

## 04 | TCP三次握手：怎么使用套接字格式建立连接？

2019-08-09 盛延敏

网络编程实战

[进入课程 >](#)



讲述：冯永吉

时长 13:52 大小 12.71M



你好，我是盛延敏，这里是网络编程实战第 4 讲，欢迎回来。

在上一讲里我们介绍了 IPv4、IPv6 以及本地套接字格式，这一讲我们来讲一讲怎么使用这些套接字格式完成连接的建立，当然，经典的 TCP 三次握手理论也会贯穿其中。我希望经过这一讲的讲解，你会牢牢记住 TCP 三次握手和客户端、服务器模型。

让我们先从服务器端开始。

### 服务端准备连接的过程

#### 创建套接字

要创建一个可用的套接字，需要使用下面的函数：

```
1 int socket(int domain, int type, int protocol)
```

domain 就是指 PF\_INET、PF\_INET6 以及 PF\_LOCAL 等，表示什么样的套接字。

type 可用的值是：

**SOCK\_STREAM:** 表示的是字节流，对应 TCP；

**SOCK\_DGRAM：** 表示的是数据报，对应 UDP；

**SOCK\_RAW:** 表示的是原始套接字。

参数 protocol 原本是用来指定通信协议的，但现在基本废弃。因为协议已经通过前面两个参数指定完成。protocol 目前一般写成 0 即可。

## bind: 设定电话号码

创建出来的套接字如果需要被别人使用，就需要调用 bind 函数把套接字和套接字地址绑定，就像去电信局登记我们的电话号码一样。

调用 bind 函数的方式如下：

```
1 bind(int fd, sockaddr * addr, socklen_t len)
```


我们需要注意到 bind 函数后面的第二个参数是通用地址格式 `sockaddr * addr`。这里有一个地方值得注意，那就是虽然接收的是通用地址格式，实际上传入的参数可能是 IPv4、IPv6 或者本地套接字格式。bind 函数会根据 len 字段判断传入的参数 addr 该怎么解析，len 字段表示的就是传入的地址长度，它是一个可变值。

这里其实可以把 bind 函数理解成这样：

```
1 bind(int fd, void * addr, socklen_t len)
```

不过 BSD 设计套接字的时候大约是 1982 年，那个时候的 C 语言还没有 `void *` 的支持，为了解决这个问题，BSD 的设计者们创造性地设计了通用地址格式来作为支持 `bind` 和 `accept` 等这些函数的参数。

对于使用者来说，每次需要将 IPv4、IPv6 或者本地套接字格式转化为通用套接字格式，就像下面的 IPv4 套接字地址格式的例子一样：

 复制代码

```
1 struct sockaddr_in name;  
2 bind (sock, (struct sockaddr *) &name, sizeof (name))
```


对于实现者来说，可根据该地址结构的前两个字节判断出是哪种地址。为了处理长度可变的结构，需要读取函数里的第三个参数，也就是 `len` 字段，这样就可以对地址进行解析和判断了。

设置 `bind` 的时候，对地址和端口可以有多种处理方式。

我们可以把地址设置成本机的 IP 地址，这相当告诉操作系统内核，仅仅对目标 IP 是本机 IP 地址的 IP 包进行处理。但是这样写的程序在部署时有一个问题，我们编写应用程序时并不清楚自己的应用程序将会被部署到哪台机器上。这个时候，可以利用**通配地址**的能力帮助我们解决这个问题。通配地址相当于告诉操作系统内核：“Hi，我可不挑活，只要目标地址是咱们的都可以。”比如一台机器有两块网卡，IP 地址分别是 202.61.22.55 和 192.168.1.11，那么向这两个 IP 请求的请求包都会被我们编写的应用程序处理。

那么该如何设置通配地址呢？

对于 IPv4 的地址来说，使用 `INADDR_ANY` 来完成通配地址的设置；对于 IPv6 的地址来说，使用 `IN6ADDR_ANY` 来完成通配地址的设置。

 复制代码

```
1 struct sockaddr_in name;
```

```
2 name.sin_addr.s_addr = htonl (INADDR_ANY); /* IPV4 通配地址 */
```

除了地址，还有端口。如果把端口设置成 0，就相当于把端口的选择权交给操作系统内核来处理，操作系统内核会根据一定的算法选择一个空闲的端口，完成套接字的绑定。这在服务器端不常使用。

