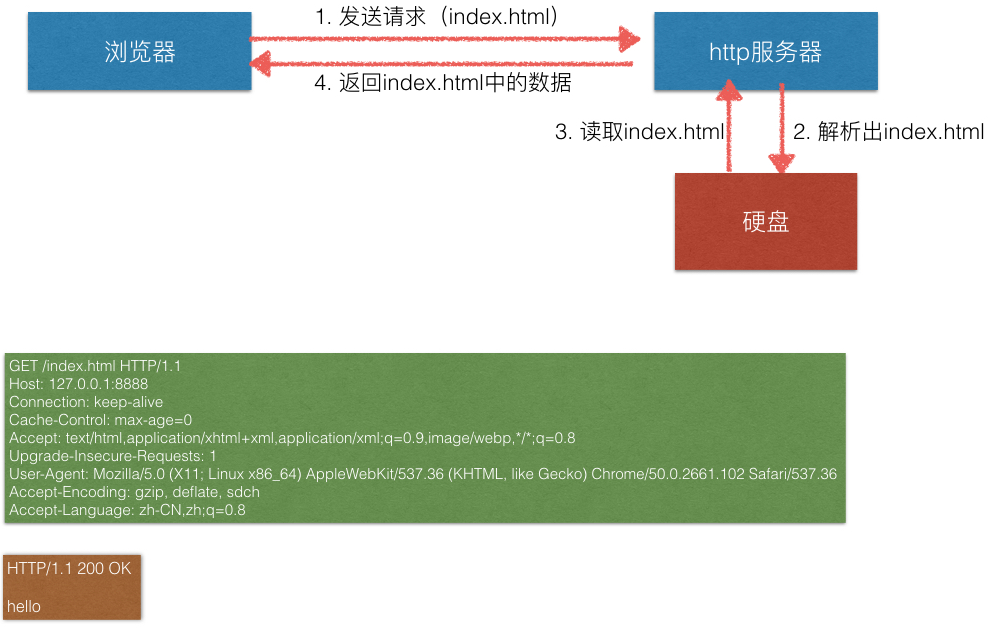
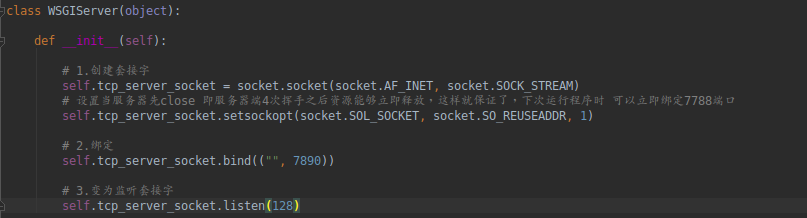


