四种模式:

标准模式(standard mode): 发送的消息已经到了守护进程

缓冲模式(buffered mode): 发送的消息进了程序提供的缓冲区

同步模式(synchronous mode): 接收方已经开始接收

就绪模式(ready mode): 接收方先调用

```
MPI_Send     MPI_Bsend    MPI_Ssend    MPI_Rsend
MPI_Recv     MPI_Brecv    MPI_Srecv    MPI_Rrecv

MPI_Buffer_attach(void * buffer, int size)
```

阻塞：等待操作安全完成以后才返回

可能导致程序死锁！

非阻塞：立刻返回，需要时候查询操作完成情况

在操作完成前，不要去修改缓冲区，否则可能出错！

MPI_Send		MPI_Isend	
MPI_Recv		MPI_Irecv	
MPI_Wait	MPI_Waitany	MPI_Waitall	MPI_Waitsome
MPI_Test	MPI_Testany	MPI_Testall	MPI_Testsome

消息探测和通信请求的释放和取消

```
int MPI_Probe(int src, int tag, MPI_Comm comm,
        MPI_Status * status);

int MPI_Iprobe(int src, int tag, MPI_Comm comm,
        int * flag, MPI_Status * status);

int MPI_Get_count(MPI_Status * status,
        MPI_Datatype datatype, int * count);

int MPI_Request_free(MPI_Request * request);

int MPI_Cancel(MPI_Request * request);
```

持久通信请求

```
int MPI_Send_init(void * buffer, int count,
        MPI_Datatype datatype, int dest, int tag,
        MPI_Comm comm, MPI_Request * request);

int MPI_Recv_init(void * buffer, int count,
        MPI_Datatype datatype, int src, int tag,
        MPI_Comm comm, MPI_Request * request);

int MPI_Start(MPI_Request * request);

int MPI_Startall(int count, MPI_Request * requests);
```

原始数据类型

MPI	C
MPI_INT	int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_SHORT	short
MPI_LONG	long (int)
MPI_CHAR	char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short
MPI_UNSIGNED	unsigned (int)
MPI_UNSIGNED_LONG	unsigned long
MPI_LONG_DOUBLE	long double
MPI_BYTE	
MPI_PACKED	

原始数据类型

MPI	Fortran
MPI_INTEGER	INTEGER
MPI_REAL	REAL
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_DOUBLE_COMPLEX	DOUBEL COMPLEX
MPI_LOGICAL	LOGICAL
MPI_CHARACTER	CHARACTER*1
MPI_BYTE	
MPI_PACKED	

派生数据类型

（类型，位移）对：类型是已知数据类型
位移是偏移字节数

数据类型的描述由（类型，位移）对的序列构成

$$type = \{(type_0, disp_0), \cdots, (type_{n-1}, disp_{n-1})\}$$

大小

$$\sum_{i=0}^{n-1} sizeof(type_i)$$

下界，上界和域，MPI_LB 和 MPI_UB

$$lb(type) = \min_{i=0}^{n-1} disp_i$$

$$up(type) = \max_{i=0}^{n-1}(disp_i + sizeof(type_i)) + \epsilon$$

$$extent(type) = ub(type) - lb(type)$$

查询数据类型的信息

```
int MPI_Type_size(MPI_Datatype datatype,
                  int * size);

int MPI_Type_extent(MPI_Datatype datatype,
                  MPI_Aint * extent);

int MPI_Type_lb(MPI_Datatype datatype,
                  MPI_Aint * lb);

int MPI_Type_ub(MPI_Datatype datatype,
                  MPI_Aint * ub);

typedef long MPI_Aint;
```

创建数据类型

```
int MPI_Type_contiguous(int count,
                        MPI_Datatype old_type,
                        MPI_Datatype * new_type);
```

连续 count 个 old_type 构成 new_type，下一个的开始位移根据 old_type 的 extent 进行计算。

创建数据类型

```
int MPI_Type_vector(int count, int blocklength,
                    int stride, MPI_Datatype old_type,
                    MPI_Datatype * new_type);
```

count 个数据块构成 new_type，每个数据块由连续 blocklength 个 old_type 构成，每个数据块之间的位移差别为 stride*extent(old_type) 个字节。

```
int MPI_Type_hvector(int count, int blocklength,
                     int stride, MPI_Datatype old_type,
                     MPI_Datatype * new_type);
```

count 个数据块构成 new_type，每个数据块由 blocklength 个连续 old_type 组成，每两个数据块之间的位移差别为 stride 个字节。

创建数据类型

```
int MPI_Type_indexed(int count,
            int * blocklengths,
            MPI_Aint * displacements,
            MPI_Datatype old_type,
            MPI_Datatype * new_type);
```

count 个数据块构成 new_type，第 i 个数据块是 blocklengths[i] 个连续的 old_type 构成，第 i-1 和 i 个数据块之间的位移差别是 displacements[i]*sizeof(old_type) 个字节。

```
int MPI_Type_hindexed(int count,
            int * blocklengths,
            MPI_Aint * displacements,
            MPI_Datatype old_type,
            MPI_Datatype * new_type);
```

count 个数据块构成 new_type，第 i 个数据块是 blocklengths[i] 个连续的 old_type 构成，第 i-1 和 i 个数据块之间的位移差别是 displacements[i] 个字节。

创建数据类型

```
int MPI_Type_struct(int count,
            int * blocklengths,
            MPI_Aint * displacements,
            MPI_Datatype * old_types,
            MPI_Datatype * new_type);
```

count 个数据块构成 new_type，第 i 个数据块是 blocklengths[i] 个连续的 old_types[i] 构成，第 i-1 和 i 个数据块之间的个字节是 displacements[i] 个字节。

提交和释放数据类型

```
int MPI_Type_commit(MPI_Datatype * datatype);

int MPI_Type_free(MPI_Datatype * datatype);
```

数据类型释放以后不会影响其派生类型！

数据的打包和拆包

```
int MPI_Pack(void * inbuf, int in_count,
             MPI_Datatype datatype, void * outbuf,
             int out_size, int * position,
             MPI_Comm comm);

int MPI_Unpack(void * inbuf, int in_size,
             int * position, void * outbuf,
             int out_count, MPI_Datatype datatype,
             MPI_Comm comm);
```

第一次打包前将 position 设为 0 。