一般来说，服务器端的程序一定要绑定到一个众所周知的端口上。服务器端的 IP 地址和端口数据，相当于打电话拨号时需要知道的对方号码，如果没有电话号码，就没有办法和对方建立连接。

我们来看一个初始化 IPv4 TCP 套接字的例子：

 复制代码

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/socket.h>
4 #include <netinet/in.h>
5
6
7 int make_socket (uint16_t port)
8 {
9     int sock;
10    struct sockaddr_in name;
11
12
13    /* 创建字节流类型的 IPV4 socket. */
14    sock = socket (PF_INET, SOCK_STREAM, 0);
15    if (sock < 0)
16    {
17        perror ("socket");
18        exit (EXIT_FAILURE);
19    }
20
21
22    /* 绑定到 port 和 ip. */
23    name.sin_family = AF_INET; /* IPV4 */
24    name.sin_port = htons (port); /* 指定端口 */
25    name.sin_addr.s_addr = htonl (INADDR_ANY); /* 通配地址 */
26    /* 把 IPV4 地址转换成通用地址格式，同时传递长度 */
27    if (bind (sock, (struct sockaddr *) &name, sizeof (name)) < 0)
28    {
29        perror ("bind");
30        exit (EXIT_FAILURE);
31    }
```


```
32
33
34     return sock
35 }
```

## listen：接上电话线，一切准备就绪

bind 函数只是让我们的套接字和地址关联，如同登记了电话号码。如果要是让别人打通电话，还需要我们把电话设备接入电话线，让服务器真正处于可接听的状态，这个过程需要依赖 listen 函数。

初始化创建的套接字，可以认为是一个"主动"套接字，其目的是之后主动发起请求（通过调用 connect 函数，后面会讲到）。通过 listen 函数，可以将原来的"主动"套接字转换为"被动"套接字，告诉操作系统内核：“我这个套接字是用来等待用户请求的。”当然，操作系统内核会为此做好接收用户请求的一切准备，比如完成连接队列。

listen 函数的原型是这样的：

 复制代码


```
1 int listen (int sockfd, int backlog)
```

我来稍微解释一下。第一个参数 sockfd 为套接字描述符，第二个参数 backlog，官方的解释为未完成连接队列的大小，这个参数的大小决定了可以接收的并发数目。这个参数越大，并发数目理论上也会越大。但是参数过大也会占用过多的系统资源，一些系统，比如 Linux 并不允许对这个参数进行改变。对于 backlog 整个参数的设置有一些最佳实践，这里就不展开，后面结合具体的实例进行解读。

## accept: 电话铃响起了.....

当客户端的连接请求到达时，服务器端应答成功，连接建立，这个时候操作系统内核需要把这个事件通知到应用程序，并让应用程序感知到这个连接。这个过程，就好比电信运营商完成了一次电话连接的建立，应答方的电话铃声响起，通知有人拨打了号码，这个时候就需要拿起电话筒开始应答。

accept 这个函数的作用就是连接建立之后，操作系统内核和应用程序之间的桥梁。它的原型是：

 复制代码

```
1 int accept(int listensockfd, struct sockaddr *cliaddr, socklen_t *addrlen)
```

函数的第一个参数 listensockfd 是套接字，可以叫它为 listen 套接字，因为这就是前面通过 bind，listen 一系列操作而得到的套接字。函数的返回值有两个部分，第一个部分 cliadd 是通过指针方式获取的客户端的地址，addrlen 告诉我们地址的大小，这可以理解成当我们拿起电话机时，看到了来电显示，知道了对方的号码；另一个部分是函数的返回值，这个返回值是一个全新的描述字，代表了与客户端的连接。

这里一定要注意有两个套接字描述字，第一个是监听套接字描述字 listensockfd，它是作为输入参数存在的；第二个是返回的已连接套接字描述字。

你可能会问，为什么要把两个套接字分开呢？用一个不是挺好的么？

这里和打电话的情形非常不一样的地方就在于，打电话一旦有一个连接建立，别人是不能再打进来的，只会得到语音播报：“您拨的电话正在通话中。”而网络程序的一个重要特征就是并发处理，不可能一个应用程序运行之后只能服务一个客户，如果是这样，双 11 抢购得需要多少服务器才能满足全国“剁手党”的需求？

所以监听套接字一直都存在，它是要为成千上万的客户来服务的，直到这个监听套接字关闭；而一旦一个客户和服务器连接成功，完成了 TCP 三次握手，操作系统内核就为客户生成一个已连接套接字，让应用服务器使用这个**已连接套接字**和客户进行通信处理。如果应用服务器完成了对这个客户的服务，比如一次网购下单，一次付款成功，那么关闭的就是**已连接套接字**，这样就完成了 TCP 连接的释放。请注意，这个时候释放的只是这一个客户连接，其它被服务的客户连接可能还存在。最重要的是，监听套接字一直都处于“监听”状态，等待新的客户请求到达并服务。

## 客户端发起连接的过程

前面讲述的 bind、listen 以及 accept 的过程，是典型的服务器端的过程。下面我来讲下客户端发起连接请求的过程。




第一步还是和服务端一样，要建立一个套接字，方法和前面是一样的。

不一样的是客户端需要调用 `connect` 向服务端发起请求。

## connect: 拨打电话

客户端和服务端端的连接建立，是通过 `connect` 函数完成的。这是 `connect` 的构造函数：

 复制代码

