Apache源代码全景分析：网络地址处理 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/252/2021_2022_Apache_E6_BA_90_E4_c103_252950.htm 套接字地址 在了解APR中对IP地址的封装之前，我们首先看一下通常情况下对IP地址的使用情况。下面的代码掩饰了简单的服务器端套接字的地址初始化过程： struct sockaddr_in server_addr. /* 本机地址信息 */ server_addr.sin_family=AF_INET. server_addr.sin_port=htons(SERVPORT). server_addr.sin_addr.s_addr = INADDR_ANY. bzero(amp.my_addr, sizeof(struct sockaddr)). accept(sockfd, (struct sockaddr *)amp.sin_size). Socket API中提供了三种类型的地址：sockaddr，sockaddr_in和sockaddr_un。sockaddr是通用的套接字结构，sockaddr_in为Internet协议族的地址描述结构，sockaddr_un则是Unix协议组的地址描述结构。sockaddr_in结构中的sa_family决定是sockaddr_in还是sockaddr_un。 如果直接使用Socket API提供的地址结构，则至少存在下面的几个问题： 1、在网络应用程序中，对于internet地址，如上面的程序代码所示，通常总是使用sockaddr_in描述，而在一些Socket API函数中则使用sockaddr作为套接字地址，因此在使用的时候必须将sockaddr_in强制转换为sockaddr类型，这是一个麻烦而且容易出错的地方。 2、sockaddr_in也不是一个特别容易理解的数据结构。通常情况下，sin_family和sin_port相对容易记忆，而套接字地址sin_addr.s_addr则未必。套接字的这种结构对一般人而言无疑是一种噩梦。 3、另外一个问题则是Ipv6的地址问题。目前，Apache已经开始同时支持Ipv4

和Ipv6两种类型的地址，如果用户需要支持Ipv6，则还必须使用Ipv6对应的地址数据结构。 对于一个良好的类库，不管是Ipv4还是Ipv6协议，都必须提供同样的接口，这种接口必须简单易懂，同时必须尽可能的隐藏内部的细节，比如对于sin_addr.s_addr无非暴露给用户。 基于上面的分析，APR中只使用一种数据结构apr_sockaddr_t来描述IP地址，该结构定义在文件apr_network_io.h中： struct apr_sockaddr_t { apr_pool_t *pool. /*第一部分*/ char *hostname. char *servname. /*第二部分*/ apr_port_t port. apr_int32_t family. union { struct sockaddr_in sin. #if APR_HAVE_IPV6 struct sockaddr_in6 sin6. #endif #if APR_HAVE_SA_STORAGE struct sockaddr_storage sas. #endif } sa. /*第三部分*/ apr_socklen_t salen. int ipaddr_len. int addr_str_len. void *ipaddr_ptr. apr_sockaddr_t *next. }. 该结构描述了socket地址的三部分的信息内容： 第一部分： Hostname是该地址对应的主机名称，而servname则是对应端口的服务名称，比如80对应的名称为 ”www ”，21端口对应的servname则是 ”FTP ”。如果某个端口比如9889并没有对应某个众所皆知的服务，那么servname则直接是端口的字符串描述。 第二部分： 该部分则对应的是sockaddr结构中的内容，port是端口，family则是地址协议族类型，包括AF_INET，AF_UNIX等。sa则为联合类型，用以描述对应的套接字地址，或者是Ipv4类型，或者是Ipv6类型，两者只能居其一。 