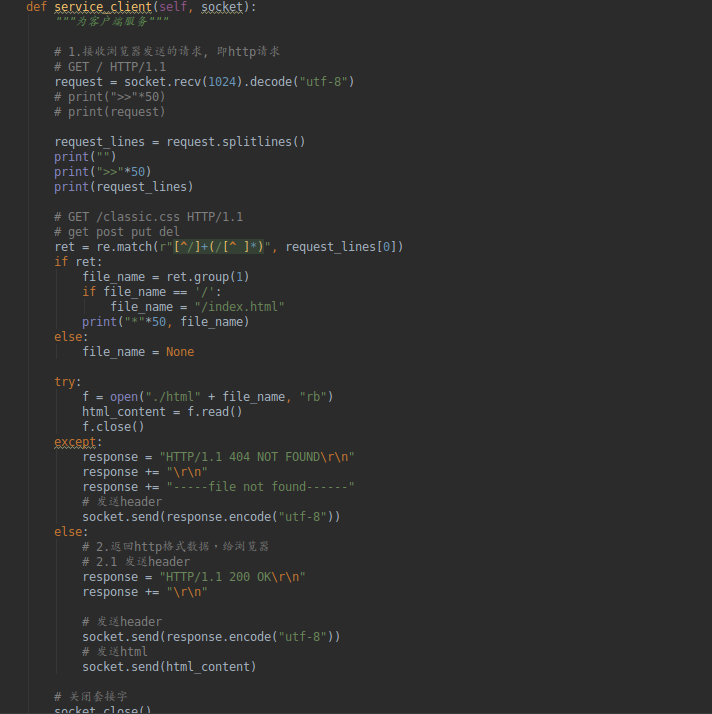
**一、WSGI-mini-web框架**

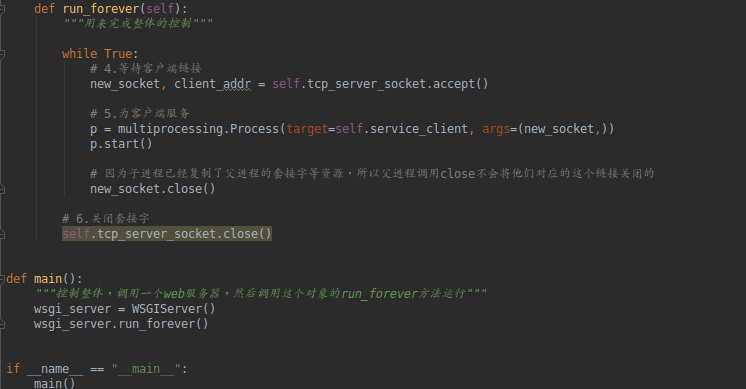
**1. 静态服务器**



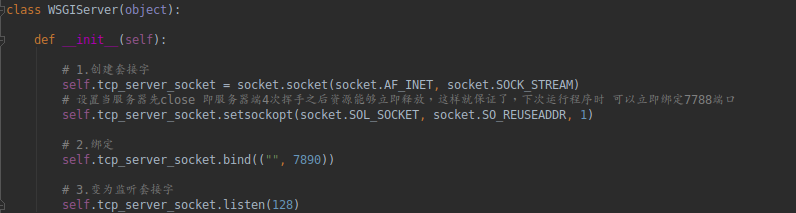
浏览器直接申请资源，服务器提取url信息，读取后发送数据。

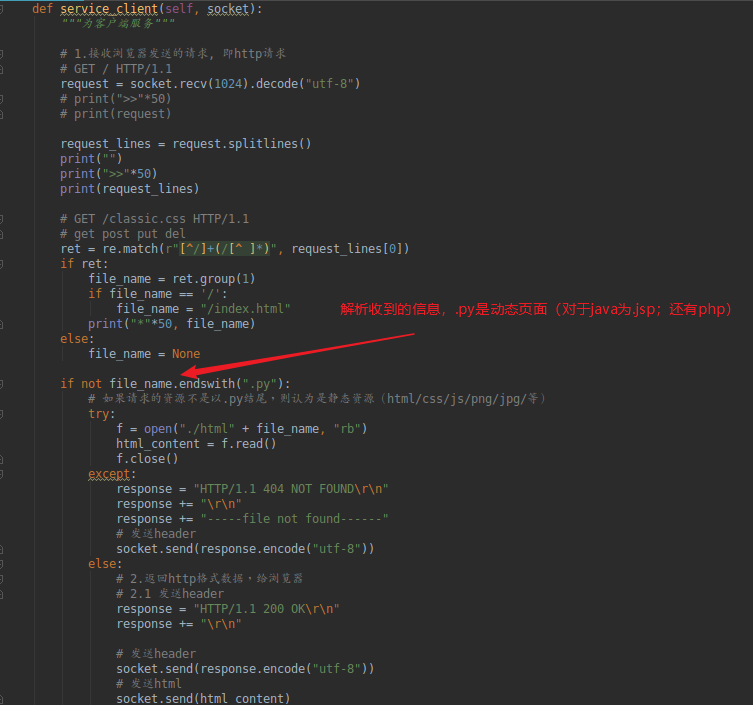


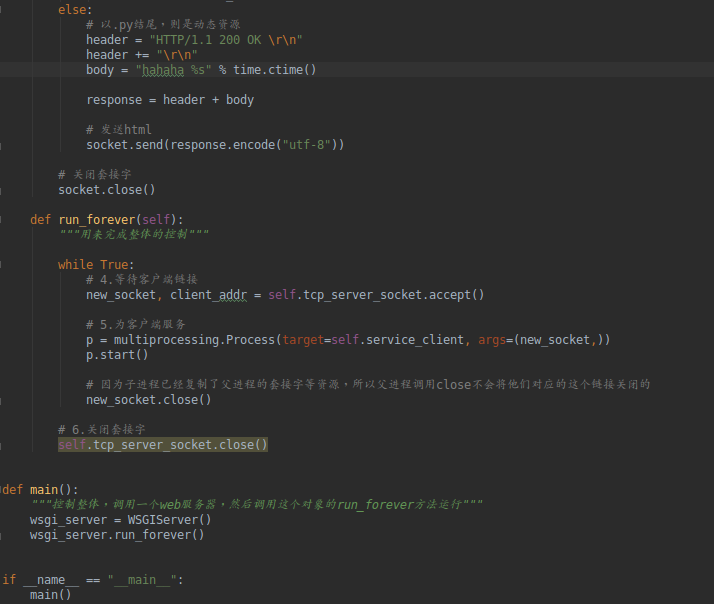




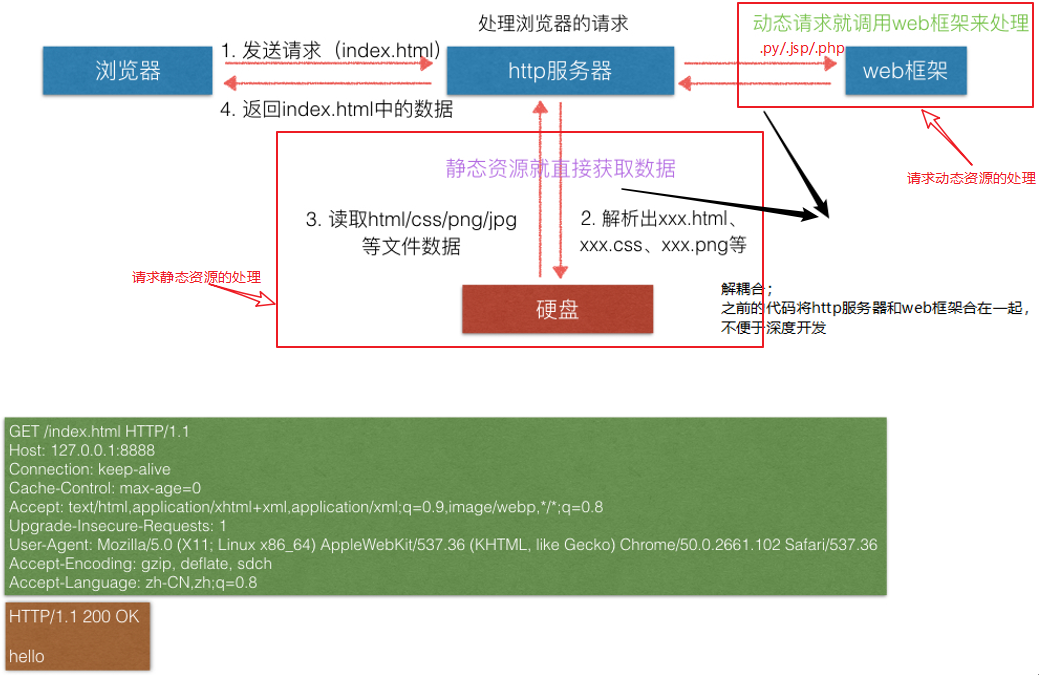
**2. 动态服务器**



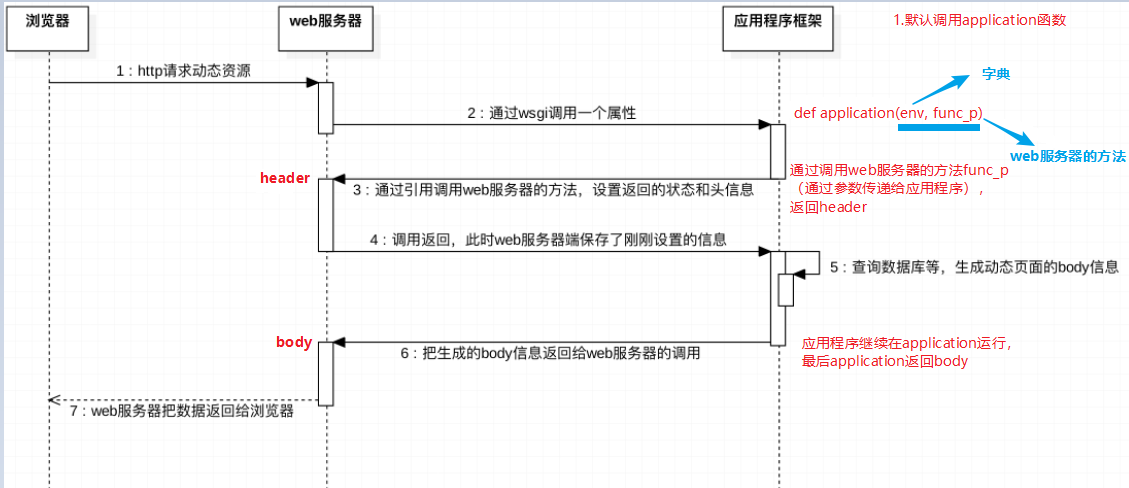




静态服务器是静态的资源，如js/css/html/jpg/bnp等，动态的是根据代码要求进行变化；



* 1. **服务器动态资源请求**

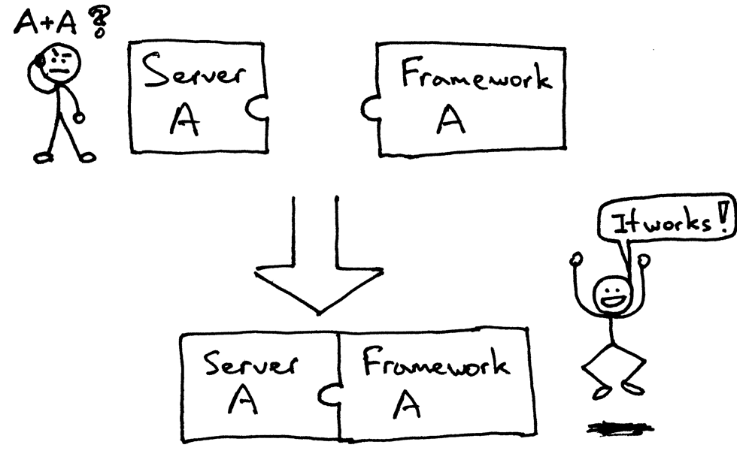


应用程序框架通过调用application，application传递两个参数，第一个是字典，第二个是web服务器的方法，应用程序通过调用web服务器方法func\_p返回header，再继续运行application返回body。

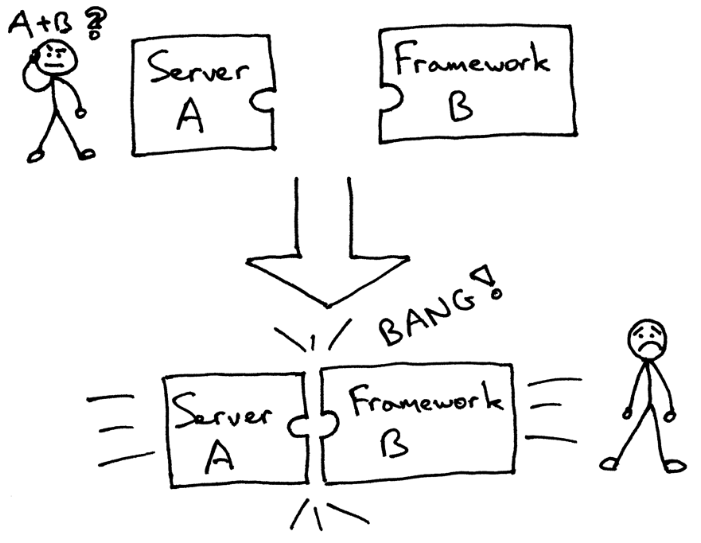
* 1. **WSGI**

怎么在你刚建立的Web服务器上运行一个Django应用和Flask应用，如何不做任何改变而适应不同的web架构呢？

在以前，选择 Python web 架构会受制于可用的web服务器，反之亦然。如果架构和服务器可以协同工作，那就好了：

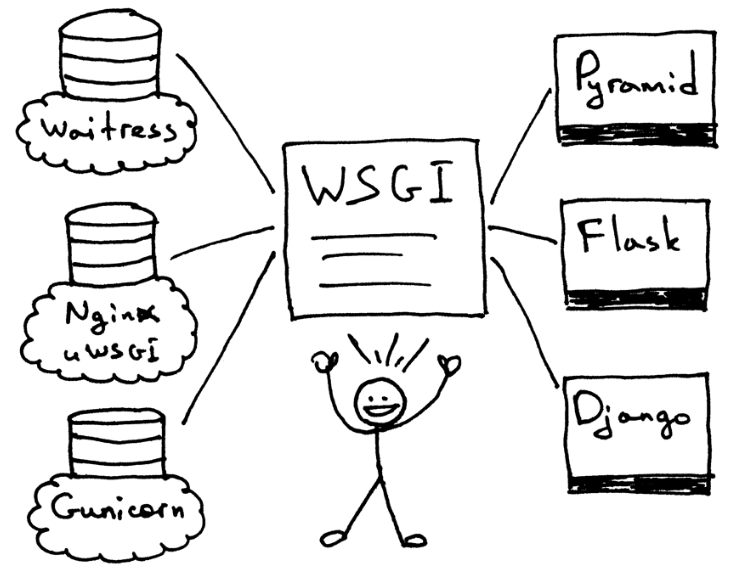


但有可能面对（或者曾有过）下面的问题，当要把一个服务器和一个架构结合起来时，却发现他们不是被设计成协同工作的：



那么，怎么可以不修改服务器和架构代码而确保可以在多个架构下运行web服务器呢？答案就是 Python Web Server Gateway Interface (或简称 WSGI，读作“wizgy”)。

WSGI允许开发者将选择web框架和web服务器分开。可以混合匹配web服务器和web框架，选择一个适合的配对。比如,可以在Gunicorn 或者 Nginx/uWSGI 或者 Waitress上运行 Django, Flask, 或 Pyramid。真正的混合匹配，得益于WSGI同时支持服务器和架构：



web服务器必须具备WSGI接口，所有的现代Python Web框架都已具备WSGI接口，它让你不对代码作修改就能使服务器和特点的web框架协同工作。

WSGI由web服务器支持，而web框架允许你选择适合自己的配对，但它同样对于服务器和框架开发者提供便利使他们可以专注于自己偏爱的领域和专长而不至于相互牵制。其他语言也有类似接口：java有Servlet API，Ruby 有 Rack。

* 1. **定义WSGI接口**

WSGI接口定义非常简单，它只要求Web开发者实现一个函数，就可以响应HTTP请求。我们来看一个最简单的Web版本的“Hello World!”：

def application(environ, start\_response):

start\_response('200 OK', [('Content-Type', 'text/html')])

return 'Hello World!'

