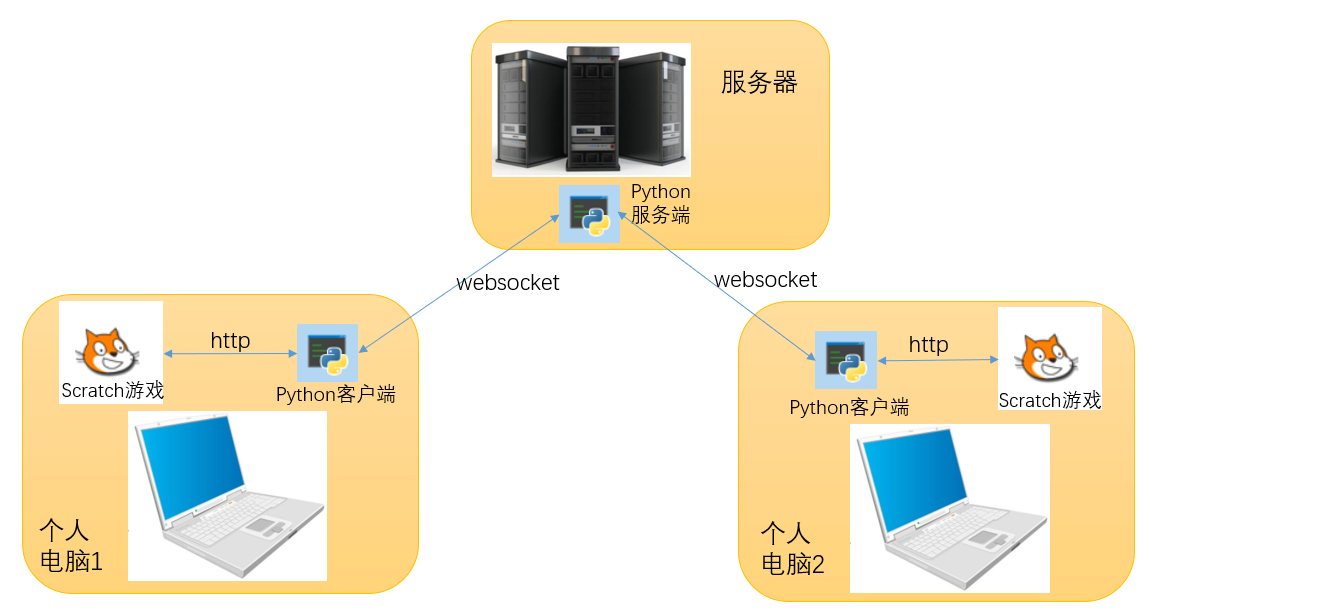
Scratch2有一个扩展功能，可以让Scratch和第三方应用通过http连接起来，实现访问网站数据、控制硬件等应用。我们今天就使用这个扩展，连接到一个用Python实现的网络服务，来做一个联网游戏。希望大家能通过今天的学习了解一些网络编程的基本知识，以及知道要做出一个比较复杂的系统要关心的地方。

我们先用Scratch做一个大富翁游戏：有两个以上的游戏角色（这个例子里是一只猫和一只狗）分别由两台电脑控制。角色轮流滚动色子走动，走动的步数由骰子的点数随机决定。走动途中会碰到不同的魔术指令可以改变走动的路径、加分等等。谁先走到终点就算赢。游戏画面如下所示：



**系统结构图**

有两个玩家情况下的系统架构图如下所示：



我们这里说的系统架构图是一个计算机系统的设计图。描述的对象包括直接构成系统的各个硬件和软件组件，以及各个组件之间的连接和通讯方法。

系统架构图是构建计算机系统实践的基础，与建筑师在开始建设前必须完成的设计图纸一样重要。现代的大型计算机系统就如同一个复杂庞大的建筑，完成它需要多方面的计算机网络、硬件和软件配合以及涉及大量的知识。如果正式搭建之前没有想好系统架构和设计细节，往往会导致事倍功半，甚至在项目后期才发现无法解决的问题而不得不推倒重来。

