import asyncio  
  
from aiohttp import web  
  
async def index(request):  
 await asyncio.sleep(0.5)  
 return web.Response(body=b'<h1>Index</h1>')  
  
async def hello(request):  
 await asyncio.sleep(0.5)  
 text = '<h1>hello, %s!</h1>' % request.match\_info['name']  
 return web.Response(body=text.encode('utf-8'))  
  
async def init(loop):  
 app = web.Application(loop=loop)  
 app.router.add\_route('GET', '/', index)  
 app.router.add\_route('GET', '/hello/{name}', hello)  
 srv = await loop.create\_server(app.make\_handler(), '127.0.0.1', 8000)  
 print('Server started at http://127.0.0.1:8000...')  
 return srv  
  
loop = asyncio.get\_event\_loop()  
loop.run\_until\_complete(init(loop))  
loop.run\_forever()

async def index(request):  
 await asyncio.sleep(0.5)  
 return web.Response(body=b'<h1>Index</h1>')

1.参数：aiohttp.web.request实例

aiohttp.web.request实例，包含了所有浏览器发送过来的 HTTP 协议里面的信息，以及服务器自身信息

2.返回值：aiohttp.web.Response实例

继承自StreamResponse，功能为构造一个HTTP响应类声明 class aiohttp.web.Response(\*, status=200, headers=None, content\_type=None, body=None, text=None)

app = web.Application(loop=loop)

1. 创建application: aiohttp.web.Application类的实例

class Application(MutableMapping):

def \_\_init\_\_(self, \*,  
 logger=web\_logger,  
 router=None,  
 middlewares=(),  
 handler\_args=None,  
 client\_max\_size=1024\*\*2,  
 loop=None,  
 debug=...)

if router is None:  
 router = UrlDispatcher()

app.router.add\_route('GET', '/', index)

1. 将处理函数注册到app.router中

def add\_route(self, method, path, handler,  
 \*, name=None, expect\_handler=None):  
 resource = self.add\_resource(path, name=name)  
 return resource.add\_route(method, handler,  
 expect\_handler=expect\_handler)

UrlDispatcher类中有方法add\_route(method, path, handler, \*, name=None, expect\_handler=None)，该方法将处理函数（其参数名为handler）与对应的URL（HTTP方法metho，URL路径path）绑定

def add\_resource(self, path, \*, name=None):  
 if path and not path.startswith('/'):  
 raise ValueError("path should be started with / or be empty")  
 # Reuse last added resource if path and name are the same  
 if self.\_resources:  
 resource = self.\_resources[-1]  
 if resource.name == name and resource.raw\_match(path):  
 return resource  
 if not ('{' in path or '}' in path or ROUTE\_RE.search(path)):

//这里的url是一个URL类的实例，url为’/’   
 url = URL.build(path=path)

resource = PlainResource(url.raw\_path, name=name)

//这里是将resource加入到UrlDipatcher实例的一个list去，  
 self.register\_resource(resource)

return resource  
 resource = DynamicResource(path, name=name)  
 self.register\_resource(resource)  
 return resource

对于path=‘/’,会生成一个PlainRescource对象，并且把这个对象注册到UrlDispatcher类的对象app.router里。先进入PlainResource类里看看，发现了一条继承链

PlainResource继承自Resource继承自AbstractResource

class UrlDispatcher

def register\_resource(self, resource):

name = resource.name

if name is not None:

name = resource.name

………………

self.\_named\_resources[name] = resource

self.\_resources.append(resource)

\_resources是UrlDipatcher实例的一个list，

resource实际上是得到一个self.\_resources里一个刚构建的PlainResource的对象，然后对这个对象执行add\_route方法，进入PlainResource类里查看这个add\_route方法，但是PlainResource没有这个方法，于是到父类Resource里找，找到了这个add\_route方法：

def add\_route(self, method, handler, \*,  
 expect\_handler=None):  
 //self是PlainResource, method:GET handler是index函数  
 for route\_obj in self.\_routes:

//由于PlainResource是新创建的,self.\_routes是空list，会略过这一步   
 if route\_obj.method == method or route\_obj.method == hdrs.METH\_ANY:  
 raise RuntimeError("Added route will never be executed, "  
 "method {route.method} is already "  
 "registered".format(route=route\_obj))  
  
 route\_obj = ResourceRoute(method, handler, self,  
 expect\_handler=expect\_handler)  
 self.register\_route(route\_obj)  
 return route\_obj

route\_obj = ResourceRoute(method, handler, self,expect\_handler=expect\_handler)

这里的self是PlainResource类实例，

def \_\_init\_\_(self, method, handler, \*,  
 expect\_handler=None,  
 resource=None):  
 method = method.upper()  
 if asyncio.iscoroutinefunction(handler):  
 pass  
 elif inspect.isgeneratorfunction(handler):  
 warnings.warn("Bare generators are deprecated, "  
 "use @coroutine wrapper", DeprecationWarning)  
 elif (isinstance(handler, type) and  
 issubclass(handler, AbstractView)):  
 pass  
 else:  
 warnings.warn("Bare functions are deprecated, "  
 "use async ones", DeprecationWarning)  
 #如果添加的方法既不是生成器，也不是协程。则aiohttp帮我们写一个处理函数的协程版本  
 @wraps(handler)  
 async def handler\_wrapper(\*args, \*\*kwargs):  
 result = old\_handler(\*args, \*\*kwargs)  
 if asyncio.iscoroutine(result):  
 result = await result  
 return result

#此时handler被赋值给old\_handler，即old\_handler = index

old\_handler = handler

handler = handler\_wrapper  
  
 self.\_method = method  
 self.\_handler = handler  
 self.\_expect\_handler = expect\_handler  
 self.\_resource = resource

app.router.add\_route里的handler，最终成了ResourceRoute实例的\_handler属性

这里的self是PlainResource的实例，最后将ResourceRoute实例加入到了\_routes这个list里，

def register\_route(self, route):  
 assert isinstance(route, ResourceRoute), \  
 'Instance of Route class is required, got {!r}'.format(route)  
 self.\_routes.append(route)