
socket = 0.0.0.0:8000  
  
chdir = /opt/myapp  
pidfile=/opt/myapp/myapp.pid  
processes = 4  
threads = 2  
master = true  
vacuum = true  
py-autoreload = 1  
daemonize = /tmp/uwsgi.log
```

nginx配置

```
upstream backend {  
    server 192.168.182.110:8000;  
    server 192.168.182.111:8000;  
}  
  
server {  
    listen 8080;  
    server_name myapp.com;  
    location / {  
        include uwsgi_params;  
        proxy_pass http://backend;  
    }  
}
```

【3】 http-socket 示例 (uwsgi.ini)：

```
[uwsgi]
module = app:app
http = 0.0.0.0:8000

chdir = /opt/myapp
pidfile=/opt/myapp/myapp.pid
processes = 4
threads = 2
master = true
vacuum = true
py-autoreload = 1
daemonize = /tmp/uwsgi.log
```

nginx配置

```
upstream backend {
    server 192.168.182.110:8000;
    server 192.168.182.111:8000;
}

server {
    listen 8080;
    server_name myapp.com;
    location / {
        include uwsgi_params;
        proxy_pass http://backend;
    }
}
```

9) TCP 与 unix 区别

TCP和Unix套接字（Unix domain socket）是两种不同类型的套接字。

- TCP套接字是基于TCP/IP协议的网络套接字，用于在网上进行进程间通信。TCP套接字需要指定IP地址和端口号，以便其他进程可以连接到该套接字进行通信。TCP套接字是一种跨网络边界的套接字，可以在不同的计算机之间进行通信。TCP套接字常用于客户端/服务器架构中，如Web服务器、数据库服务器等。
- Unix套接字是基于Unix域套接字（Unix domain socket）的本地套接字，用于在同一台计算机上进行进程间通信。Unix套接字只需要指定一个文件路径，而不需要使用IP地址和端口号。Unix套接字是一种进程间通信（IPC）机制，它提供了高效、可靠和安全的进程间通信方式。Unix套接字通常用于本地服务器和本地客户端之间的通信，例如X Window系统中的客户端和服务端。

因此，TCP套接字用于在网上进行通信，而Unix套接字用于在同一台计算机上进行通信。虽然TCP套接字可以通过网络连接到不同的计算机，但是Unix套接字提供了更高效的进程间通信机制，并且更适合于需要在同一台计算机上运行的进程间通信。

【TCP 示例】常用

uwsgi.ini：

```
[uwsgi]
module = app:app
socket = 127.0.0.1:8000

chdir = /opt/myapp
pidfile=/opt/myapp/myapp.pid
processes = 4
threads = 2
master = true
vacuum = true
py-autoreload = 1
daemonize = /tmp/uwsgi.log
```

【unix 示例】仅限于本机通信，很少使用。

uwsgi.ini：

```
[uwsgi]
module = app:app
socket = /opt/myapp/myapp.socket

chdir = /opt/myapp
pidfile=/opt/myapp/myapp.pid
processes = 4
threads = 2
master = true
vacuum = true
py-autoreload = 1
daemonize = /tmp/uwsgi.log
```

nginx配置

```
server {
    listen 8080;
    server_name myapp.com;
    location / {
        include uwsgi_params;
        proxy_pass unix:///opt/myapp/myapp.sock;
    }
}
```

Python 中 web开发中的 WSGI、uWSGI 和 uwsgi 三者介绍就先到这里了，有任何疑问欢迎给我留言或私信，也可关注我的公众号【大数据与云原生技术分享】深入技术交流~





微信搜一搜




大数据与云原生技术分享

好文要顶

关注我

收藏该文

微信分享



大数据老司机

粉丝 - 243 关注 - 1

+加关注

0

0

升级成为会员

« 上一篇: [23种设计模式介绍 \(Python示例讲解\)](#)

» 下一篇: [【云原生】k8s 离线部署讲解和实战操作](#)

posted @ 2023-02-24 22:28 大数据老司机 阅读(817) 评论(0) 编辑 收藏 举报

[刷新页面](#) [返回顶部](#)

登录后才能查看或发表评论, 立即 [登录](#) 或者 [逛逛](#) 博客园首页



编辑推荐:

- [golang slice](#)相关常见的性能优化手段
- 谈一谈 Netty 的内存管理，且看 Netty 如何实现 Java 版的 Jemalloc
- [修复一个kubernetes集群](#)
- AOT漫谈专题(第六篇): C# AOT 的泛型,序列化,反射问题
- .NET云原生应用实践（三）：连接到PostgreSQL数据库

阅读排行:

- [一个整合性、功能丰富的.NET网络通信框架](#)
- [Python实现微博舆情分析的设计与实现](#)
- [python编程基础](#)
- [FFmpeg开发笔记（六十）使用国产的ijkplayer播放器观看网络视频](#)
- [只需初中数学知识就能理解人工智能大语言模型](#)