Socket 是通用 BSDsocket，其他不同的 socket 都是在此基础上进行扩展。

```
struct socket {
        socket_state            state;
        unsigned long           flags;
        const struct proto_ops      *ops;从传输层到 socket 的接口
        struct fasync_struct *fasync_list;
        struct file       *file;
        struct sock               *sk;
        wait_queue_head_t wait;
        short                     type;
};
```

是一组传输层给系统提供的函数接口。不同的协议(TCP, UDP, RAW)都要这个 proto_ops 的实例，实现这些接口。

```
struct proto_ops {
        int          family;
        struct module  *owner;
        int          (*release)      (struct socket *sock);
        int          (*bind)             (struct socket *sock,
                                      struct sockaddr *myaddr,
                                      int sockaddr_len);
        int          (*connect)      (struct socket *sock,
                                      struct sockaddr *vaddr,
                                      int sockaddr_len, int flags);
        int          (*socketpair)(struct socket *sock1,
                                      struct socket *sock2);
        int          (*accept)       (struct socket *sock,
                                      struct socket *newsock, int flags);
        int          (*getname)      (struct socket *sock,
                                      struct sockaddr *addr,
                                      int *sockaddr_len, int peer);
        unsigned int    (*poll)             (struct file *file, struct socket *sock,
                                      struct poll_table_struct *wait);
        int          (*ioctl)        (struct socket *sock, unsigned int cmd,
                                      unsigned long arg);
        int          (*compat_ioctl) (struct socket *sock, unsigned int cmd,
                                      unsigned long arg);
        int          (*listen)       (struct socket *sock, int len);
        int          (*shutdown)    (struct socket *sock, int flags);
        int          (*setsockopt)(struct socket *sock, int level,
                                      int optname, char __user *optval, int optlen);
        int          (*getsockopt)(struct socket *sock, int level,
                                      int optname, char __user *optval, int __user *optlen);
        int          (*compat_setsockopt)(struct socket *sock, int level,
                                      int optname, char __user *optval, int optlen);
        int          (*compat_getsockopt)(struct socket *sock, int level,
                                      int optname, char __user *optval, int __user *optlen);
        int          (*sendmsg)      (struct kiocb *iocb, struct socket *sock,
                                      struct msghdr *m, size_t total_len);
        int          (*recvmsg)      (struct kiocb *iocb, struct socket *sock,
                                      struct msghdr *m, size_t total_len,
                                      int flags);
        int          (*mmap)           (struct file *file, struct socket *sock,
                                      struct vm_area_struct * vma);
        ssize_t            (*sendpage)    (struct socket *sock, struct page *page,
                                      int offset, size_t size, int flags);
};
```

```
struct sock_common {
        unsigned short          skc_family;
        volatile unsigned char     skc_state;
        unsigned char           skc_reuse;
        int            skc_bound_dev_if;
        struct hlist_node       skc_node;
        struct hlist_node       skc_bind_node;
        atomic_t          skc_refcnt;
        unsigned int            skc_hash;
        struct proto            *skc_prot;
};是网络层对 socket 表示的最简形式
```

```
enum sock_type {
        SOCK_STREAM    = 1,
        SOCK_DGRAM     = 2,
        SOCK_RAW   = 3,
        SOCK_RDM   = 4,
        SOCK_SEQPACKET      = 5,
        SOCK_DCCP = 6,
        SOCK_PACKET     = 10,
};
```

Sock 是 socket 的网络层表示，是 TCP 传输控制块

```
struct sock {
        /*
         * Now struct inet_timewait_sock also uses sock_common, so please just
         * don't add nothing before this first member (__sk_common) --acme
         */
        struct sock_common       __sk_common;
#define sk_family           __sk_common.skc_family
#define sk_state            __sk_common.skc_state
#define sk_reuse            __sk_common.skc_reuse
#define sk_bound_dev_if             __sk_common.skc_bound_dev_if
#define sk_node                 __sk_common.skc_node
#define sk_bind_node            __sk_common.skc_bind_node
#define sk_refcnt           __sk_common.skc_refcnt
#define sk_hash                 __sk_common.skc_hash
#define sk_prot                 __sk_common.skc_prot  从网络层到传输层的函数接口
        unsigned char        sk_shutdown : 2,
                             sk_no_check : 2,
                             sk_userlocks : 4;
        unsigned char        sk_protocol;
        unsigned short       sk_type;
        int                  sk_rcvbuf;
        socket_lock_t        sk_lock;
        wait_queue_head_t *sk_sleep;
        struct dst_entry     *sk_dst_cache;
        struct xfrm_policy   *sk_policy[2];
        rwlock_t          sk_dst_lock;
        atomic_t          sk_rmem_alloc;
        atomic_t          sk_wmem_alloc;
        atomic_t          sk_omem_alloc;
        struct sk_buff_head sk_receive_queue;
        struct sk_buff_head sk_write_queue;
        struct sk_buff_head sk_async_wait_queue
        int               sk_wmem_queued;
        int               sk_forward_alloc;
        gfp_t                sk_allocation;
        int               sk_sndbuf;
        int               sk_route_caps;
        int               sk_gso_type;
        int               sk_rcvlowat;
        unsigned long             sk_flags;
        unsigned long             sk_lingertime;
        /*
         * The backlog queue is special, it is always used with
         * the per-socket spinlock held and requires low latency
         * access. Therefore we special case it's implementation.
         */
        struct {
                struct sk_buff *head;
                struct sk_buff *tail;
        } sk_backlog;
        struct sk_buff_head sk_error_queue;
        struct proto         *sk_prot_creator;
        rwlock_t          sk_callback_lock;
        int               sk_err,
                          sk_err_soft;
        unsigned short       sk_ack_backlog;
        unsigned short       sk_max_ack_backlog;
        __u32                sk_priority;
        struct ucred         sk_peercred;
        long              sk_rcvtimeo;
        long              sk_sndtimeo;
        struct sk_filter             *sk_filter;
        void              *sk_protinfo;
        struct timer_list    sk_timer;
        struct timeval       sk_stamp;
        struct socket        *sk_socket;
        void              *sk_user_data;
        struct page          *sk_sndmsg_page;
        struct sk_buff       *sk_send_head;
        __u32                sk_sndmsg_off;
        int               sk_write_pending;
        void              *sk_security;
        void              (*sk_state_change)(struct sock *sk);
        void              (*sk_data_ready)(struct sock *sk, int bytes);
        void              (*sk_write_space)(struct sock *sk);
        void              (*sk_error_report)(struct sock *sk);
        int               (*sk_backlog_rcv)(struct sock *sk,
                                  struct sk_buff *skb);
        void                          (*sk_destruct)(struct sock *sk);
```