大家编程的过程中如果涉及多个组件（或对象），特别是如果涉及网络的编程，在开始写程序之前请务必通过画出系统架构图来理清各组件之间的关系。

架构图等计算机设计图表是用来辅助我们理解、维护计算机系统以及方便我们和其他部门沟通交流的，对于图表的格式，不同的单位可能会有不同的要求，总的来说只要方便相关的人员理解就行。比较著名的一种格式叫做统一建模语言（Unified Modeling Language, UML），有时间的可以进一步了解。

上面的架构图说明了在有两个玩家的情况下，网络里会有：

3个硬件：

* 一台服务器
* 两台个人电脑

两台个人电脑实际上是可以直接连接起来实现联网对战的。为什么需要一台服务器那么复杂呢？原因是：如果其中一个玩家没开电脑，没有服务器的话，另一个玩家需要找一个新玩家的话就不是那么容易了。就算找到了新玩家，两台电脑之间还需要重新设置网络参数以实现联网。这对于没有多少电脑知识只想简简单单玩个游戏的人来说非常困难。而服务器一般都是7\*24小时稳定提供服务的，在游戏开发时设置好服务器地址后，任意一个玩家打开游戏就可以连上同一个服务器，再通过服务器寻找其他在线的玩家一起玩了。

5个软件组件：

* 服务器里的Python服务端 （实现联网游戏的协作和数据共享）
* 每台电脑上的Python客户端 （负责和服务器连接，是服务器和Scratch游戏之间沟通的桥梁）
* 每台电脑上的Scratch游戏端 （负责展示游戏画面）

上面的架构图还说明了不同部件之间的连接协议。这里的协议可以理解为部件之间沟通的一种语言。Scratch游戏不能直接和Python语言写的客户端沟通；Python语言写的客户端和服务端分布在网络的两端，它们之间的沟通也不像在同一台电脑上沟通那么简单（比如不能直接访问文件），这就需要根据需求选择沟通协议。

Scratch的扩展规定了只能使用http协议，所以第三方应用必须要使用http协议才能和Scratch游戏沟通。用Python可以很容易地写一个支持http协议的应用。

Python客户端和服务端可以选择的连接协议就非常之多了。我们没有选择最方便的http协议是因为http协议只能单向请求，即只能等客户端向服务端发起请求之后服务端才能给客户端发送信息，服务端不能主动随时向客户端发送信息。我们的联网游戏需要频繁地双向沟通，这种协议并不是太适合。

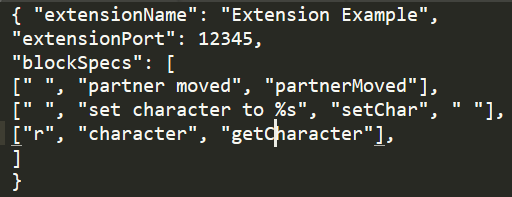
我们选择了websocket作为python客户端和服务端的沟通协议。Websocket的连接方式和http一样，只需要服务端定义好连接接口，不需要在客户端进行额外的网络设置。连接后客户端和服务端之间可以随时互相发送消息了。

**Scratch扩展的沟通协议**

Scratch扩展的沟通协议有专门的定义。

首先，需要建立一个“.s2e”描述文件来定义扩展积木。该文件是json格式的。然后把该文件导入到Scratch里面生成扩展积木。导入方法是在scratch里按住键盘“Shift”键然后点击“文件”，然后选择“导入实验性HTTP扩展功能”，接着选择你的“.s2e”文件就可以了。导入成功后在“更多积木”那里就能看到自定义的积木了。

下面是一个“.s2e”文件例子：



导入后的积木是这样的：



第一行："Extension Example"就是显示的扩展积木模块名称。

第二行：12345是http客户端的端号。这个扩展积木会访问本地电脑的12345 http端口来通信，即<http://localhost:12345/>

第三行开始定义不同的积木。

第四行定义了一个执行命令积木，它在scratch里面的显示名称是“partner moved”,当刚积木被执行的时候会发送一个http请求到<http://localhost:12345/partnerMoved> 地址里。最后一个参数就是该请求的地址。

第五行定义了一个设置命令参数，它会把“set character to\_”里的“\_”传输出去。比如执行了“set character to 3”积木就会发一个http请求到<http://localhost:12345/setChar/3> 地址去。

第六行是一个从http服务端取得当前参数返回值得命令。第一个参数“r”代表他是一个“报告”指令，它可以取得“getCharacter”的值。注意这里取值并不是直接发一条getCharacter请求到http服务端，而是从最近的一次<http://localhost:12345/poll> 请求里取出getCharacter指令的值。关于“/poll”请求请看下面详细说明：