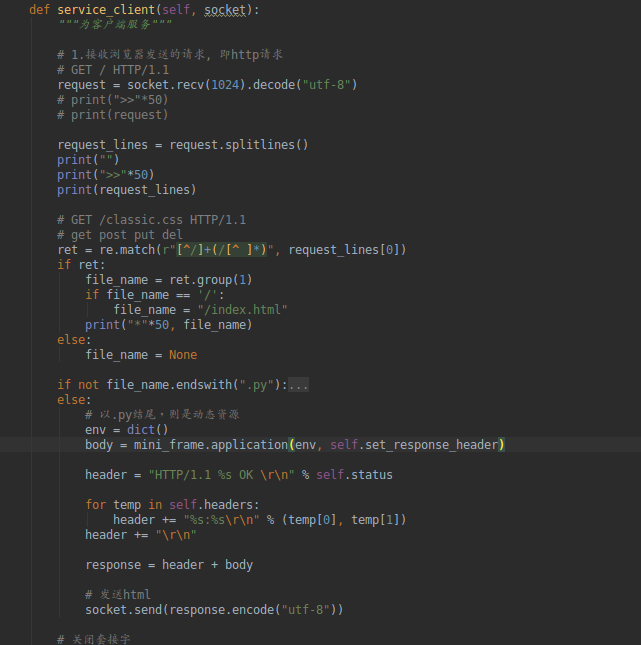
上面的application()函数就是符合WSGI标准的一个HTTP处理函数，它接收两个参数：

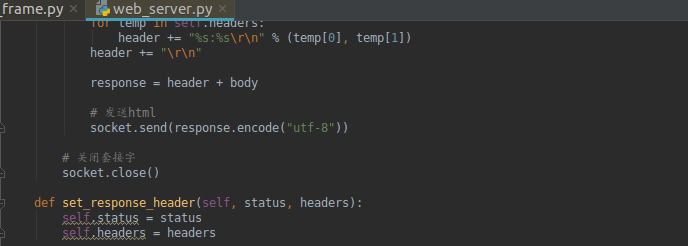
* environ：一个包含所有HTTP请求信息的dict对象；
* start\_response：一个发送HTTP响应的函数。

整个application()函数本身没有涉及到任何解析HTTP的部分，也就是说，把底层web服务器解析部分和应用程序逻辑部分进行了分离，这样开发者就可以专心做一个领域了

不过，等等，这个application()函数怎么调用？如果我们自己调用，两个参数environ和start\_response我们没法提供，返回的str也没法发给浏览器。

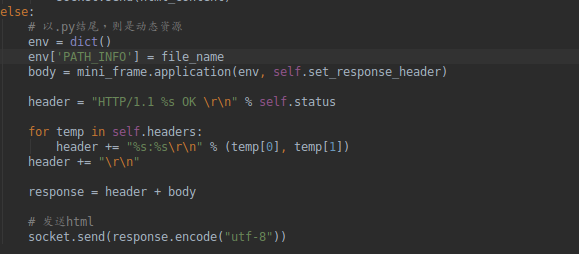
所以application()函数必须由WSGI服务器来调用。有很多符合WSGI规范的服务器。而我们此时的web服务器项目的目的就是做一个既能解析静态网页还可以解析动态网页的服务器