```
1 int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen)
```

函数的第一个参数 `sockfd` 是连接套接字，通过前面讲述的 `socket` 函数创建。第二个、第三个参数 `servaddr` 和 `addrlen` 分别代表指向套接字地址结构的指针和该结构的大小。套接字地址结构必须含有服务器的 IP 地址和端口号。

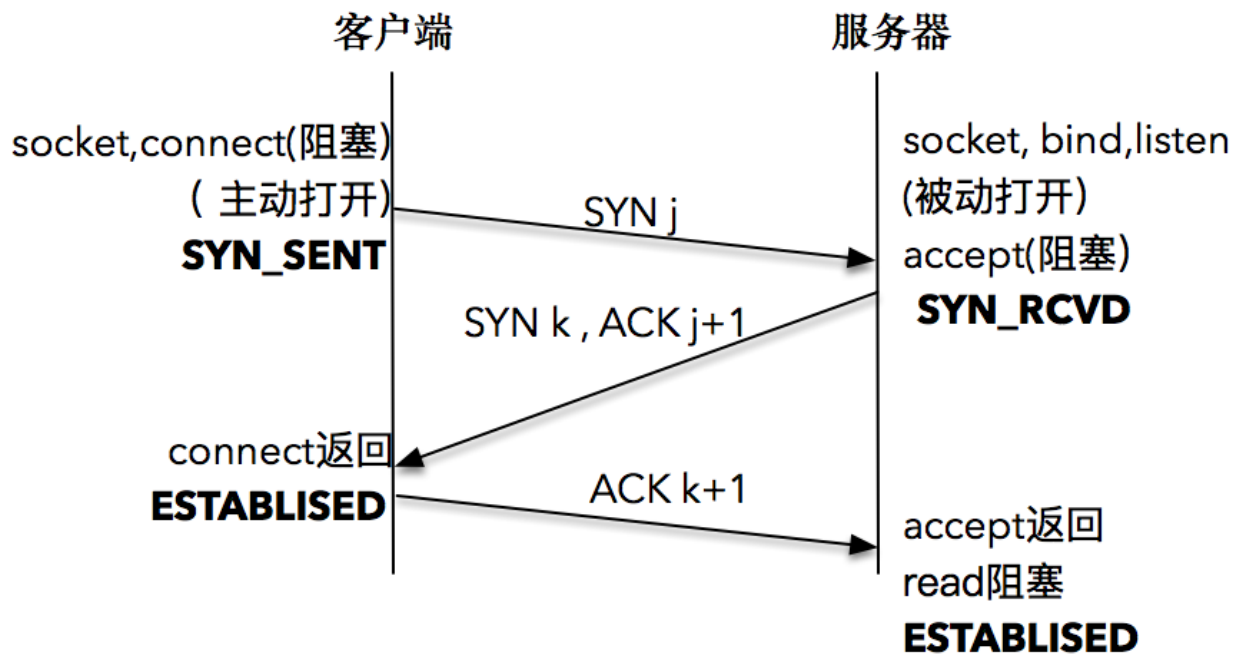
客户在调用函数 `connect` 前不必非得调用 `bind` 函数，因为如果需要的话，内核会确定源 IP 地址，并按照一定的算法选择一个临时端口作为源端口。

如果是 TCP 套接字，那么调用 `connect` 函数将激发 TCP 的三次握手过程，而且仅在连接建立成功或出错时才返回。其中出错返回可能有以下几种情况：

1. 三次握手无法建立，客户端发出的 SYN 包没有任何响应，于是返回 `TIMEOUT` 错误。这种情况比较常见的原因是对应的服务端 IP 写错。
2. 客户端收到了 `RST`（复位）回答，这时候客户端会立即返回 `CONNECTION REFUSED` 错误。这种情况比较常见于客户端发送连接请求时的请求端口写错，因为 `RST` 是 TCP 在发生错误时发送的一种 TCP 分节。产生 `RST` 的三个条件是：目的地为某端口的 SYN 到达，然而该端口上没有正在监听的服务器（如前所述）；TCP 想取消一个已有连接；TCP 接收到一个根本不存在的连接上的分节。
3. 客户发出的 SYN 包在网络上引起了 "destination unreachable"，即目的不可达的错误。这种情况比较常见的原因是客户端和服务端路由不通。

根据不同的返回值，我们可以做进一步的排查。

## 著名的 TCP 三次握手：这一次不用背记



你在各个场合都会了解到著名的 TCP 三次握手，可能还会被要求背下三次握手整个过程，但背后的原理和过程可能未必真正理解。我们刚刚学习了服务端和客户端连接的主要函数，下面结合这些函数讲解一下 TCP 三次握手的过程。这样我相信你不用背，也能根据理解轻松掌握这部分的知识。

这里我们使用的网络编程模型都是阻塞式的。所谓阻塞式，就是调用发起后不会直接返回，由操作系统内核处理之后才会返回。相对的，还有一种叫做非阻塞式的，我们在后面的章节里会讲到。

## TCP 三次握手的解读

我们先看一下最初的过程，服务器端通过 `socket`，`bind` 和 `listen` 完成了被动套接字的准备工作，被动的意思就是等着别人来连接，然后调用 `accept`，就会阻塞在这里，等待客户端的连接来临；客户端通过调用 `socket` 和 `connect` 函数之后，也会阻塞。接下来的事情是由操作系统内核完成的，更具体一点的说，是操作系统内核网络协议栈在工作。

下面是具体的过程：

1. 客户端的协议栈向服务器端发送了 SYN 包，并告诉服务器端当前发送序列号  $j$ ，客户端进入 `SYN_SENT` 状态；
2. 服务器端的协议栈收到这个包之后，和客户端进行 ACK 应答，应答的值为  $j+1$ ，表示对 SYN 包  $j$  的确认，同时服务器也发送一个 SYN 包，告诉客户端当前我的发送序列号