表示 socket 的连接状态：

```
enum {
        TCP_ESTABLISHED = 1,
        TCP_SYN_SENT,
        TCP_SYN_RECV,
        TCP_FIN_WAIT1,
        TCP_FIN_WAIT2,
        TCP_TIME_WAIT,
        TCP_CLOSE,
        TCP_CLOSE_WAIT,
        TCP_LAST_ACK,
        TCP_LISTEN,
        TCP_CLOSING,    /*  Now  a
valid state */

        TCP_MAX_STATES      /*
Leave at the end! */
};
```

Inet_sock 是 INET 域（ipv4 协议族）专用的 socket 表示，在 struct sock 的基础上进行了扩展，提供了 INET 域专有的属性：TTL, IP 地址，端口等

```
struct inet_sock {
        /* sk and pinet6 has to be the first two members of inet_sock */
        struct sock             sk;
#if defined(CONFIG_IPV6) || defined(CONFIG_IPV6_MODULE)
        struct ipv6_pinfo    *pinet6;
#endif
        /* Socket demultiplex comparisons on incoming packets. */
        __be32                  daddr; //目的 IP 地址
        __be32                  rcv_saddr; //本地接收地址
        __be16                  dport; //目的端口
        __u16                   num;
        __be32                  saddr; //发送地址
        __s16                   uc_ttl; //单播 TTL
        __u16                   cmsg_flags;
        struct ip_options       *opt;
        __be16                  sport; //源端口
        __u16                   id;
        __u8                    tos; //服务类型
        __u8                    mc_ttl; //组播 TTL
        __u8                    pmtudisc;
        __u8                    recverr:1,
                            is_icsk:1,
                            freebind:1,
                            hdrincl:1,
                            mc_loop:1;
        int                 mc_index;
        __be32                  mc_addr;
        struct ip_mc_socklist      *mc_list;
        struct {
                unsigned int          flags;
                unsigned int          fragsize;
                struct ip_options     *opt;
                struct rtable         *rt;
                int               length; /* Total length of all frames */
                __be32                addr;
                struct flowi          fl;
        } cork;
};
```

Inet_connection_sock 表示的是 INET 域中面向连接的 socket，在 inet_sock 的基础上进行扩展，添加了有关连接的一些属性，如重传计时器、拥塞控制算法等

```
struct inet_connection_sock {
        /* inet_sock has to be the first member! */
        struct inet_sock          icsk_inet;
        struct request_sock_queue icsk_accept_queue;
        struct inet_bind_bucket       *icsk_bind_hash;
        unsigned long             icsk_timeout;
        struct timer_list         icsk_retransmit_timer;
        struct timer_list         icsk_delack_timer;
        __u32                     icsk_rto;
        __u32                     icsk_pmtu_cookie;
        const struct tcp_congestion_ops *icsk_ca_ops;
        const        struct        inet_connection_sock_af_ops
*icsk_af_ops;
        unsigned int              (*icsk_sync_mss)(struct    sock
*sk, u32 pmtu);
        __u8                      icsk_ca_state;
        __u8                      icsk_retransmits;
        __u8                      icsk_pending;
        __u8                      icsk_backoff;
        __u8                      icsk_syn_retries;
        __u8                      icsk_probes_out;
        __u16                     icsk_ext_hdr_len;
        struct {
        __u8                pending;       /* ACK is pending
                  */
        __u8                quick;   /*  Scheduled  number  of
quick acks          */
        __u8                pingpong;    /*   The   session   is
interactive             */
        __u8                blocked;      /*  Delayed ACK  was
blocked by socket lock */
        __u32               ato;          /*  Predicted  tick  of
soft clock      */
        unsigned long     timeout;       /* Currently scheduled
timeout             */
```