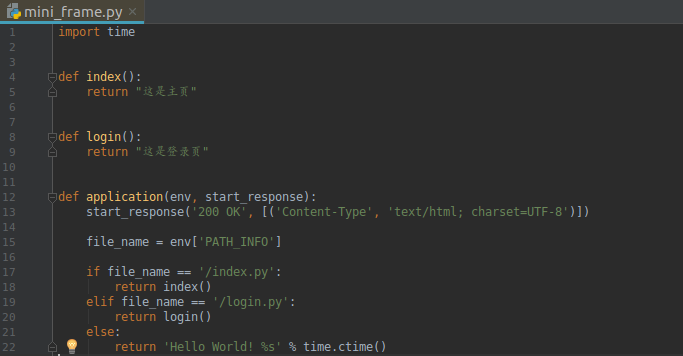




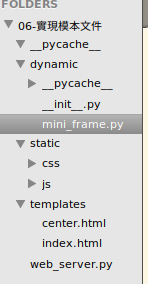


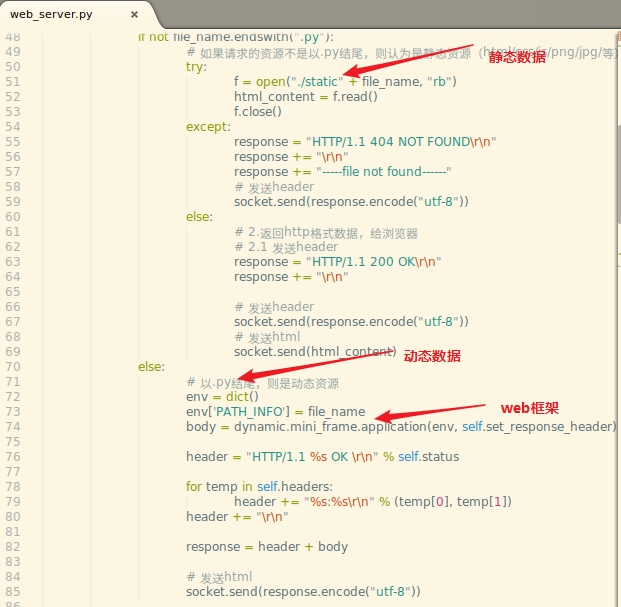
* 1. **设置传输字典**





* 1. **获取页面模板**

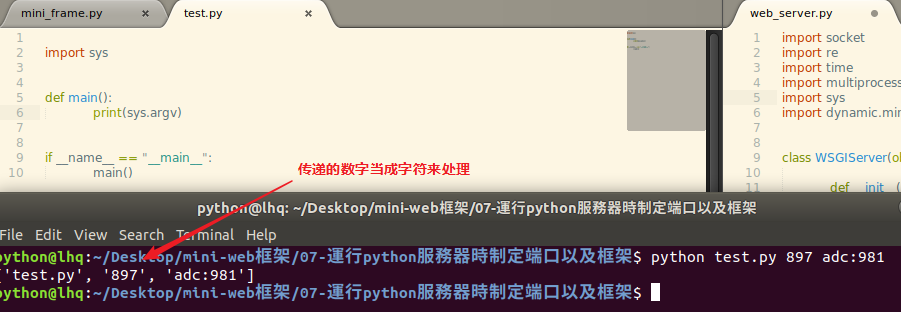




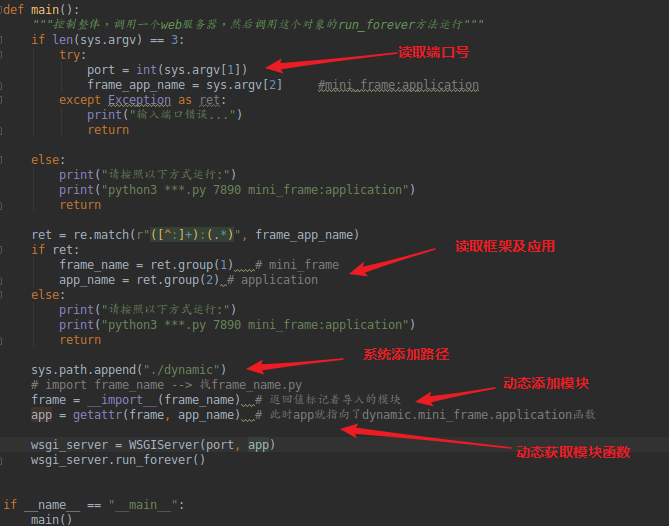


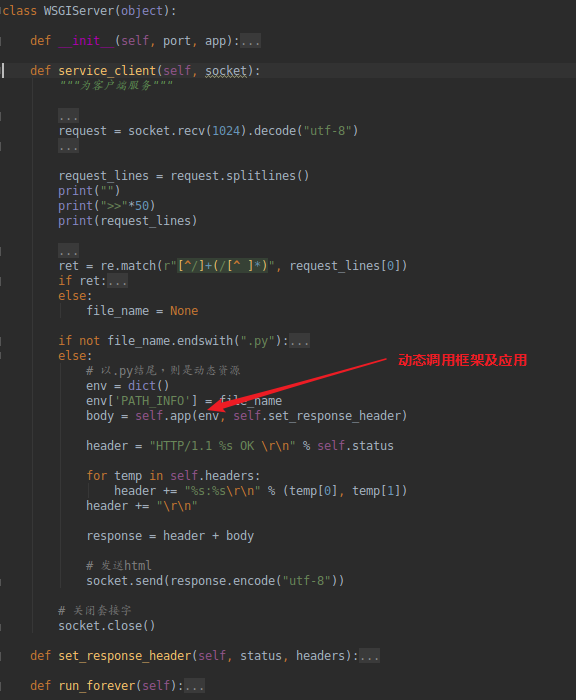
* 1. **传递参数**

**1.测试参数传递**



**2.添加web服务器的端口及应用**

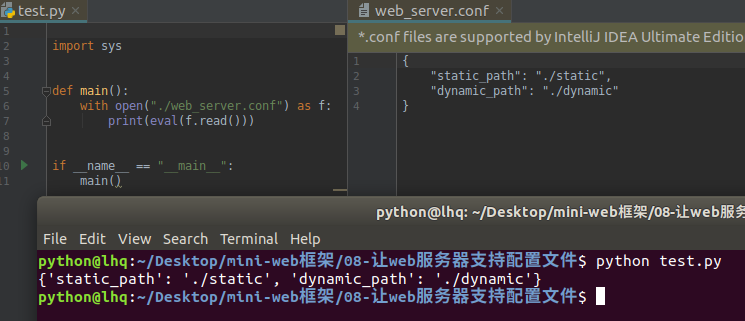




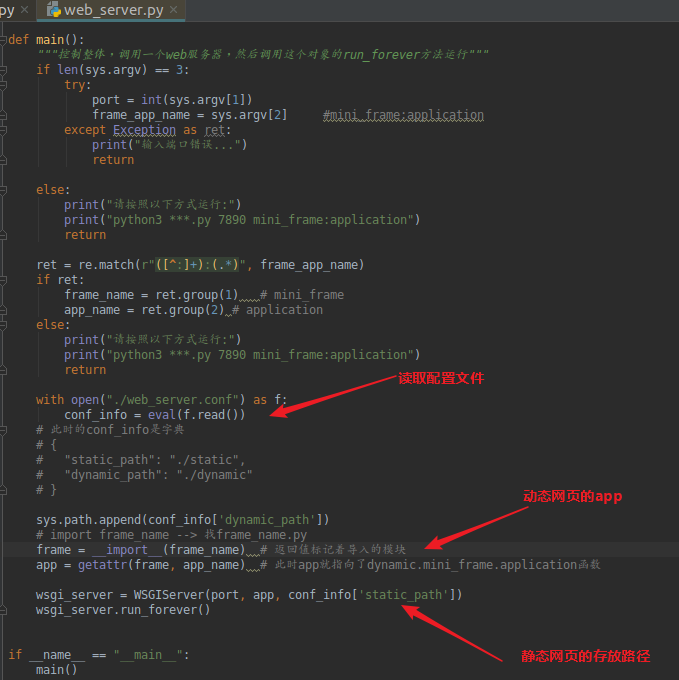


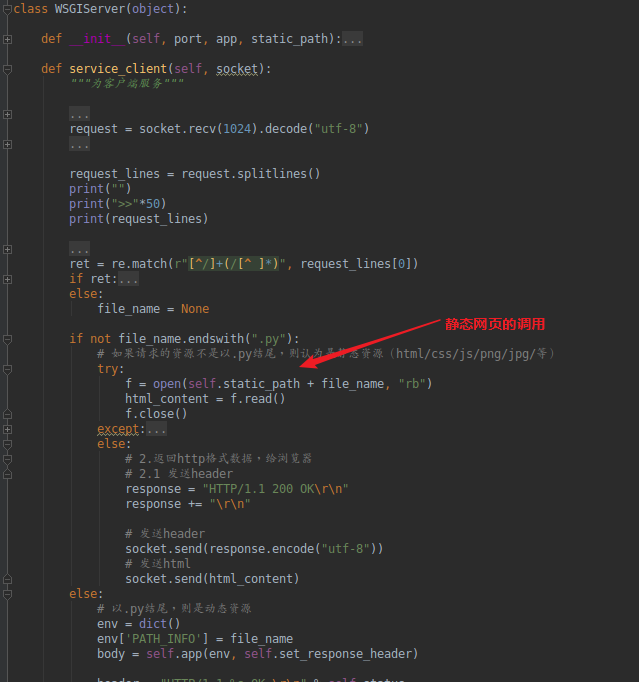
* 1. **给web服务器配置文件**

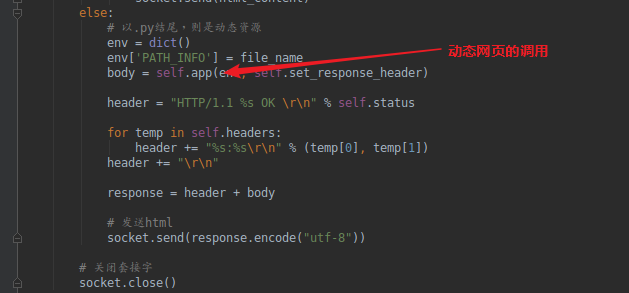
**1.测试配置文件的读取**



**2.添加web服务器的配置文件**







* 1. **添加shell功能**

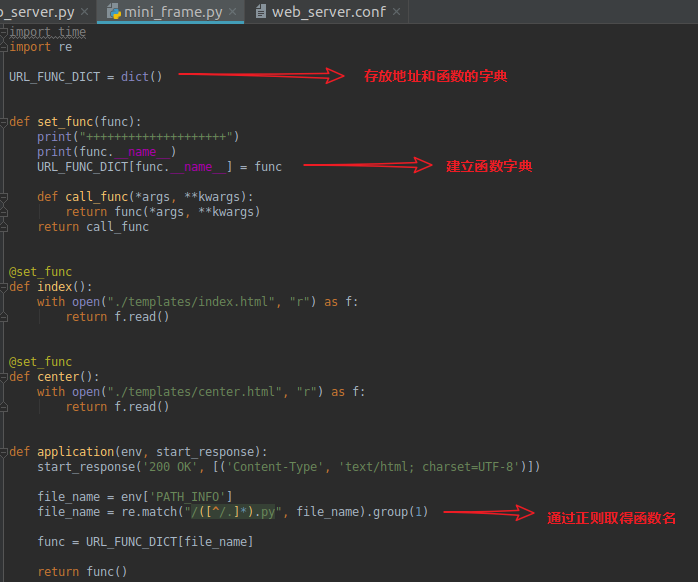


**二、mini-web框架添加路由和MySQL功能**

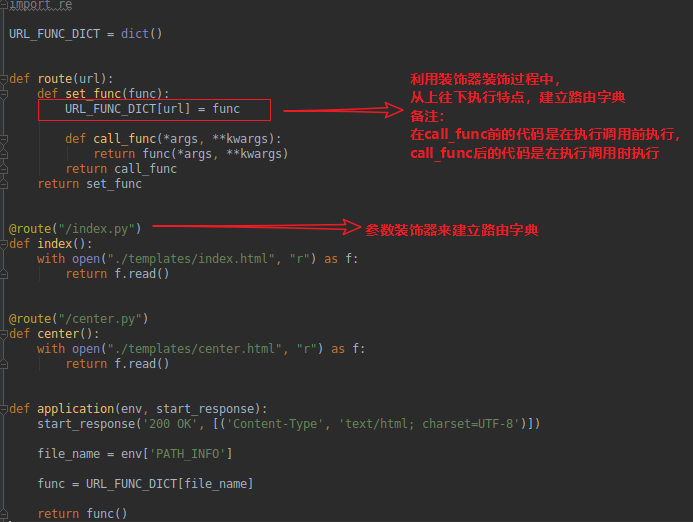
**2.1 用装饰器实现网址路由功能**

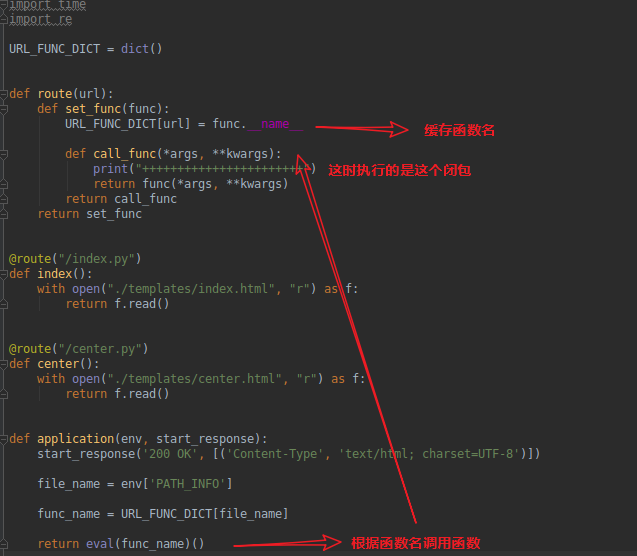
网络路由：根据网址不一样，调用的内容就不一样。

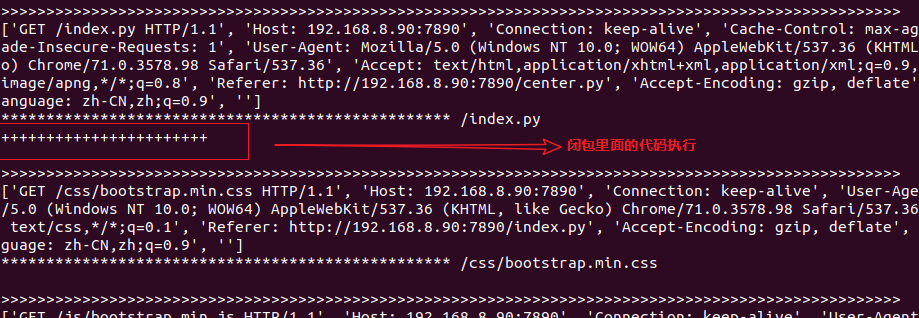
方案1：采用正则表达式解析网址



方案2：采用带参数的装饰器实现

  
理解：url\_func\_dict缓存的地址是直接函数的地址，没有经过第二层装饰器，调用时直接指向函数不会执行call\_func里面的代码；如果需要执行call\_func里面的代码，则需要如下使用，缓存函数名称，根据函数名称去指向闭包函数。





**2.2 伪静态、静态和动态的区别**

目前开发的网站其实真正意义上都是动态网站，只是URL上有些区别，一般URL分为静态URL、动态URL、伪静态URL，他们的区别是什么?