**/poll 请求**

Scratch每秒钟大概发30条“/poll”请求到http服务端，并从“/poll”请求的回应报文那里获取最新的参数值。http服务端要通知Scratch的任何信息或结果都需要通过“/poll”回应报文来传送。每一对返回参数和返回值占一行，两对之间通过“/n”行结束符分隔。返回参数和返回值之间要有一个空格。如下面的回应报文：

getCharacter abc

age 12

它返回两个参数：getCharacter的值是abc，age的值是12。getCharacter的值可以通过上面的第六行定义的“”积木获取。

**/reset\_all 请求**

在Scratch里点击结束的红球时会触发发一个“/reset\_all”请求到http服务端。可以把一些游戏重置或停止时需要做的东西放到这个请求的处理模块里。但是我的试验里有时候点绿旗也会触发“/reset\_all”请求，但并不是每次都会。所以如果需要在游戏没有结束（不点击红球）时做一些重置动作的话，保险起见最好自己定义一个“reset”积木。

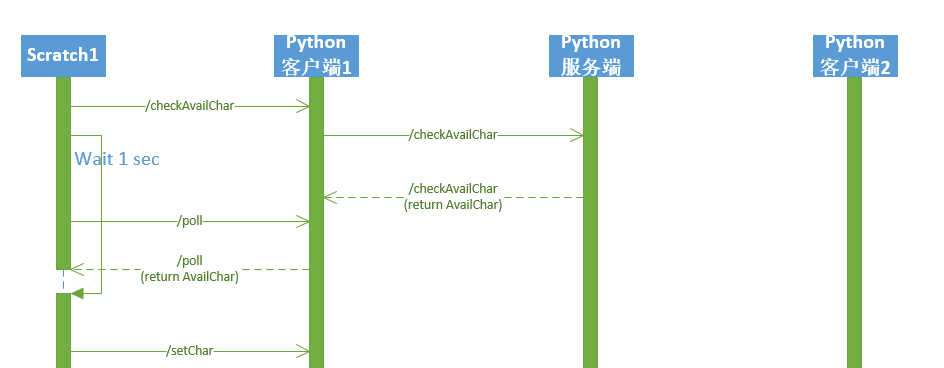
Scratch扩展的一个关键点或者说限制条件是：**Scratch只能通过poll请求从http服务端获取信息，而不能通过其他请求或者指令。换一种说法，http服务端如果想要发消息给Scratch端，只能把消息内容放在poll请求的回复里面。**

我们要做的联网游戏需要用到的Scratch扩展沟通协议知识就是上面这些。接下来介绍一些联网游戏的关键部分。不同模块之间的沟通次序。我们用一种叫“顺序图”的图表来描述这种关系。

**顺序图**

顺序图一般用于说明一个系统之间如何交互来实现一个使用场景，它体现的是系统不同组件之间如何按照时间顺序互相配合来完成一个任务。它是一个二维图，纵向是时间轴，时间沿竖线向下延伸。横向代表了协作中各个组件。

首先是一个查看可用角色的顺序图：



“可用角色”这一信息保存在服务器端，所以每个游戏的界面（Scratch端）都要通过连接Python客户端在连到服务器端去取得这个信息。顺序图的顶端是3个组件的名称。每个组件都有一条纵向的生命线。生命线间用有方向的连线表示不同组件的协作。实线表示一个组件向另一个组件发起一个请求，箭头方向表示请求的方向；虚线表示请求的结果返回。从上到下表示时间顺序。上图的意思是以下步骤按时间先后顺序进行：