为  $k$ ，服务器端进入 SYNC\_RCVD 状态；

3. 客户端协议栈收到 ACK 之后，使得应用程序从 connect 调用返回，表示客户端到服务器端的单向连接建立成功，客户端的状态为 ESTABLISHED，同时客户端协议栈也会对服务器端的 SYN 包进行应答，应答数据为  $k+1$ ；
4. 应答包到达服务器端后，服务器端协议栈使得 accept 阻塞调用返回，这个时候服务器端到客户端的单向连接也建立成功，服务器端也进入 ESTABLISHED 状态。

形象一点的比喻是这样的，有 A 和 B 想进行通话：

A 先对 B 说：“喂，你在么？我在的，我的口令是  $j$ 。”

B 收到之后大声回答：“我收到你的口令  $j$  并准备好了，你准备好了吗？我的口令是  $k$ 。”

A 收到之后也大声回答：“我收到你的口令  $k$  并准备好了，我们开始吧。”

可以看到，这样的应答过程总共进行了三次，这就是 TCP 连接建立之所以被叫为“三次握手”的原因了。

## 总结

这一讲我们分别从服务端和客户端的角度，讲述了如何创建套接字，并利用套接字完成 TCP 连接的建立。

服务器端通过创建 socket，bind，listen 完成初始化，通过 accept 完成连接的建立。

客户端通过场景 socket，connect 发起连接建立请求。

在下一讲里，我们将真正地开始客户端 - 服务端数据交互的过程。

## 思考题

最后给你布置两道思考题。

第一道是关于阻塞调用的，既然有阻塞调用，就应该有非阻塞调用，那么如何使用非阻塞调用套接字呢？使用的场景又是哪里呢？

第二道是关于客户端的，客户端发起 connect 调用之前，可以调用 bind 函数么？

欢迎你在评论区与我分享你的答案，如果这篇文章帮助你理解 TCP 三路握手，也欢迎你点击“请朋友读”，把这篇文章分享给你的朋友或者同事。



# 网络编程实战

从底层到实战，深度解析网络编程

盛延敏

前大众点评云平台首席架构师



新版升级：点击「 请朋友读」，20位好友免费读，邀请订阅更有**现金**奖励。

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 03 | 套接字和地址：像电话和电话号码一样理解它们

下一篇 05 | 使用套接字进行读写：开始交流吧

## 精选留言 (39)

写留言



假装在火星

2019-08-09

之前看过一些文章解释，为什么tcp建立连接需要三次握手，解释如下

tcp连接的双方要确保各自的收发消息的能力都是正常的。

客户端第一次发送握手消息到服务端，

服务端接收到握手消息后把ack和自己的syn一同发送给客户端，这是第二次握手，...

展开 ∨

作者回复: 赞

1

13



阿西吧

2019-08-09

这个问题的本质是, 信道不可靠, 但是通信双方需要就某个问题达成一致. 而要解决这个问题, 无论你在消息中包含什么信息, 三次通信是理论上的最小值. 所以三次握手不是TCP本身的要求, 而是为了满足"在不可靠信道上可靠地传输信息"这一需求所导致的

展开

作者回复: 赞。

4

4