**2.2.1 静态URL**

静态URL类似 域名/news/2012-5-18/110.html 我们一般称为真静态URL，每个网页有真实的物理路径，也就是真实存在服务器里的。

* 优点是：

网站打开速度快，因为它不用进行运算；另外网址结构比较友好，利于记忆。

* 缺点是：

最大的缺点是如果是中大型网站，则产生的页面特别多，不好管理。至于有的开发者说占用硬盘空间大，我觉得这个可有忽略不计，占用不了多少空间的，况且目前硬盘空间都比较大。还有的开发者说会伤硬盘，这点也可以忽略不计。

* 一句话总结:

静态网站对SEO的影响：静态URL对SEO肯定有加分的影响，因为打开速度快，这个是本质。

SEO（Search Engine Optimization）：汉译为搜索引擎优化。 是一种方式：利用搜索引擎的规则提高网站在有关搜索引擎内的自然排名。

**2.2.2 动态URL**

动态URL类似 域名/NewsMore.asp?id=5 或者 域名/DaiKuan.php?id=17，带有？号的URL，我们一般称为动态网址，每个URL只是一个逻辑地址，并不是真实物理存在服务器硬盘里的。

* 优点是：

适合中大型网站，修改页面很方便，因为是逻辑地址，所以占用硬盘空间要比纯静态网站小。

* 缺点是：

因为要进行运算，所以打开速度稍慢，不过这个可有忽略不计，目前有服务器缓存技术可以解决速度问题。最大的缺点是URL结构稍稍复杂，不利于记忆。

* 一句话总结:

动态URL对SEO的影响：目前百度SE已经能够很好的理解动态URL，所以对SEO没有什么减分的影响（特别复杂的URL结构除外）。所以你无论选择动态还是静态其实都无所谓，看你选择的程序和需求了。

**2.2.3 伪静态URL**

伪静态URL类似 域名/course/74.html 这个URL和真静态URL类似。他是通过伪静态规则把动态URL伪装成静态网址。也是逻辑地址，不存在物理地址。

* 优点是：

URL比较友好，利于记忆。非常适合大中型网站，是个折中方案。

* 缺点是：

设置麻烦，服务器要支持重写规则，小企业网站或者玩不好的就不要折腾了。另外进行了伪静态网站访问速度并没有变快，因为实质上它会额外的进行运算解释，反正增加了服务器负担，速度反而变慢，不过现在的服务器都很强大，这种影响也可以忽略不计。还有可能会造成动态URL和静态URL都被搜索引擎收录，不过可以用robots禁止掉动态地址。

* 一句话总结:

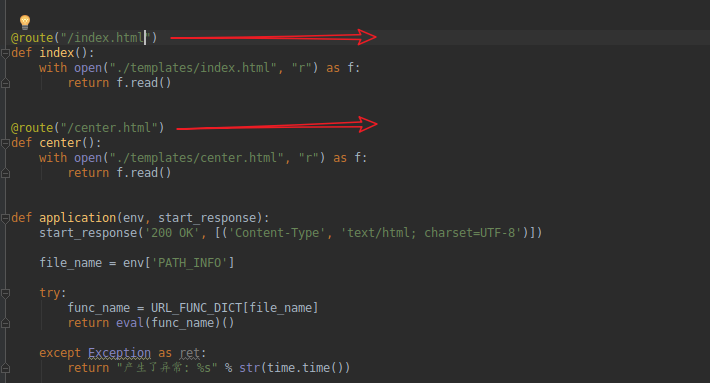
对SEO的影响：和动态URL一样，对SEO没有什么减分影响。

**2.3 mini-web框架-实现伪静态url**

修改web服务器：



修改应用：



修改模板中的超链接，将“.py”->“.html”

**2.4 准备数据**

**1. 创建数据库**

create database stock\_db charset=utf8;

**2. 选择数据库**

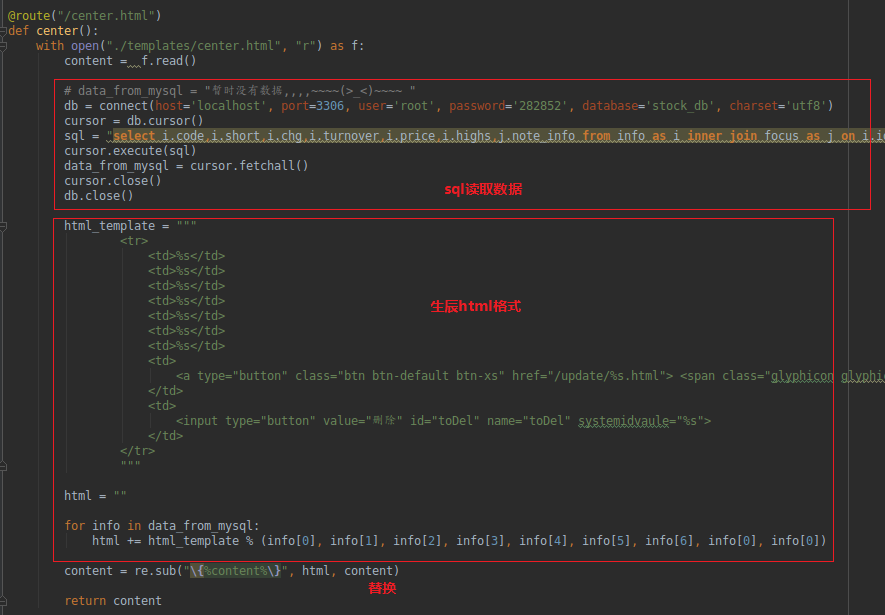
use stock\_db;