1. Scratch游戏程序向Python客户端（在同一台电脑上通过http协议）发起一个checkAvailChar(检查可用角色)的请求。
2. Scratch游戏程序执行等待1秒钟的指令。我们这里预期1秒内从网络中得到可用角色的返回值。
3. Python客户端（在网络上通过websocket协议）向服务器端发起一个checkAvailChar(检查可用角色)的请求。这一步骤是和步骤2同步进行的，我们没办法控制2和3的先后次序。
4. Python服务器端向客户端返回步骤3的请求结果 - 可用角色。
5. Scratch游戏程序向Python客户端发出poll请求。上面介绍过， 事实上Scratch游戏程序大概每秒钟发出30条poll请求。只不过 Python客户端只有在服务器端返回可用角色信息后才会把正确的值通过poll返回给Scratch游戏，在这之前Scratch游戏都只能取得一个不正确的值。
6. Python客户端向Scratch端返回步骤5的请求结果 - 可用角色。只有客户端已经从服务器端取得正确的可用角色值后，它才能向Scratch端返回正确的结果。否则就返回默认值。
7. Scratch游戏程序向Python客户端发起一个setChar(设置角色)的请求。这个步骤是在‘步骤2 – 等待1秒’之后才会进行的。并且图里省略了Scratch游戏让玩家选择角色的步骤，因为这个流程图重点在于描述不同系统组件间的协作。

上面是一个简单的顺序图介绍。顺序图对于编程人员非常重要，一个复杂的涉及多个组件的系统直接看程序的话通常很难理解。我写这个小联网游戏的过程中就有感觉，如果不看顺序图的话，经常是昨天写的程序今天已经看得非常吃力了。同一个作者尚且如此，如果是不同部门、不同领域用不同语言写的程序就看起来就更加是天书一样了。顺序图就提供了一种通用、简单、直白的语言来描述不同组件之间的联系。这也是UML的核心作用或者说目标 – 提供一种统一的设计沟通语言。

画顺序图还有一个好处就是它让人可以更早地在写程序之前真正深入地理清核心系统的逻辑关系，可以尽早地发现一些比较严重的逻辑错误。计算机系统开发有一个重要规律是问题越早发现修改起来的需要的资源（费用）就越低。一个简单的例子：一个设计错误如果在测试阶段才发现的话往往要花费很多部门（设计、开发、测试）的精力才能修复；如果设计阶段已经发现了就没有开发、测试部门的重复劳动了。

顺序图可以按不同的功能分开来画，比如上面的图只包括了检查可以角色的功能。并且它不需要包括系统的所有逻辑顺序，一些显而易见的只是系统一个小组件内部的步骤就不需要画出来了。当然这个要靠团队特别是设计师的经验和能力来判断了。对于初学者，所有涉及不同组件（比如分布在不同电脑的程序、或者使用不同语言写的程序）的交互部分应该都画出顺序图来。

画顺序图包括其他设计图的大原则是方便交流和理解系统就行，并不需要花大量时间画的很精美。它并不是最终用户看到的东西。

**Python http 服务器**

1. Http

Http（超文本传输协议）应该是大家日常接触到的最多的互联网技术。你用浏览器打开任意一个网页都会在地址栏里看到http这四个字母。



http是用来在网络间传输诸如网页、数据等信息的应用层协议。它最初用于浏览器和服务器之间的通信，现在也广泛用于各种网络组件之间通信。不懂http就几乎等于不懂网络编程。http遵循经典的客户端-服务端模型，客户端打开一个连接发出http请求，然后等待服务器端的响应。

1. 地址

每个http服务都要绑定到一个地址以接收请求，比如百度网站的地址是<http://www.baidu.com>。也就是说百度网站的服务器地址是[www.baidu.com](http://www.baidu.com)。这种地址叫做域名。可以理解为一个人在网络上注册的名字。注意这个名字和我们人的名字有点不同的是，它在互联网上是独一无二的。

还有一种地址叫做IP地址，比如183.232.231.173。简单理解IP地址就像人的身份证号。它是互联网上每台设备的独一无二的身份识别码。每个域名都要映射到一个IP地址上才能使用。互联网是先有IP地址再有域名的。由于IP地址太难记了，人们发明了域名来方便人们记住不同的网站。

要知道一个网站的IP地址很简单，在Windows菜单栏里输入‘CMD’打开命令提示符，然后输入ping和空格再加上网站的域名就可以了。比如下面的命令找到了百度网站的IP地址就是183.232.231.173：



1. 端口

一台拥有IP地址的服务器可以提供许多服务，比如网站服务、文件传输服务FTP、邮件服务SMTP等，这些服务完全可以通过1个IP地址来实现。那么，服务器是怎样区分不同的网络服务呢？显然不能只靠IP地址，因为IP 地址与网络服务的关系是一对多的关系。实际上是通过“IP地址+端口号”来区分不同的服务的。为了在一台设备上可以运行多个程序，人为的设计了端口(Port)的概念，类似的例子是公司内部的分机号码。