星亦辰

2019-08-09

思考题2

客户端可以bind 指定使用固定端口来连接。  
没有bind 则会产生一个随机的端口来完成连接请求。

...

展开

作者回复: 想多了, 可以区分出来是不是本地监听端口(被动套接字)的, 而且这个在大多数情况下不被允许的。

3

2



\_stuView

2019-08-09

sockfd里的这个fd代表什么

作者回复: file description , 文件描述符。UNIX世界里万物皆文件。

6

2



周思进

2019-08-09

刚好最近写了个通过man帮助手册编写基础tcp服务器



1



浦上清风

2019-08-09

1. 非阻塞 == 异步通信 ???
2. 可以是可以，但是不安全 ???

作者回复: 1 不是等价的

2 没有必要，还徒增了端口冲突的危险。



1



zhchnchn

2019-08-09

这篇很是解惑，感谢。有2个问题想请教老师：

1. ``socket``函数的参数``domain``的值，在``bind``函数的参数``addr``的``sin_family``中也需要设置，这样不是重复了吗？
2. ``connect``函数出错返回可能的3种情况中，其中第1种“TIMEOUT 错误”和第3种“destination unreachable”，感觉都是连接不到服务端的IP，这两种情况有什么区别吗？

作者回复: 1.两个不是一个东西，一个是套接字地址，一个是套接字，都需要设置。

2.unreachable是被动收到了其他网络涉笔发来的ICMP报文信息，而timeout是在尝试一段时间后主动放弃的。



2

1



广训

2019-08-09

记得很久前学套接字，提供的例子就是客户端bind端口，基本第二次和以后在运行，就告诉端口被占用，那时候不懂，就换个端口再来一次

作者回复: 这是有原因的，不需要每次都换端口，下面会降到具体解法。



1

1



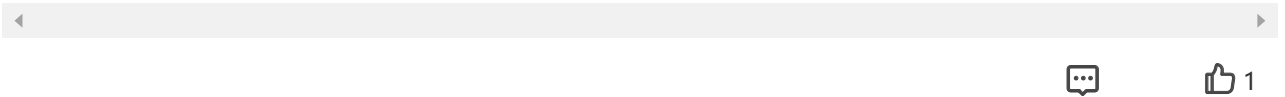
星亦辰

2019-08-09

非阻塞，一般是把请求接收到的套接字传递给子进程或子线程，然后，主进程，主线程继续等待下一个请求。多数网络服务器都是这个路子吧

作者回复: 非阻塞，是指I/O模型，不是等待数据或连接，还是通过I/O通知的方式来感知数据或连接。

后面的章节会详细讲述非阻塞模型



**nil**

2019-08-12

谈下自己关于三次握手的理解，三次握手主要解决的是如何在信道不可靠的情况下建立双向连接。那为何偏偏是三次握手，两次不行吗？四次不行吗？其实三次以上都是可以的，但是多于三次的握手属于非必要操作。两次是无法保证双向通信的建立，三个握手之中的第二次通信既是对第一次通信的回应，又是为了第三次握手发起请求，起到承上启下的作用。三次握手成功之后，通信双方的信道基本可靠，之后就要靠tcp协议保证整个传输的可靠性。



**平少**

2019-08-12

打卡



**有点意思**

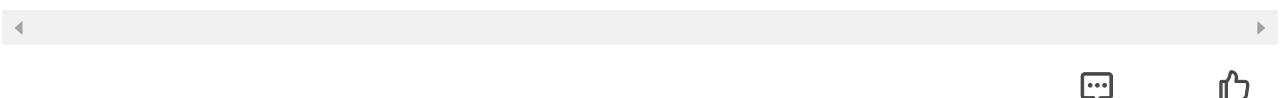
2019-08-12

老师你好