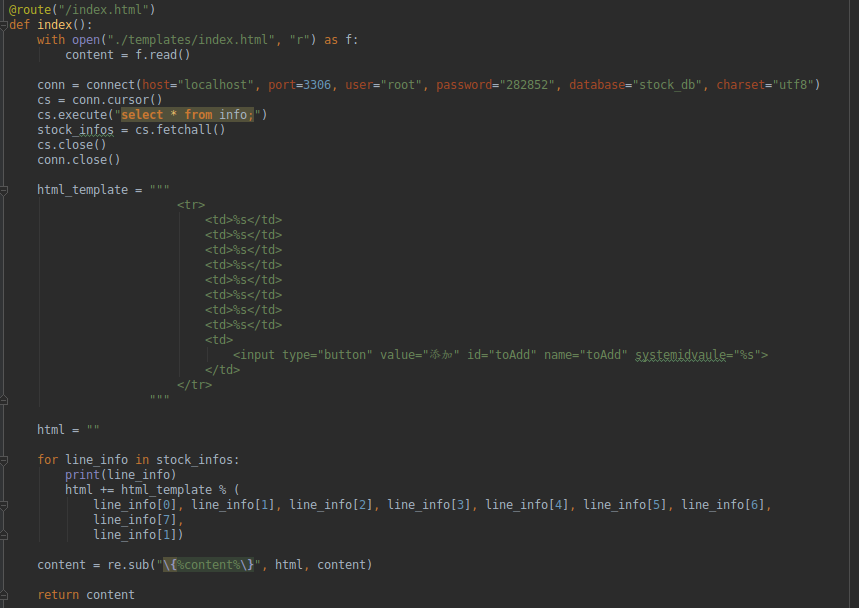
**3. 导入数据**

stock\_db.sql在课件中

source stock\_db.sql

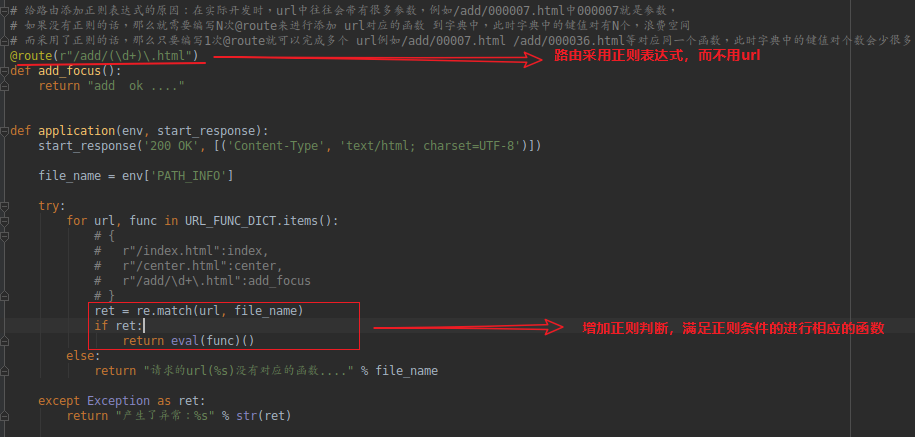
**2.5 mini-web框架-组装数据为html格式**





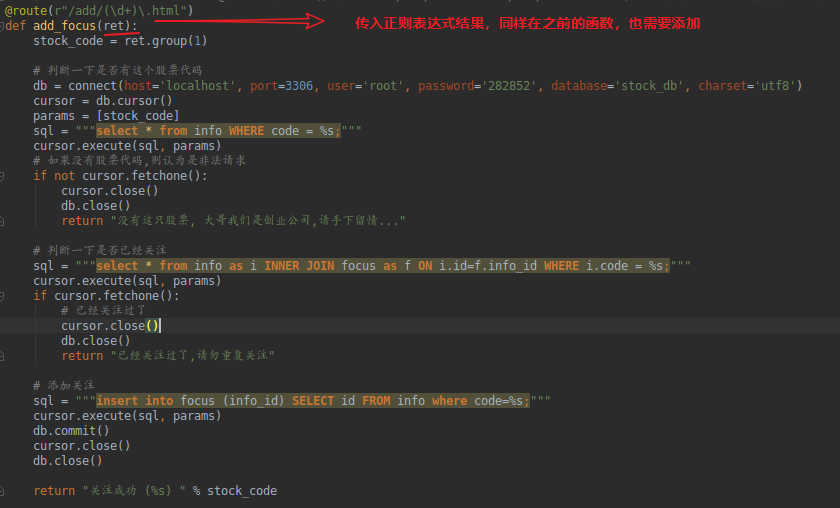
**三、mini-web框架添加log和路由正则**

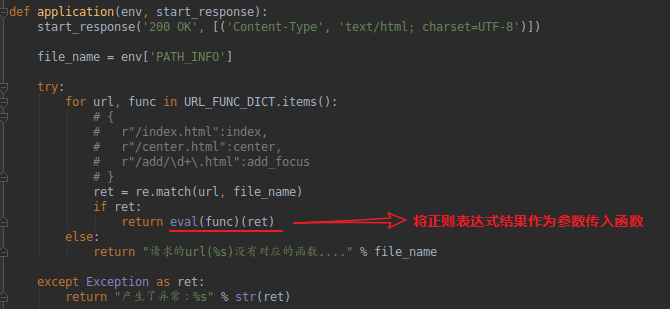
**3.1 路由支持正则**

需要将课件里面的模板更新到项目中，更新index.html、center.html、update.html

**3.2 mini-web框架-mysql-增加**

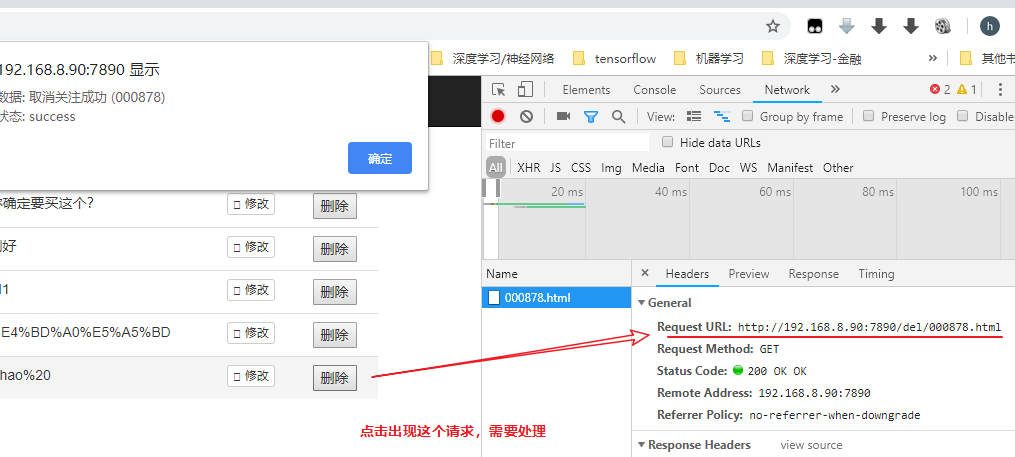


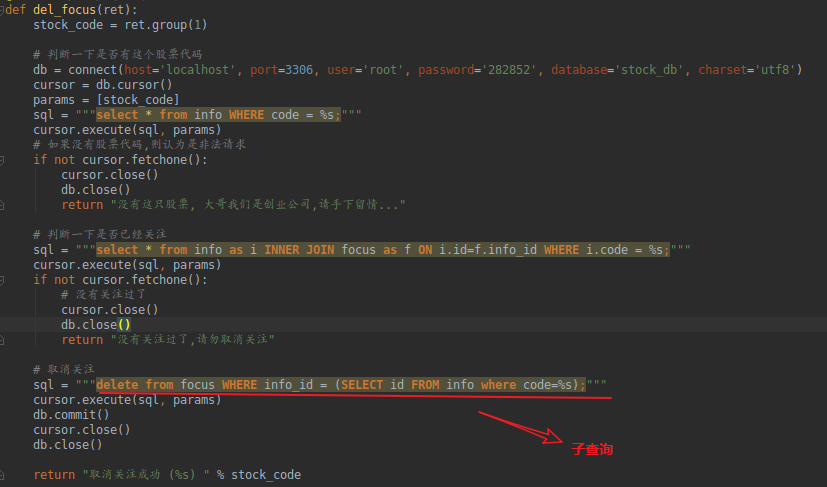




**3.3 mini-web框架-mysql-删除**

同添加类似

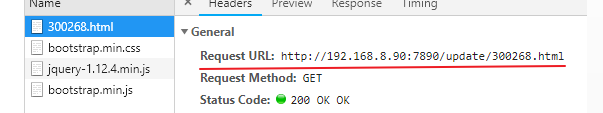


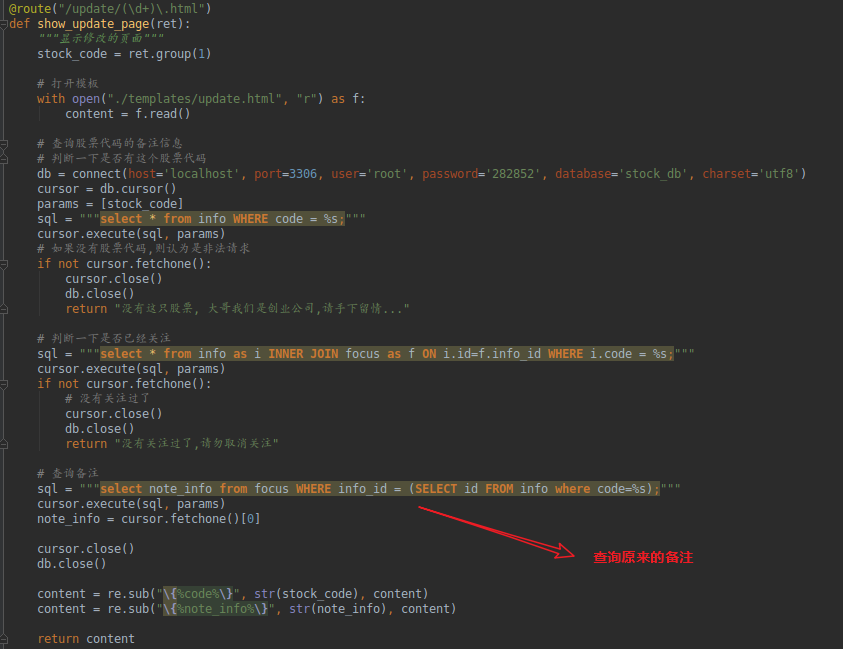


**3.4 mini-web框架-mysql-修改**

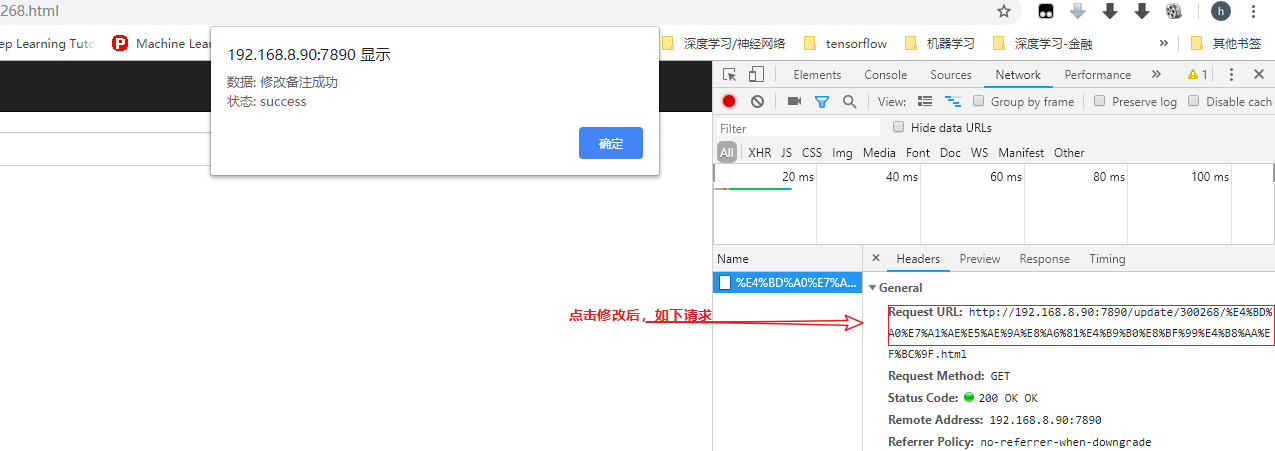
显示新的修改页面：

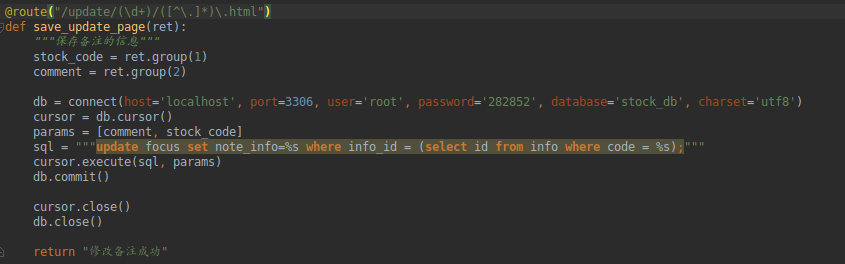
点击修改出现请求如下：





提交修改备注信息：

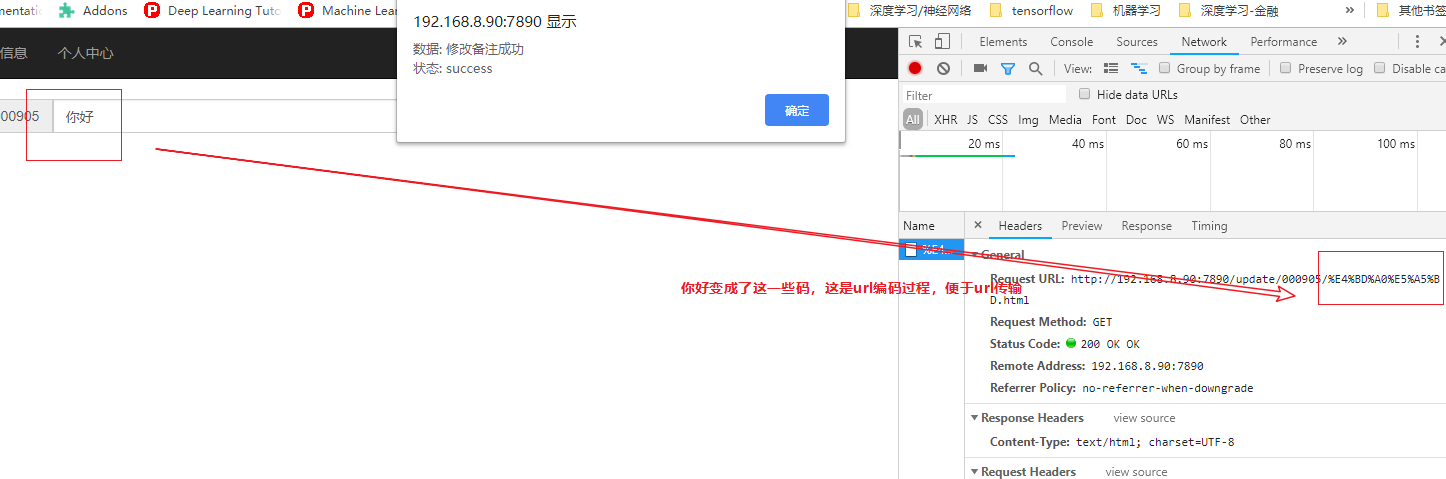




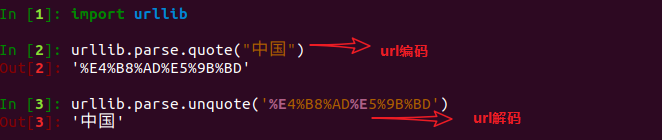
但是会出现以下乱码：

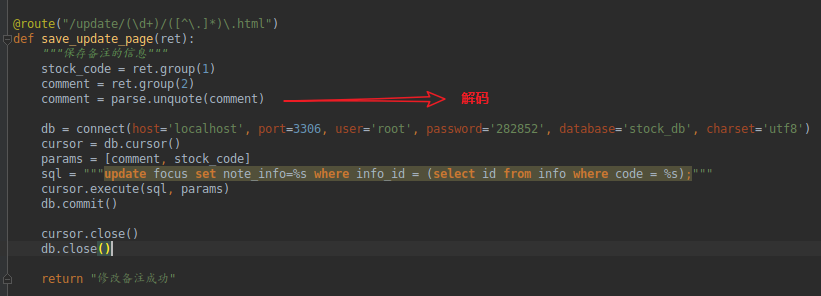