一个网络设备可以有65536个端口，0-1024之间多被操作系统占用，所以实际编程时一般采用1024以后的端口号。

要访问同一个网站的不同端口可以简单地在地址后面加冒号（：）和端口号，如访问百度地80端口：<http://www.baidu.com:80>

HTTP服务的默认端口是80和443，也就是说打开<http://www.baidu.com>实际上是打开<http://www.baidu.com:80> 或<http://www.baidu.com:443>，只不过浏览器很聪明地帮我们把补充了80或443端口。

HTTPServer

我们用python自带的HTTPServer类来做一个简单的可以接收http请求的服务器

1. 写一个SimpleHTTPRequestHandler类来继承python的BaseHTTPRequestHandler类：  
     
   from http.server import BaseHTTPRequestHandler  
   class SimpleHTTPRequestHandler(BaseHTTPRequestHandler):
2. SimpleHTTPRequestHandler重写父类BaseHTTPRequestHandler的do\_GET方法来处理http请求：  
     
   def do\_GET(self):
3. do\_GET方法根据请求地址的不同来返回不同的结果，比如收到‘/poll’请求就返回“hello“：

def do\_GET(self):

if self.path == '/poll':

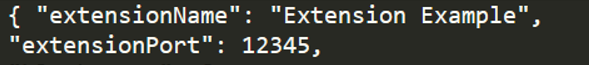
self.send\_response(200)

self.end\_headers()

self.wfile.write('hello'.encode('UTF-8'))

1. 导入HTTPServer：  
     
   from http.server import HTTPServer
2. 实列化HTTPServer， 它的地址是'localhost'，端口是12345。'localhost'是一个特殊的地址，指向本地即自己，因为我们的scratch游戏和这个http服务器是放在同一台电脑上的。SimpleHTTPRequestHandler是一个我们自己写的HTTP请求处理器。下面会详细说明：  
     
   httpd = HTTPServer(('localhost', 12345), SimpleHTTPRequestHandler)
3. 让这个服务器一直运行：  
   httpd.serve\_forever()
4. 现在简单的http服务器已经起来了。可以用浏览器测试打开[http://localhost:12345/poll](http://localhost:12345/poll%20“，然后浏览器会显示)  ，浏览器会显示“hello”就表示成功了。

接下来我们就可以根据“Scratch扩展的沟通协议”来具体实现scratch游戏的python http 客户端。记得把Scratch扩展配置文件 - “.s2e”文件里的“extensionPort”端口配成http服务器的端口，就可以实现Scratch和http服务器的通信了。



更多关于python http服务的内容请参考：

https://docs.python.org/3/library/http.server.html

Websocket 客户端和服务端

HTTP的一个局限性是只能从客户端向服务端发起请求，服务端不能主动向客户端发去消息。在有些时候，如一个玩家完成了一个动作之后，我们希望另一个玩家可以马上收到消息继续进行下一个动作，这种时候就需要游戏服务器主动向客户端发送消息了。Websocket协议是一个简单的解决方法，它用起来和HTTP类似，也是需要客户端和服务端。

我们使用了Aymeric Augustin的基于python asyncio的websocket实现。它使得使用websocket异常简单。下面是一个简单的websocket客户端，它连接到位于本机即“localhost”8765端口的websocket服务端，向它发送一句“Hello server”字符串：

**import** **asyncio**

**import** **websockets**

**async** **def** hello(uri):

**async** **with** websockets.connect(uri) **as** websocket:

**await** websocket.send("Hello server")

asyncio.get\_event\_loop().run\_until\_complete(

hello('ws://localhost:8765'))

下面是websocket服务端，它使用8765端口，当接到客户端的消息后就发送一句“Hello client”字符串：

**import** **asyncio**

**import** **websockets**

**async** **def** echo(websocket, path):

**async** **for** message **in** websocket:

**await** websocket.send("Hello client")

asyncio.get\_event\_loop().run\_until\_complete(

websockets.serve(echo, 'localhost', 8765))

asyncio.get\_event\_loop().run\_forever()

关于Aymeric Augustin websocket的详细使用方法请参考：

https://websockets.readthedocs.io/en/stable/

关于python asyncio，请参考：

https://docs.python.org/3/library/asyncio.html#module-asyncio