我看有的书上 做客户端编程的时候 建立socket连接后会启用两个线程 一个线程用来读 一个线程用来写 现在有点疑问 一个socket这样给两个线程用 不需要加锁么？

展开 ∨

作者回复: 不需要，读和写是两个不同的缓冲区。



**初见**

2019-08-11

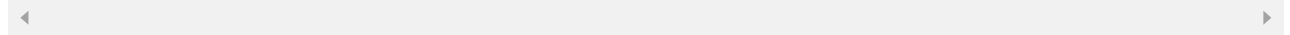
老师您好，一个小问题~

bind 的第三个参数 len，函数本身是可以根据第二个字段 addr 计算出来，为什么每次非要让调用者多此一举算完传进来呢？

...

展开 ▾

作者回复: 不能完全计算出来，比如变长地址，不知道len，是没有办法知道addr确切值的边界的。



1

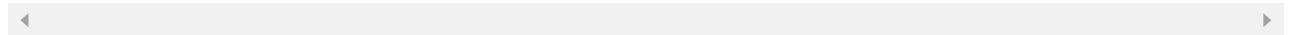


**Knight<sup>2018</sup>**

2019-08-11

老师我有一个问题，编程语言中的IO模型和操作系统中的IO关系是什么？

作者回复: 就是要和操作系统I/O打交道，编程语言的I/O模型是通过抽象和设计，总结出的一套规范。



1



**有点意思**

2019-08-11

老师好

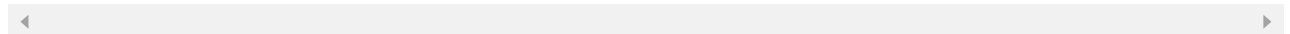
现在在做c++网络编程这块 linux和windows下都要用

现在遇到一个问题 客户端和服务端的心跳检测怎么做？

用keepalive好么？还有其他更好的方案么？

展开 ▾

作者回复: 往下看，很快讲到这部分内容。



1



**彭俊**

2019-08-11

如何使用非阻塞调用套接字:使用fcntl函数设置套接字的属性fcntl(fd, F\_SETFL, flags);  
非阻塞调用的 使用的场景：程序在调用返回之前，需要做其他事情，可以选择用定时轮询或事件通知的方式获取调用结果。

是否可以调用bind 函数：可以，但是调用bind函数，也就是客户端指定了端口号，这样容易造成端口冲突，所以客户端不调用bind函数，让系统自动选择空闲端口比较好





星亦辰

2019-08-11

<https://github.com/v1xingyue/geeknetwork/tree/master/simpletcp>



wang

2019-08-11

1. 非阻塞调用，意思也就是在服务器进程在accept 或者 connect 函数中直接return，而没有一定要这次连接应答成功然后才能return
2. 客户端当然可以使用bind 函数，显示让内核不随机分配端口号

展开 ∨



Better me

2019-08-10

老师通配地址那块有点不理解，单独设置ip地址的话，是需要提前知道该应用程序将部署到那台服务器上，而通配地址不需要那样是因为可以自动识别到服务器上每个网卡的ip地址吗？

作者回复: 你可以理解通配地址为\*，意思就是这台机器上的所有IP地址都可以归我管。



星亦辰

2019-08-10

发现一个问题。

我的server 端bind 的是0.0.0.0 的地址。

但是，我的client 连接 localhost 不能连接到，但是 连接 127.0.0.1 可以，这是为什么呢？

展开 ∨

作者回复: localhost配置了别名没？