**3.5 mini-web框架-url编码**

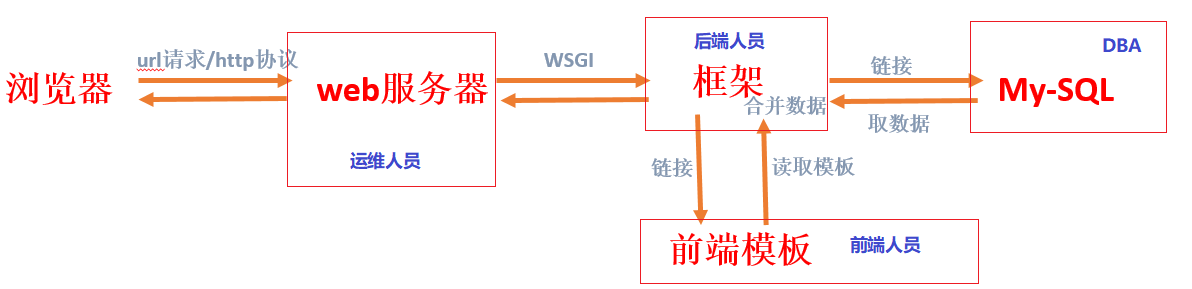


url对“你好”进行了编码，将编码传输给框架(即处理函数save\_update\_page)，如果直接存储则是这一推编码，需要在函数里面进行解码。





**3.6 web开发过程及结构**



**3.7 logging日志模块**

Python中有个logging模块可以完成相关信息的记录，在debug时用它往往事半功倍

**3.7.1. 日志级别**

日志一共分成5个等级，从低到高分别是：

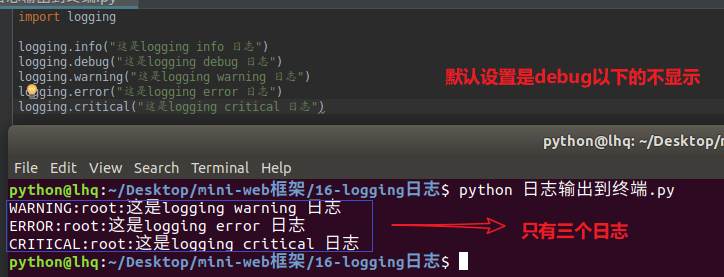
1. DEBUG
2. INFO
3. WARNING
4. ERROR
5. CRITICAL

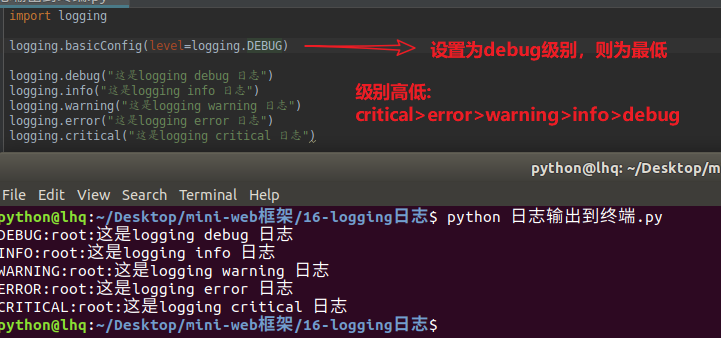
说明:

* DEBUG：详细的信息,通常只出现在诊断问题上
* INFO：确认一切按预期运行
* WARNING：一个迹象表明,一些意想不到的事情发生了,或表明一些问题在不久的将来(例如。磁盘空间低”)。这个软件还能按预期工作。
* ERROR：更严重的问题,软件没能执行一些功能
* CRITICAL：一个严重的错误,这表明程序本身可能无法继续运行

这5个等级，也分别对应5种打日志的方法： debug 、info 、warning 、error 、critical。默认的是WARNING，当在WARNING或之上时才被跟踪。

**3.7.2 将日志输出到控制台（print）**



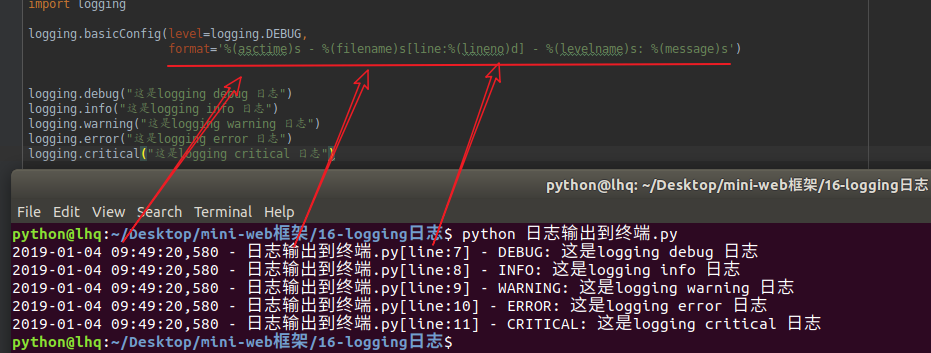


通过logging.basicConfig函数对日志的输出格式及方式做相关配置，上面代码设置日志的输出等级是DEBUG级别，意思是DEBUG级别以上的日志才会输出。

logging.basicConfig(level=logging.WARNING,

format='%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s')

**一般加上这一行，表示时间、文件名、行数，方便快速查找问题代码位置。**



注意，只要用过一次log功能再次设置格式时将失效，实际开发中格式肯定不会经常变化，所以刚开始时需要设定好格式

**3.7.3 将日志输出到文件**

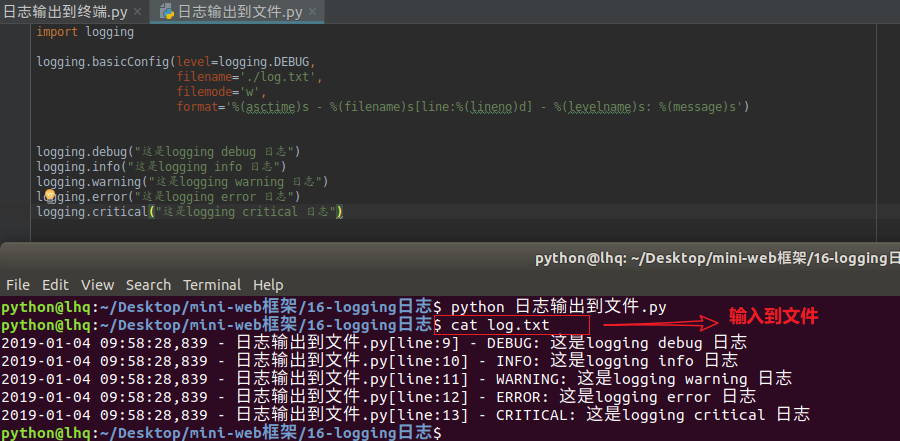
logging.basicConfig(level=logging.WARNING,

**filename='./log.txt',**

**filemode='w',**

format='%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s')

备注：这里的filemode可以是“a”（添加模式），“w”（覆盖模式）



**3.7.4 将日志输出到文件和终端**

import logging

# 第一步，创建一个logger

logger = logging.getLogger()

logger.setLevel(logging.INFO) # Log等级总开关

# 第二步，创建一个handler，用于写入日志文件

logfile = './log.txt'

fh = logging.FileHandler(logfile, mode='a') # open的打开模式这里可以进行参考

fh.setLevel(logging.DEBUG) # 输出到file的log等级的开关

# 第三步，再创建一个handler，用于输出到控制台

ch = logging.StreamHandler()

ch.setLevel(logging.WARNING) # 输出到console的log等级的开关

# 第四步，定义handler的输出格式

formatter = logging.Formatter("%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s")

fh.setFormatter(formatter)

ch.setFormatter(formatter)

# 第五步，将logger添加到handler里面

logger.addHandler(fh)

logger.addHandler(ch)

# 日志

logger.debug('这是 logger debug message')

logger.info('这是 logger info message')

logger.warning('这是 logger warning message')

logger.error('这是 logger error message')

logger.critical('这是 logger critical message')

**3.7.5 日志格式说明**

logging.basicConfig函数中，可以指定日志的输出格式format，这个参数可以输出很多有用的信息，如下:

* %(levelno)s: 打印日志级别的数值
* %(levelname)s: 打印日志级别名称
* %(pathname)s: 打印当前执行程序的路径，其实就是sys.argv[0]
* %(filename)s: 打印当前执行程序名
* %(funcName)s: 打印日志的当前函数
* %(lineno)d: 打印日志的当前行号
* %(asctime)s: 打印日志的时间
* %(thread)d: 打印线程ID
* %(threadName)s: 打印线程名称
* %(process)d: 打印进程ID
* %(message)s: 打印日志信息

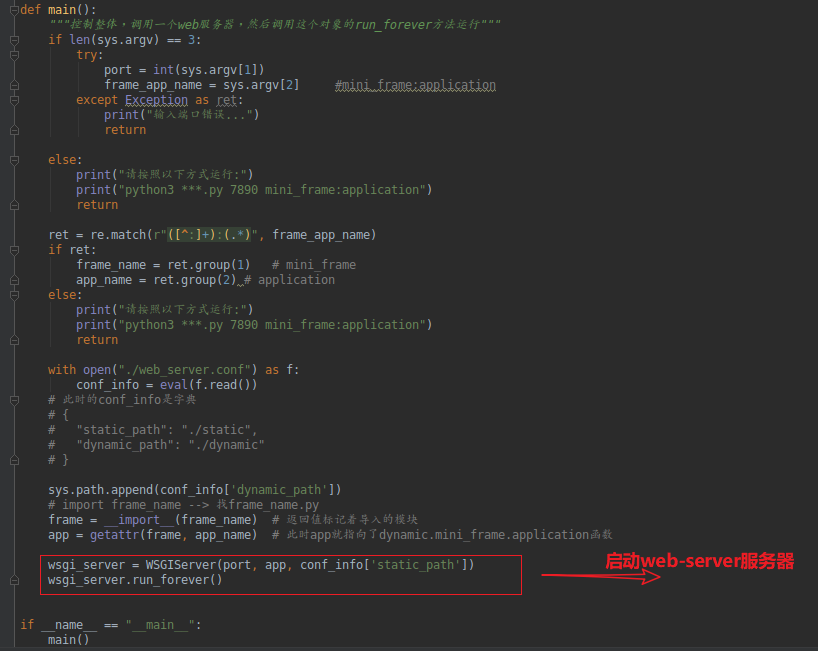
在工作中给的常用格式如下:

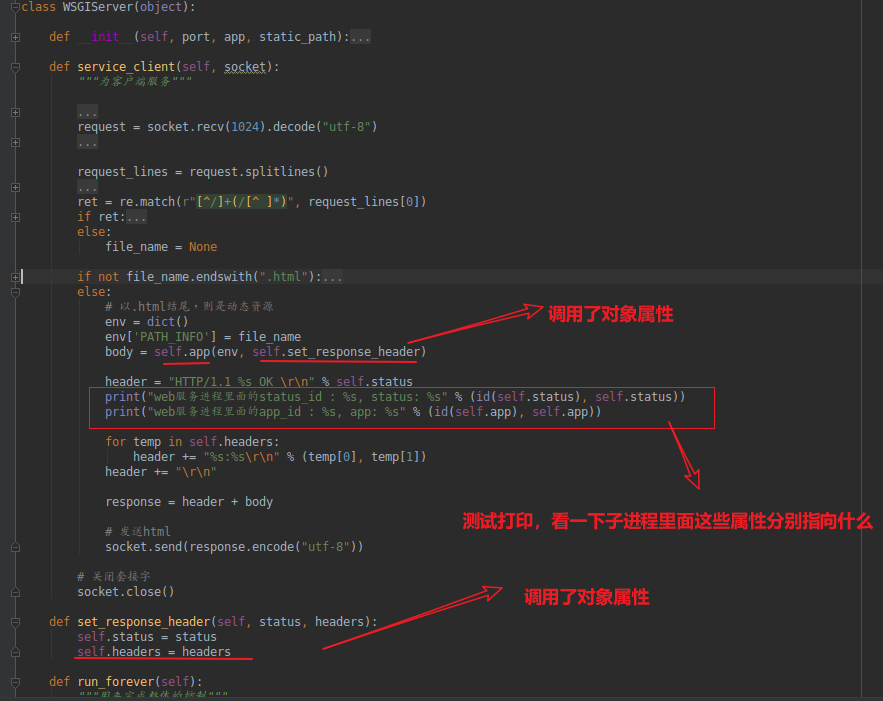
format='%(asctime)s - %(filename)s[line:%(lineno)d] - %(levelname)s: %(message)s'

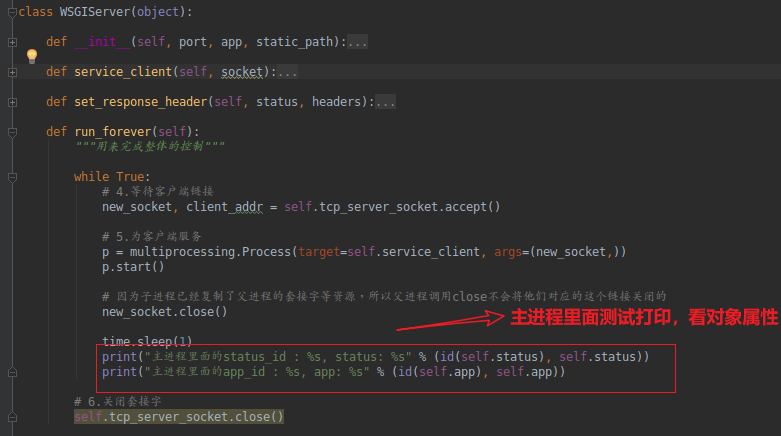
这个格式可以输出日志的打印时间，是哪个模块输出的，输出的日志级别是什么，以及输入的日志内容。

**四、再议多进程**

之前的代码中用到了mini-web-server，服务器中采用的多进程进行web服务。但是建立多进程时，传输的是service\_client函数和socket，没有传输web-server对象中的属性。但是实际服务中，调用了对象的属性，那多进程是怎么调用的？为什么没有向进程传输参数就能调用对象的属性？









分析：

从以上结果来看，在对象里面进行多进程创建，对象的属性进行了内容复制，指向不一致；对象的方法只是进行了地址复制，指向一致。因此，应用时，对象属性应该在子进程里面重新赋值，而对象方法可以直接调用，则指向对象方法地址。