data

INFO_data_IsExist
判断入参工号是否存在
（返回值为 BOOL_T 类型）

INFO_data_Create
为入参工号创建一个节点
并将该节点加入链表
- 申请新节点的内存空间
- 初始化该节点
- 将该节点加入链表中

info_data_Alloc
申请内存并将申请到的内存空间清零，返回空间首地址

info_data_Add
将入参指针指向的链表节点（按照工号从小到大顺序）加入链表结构当中

INFO_data_SetName
将入参工号的姓名更改为入参姓名
- 获取入参工号对应节点指针
- 更改姓名

INFO_data_SetSex
将入参工号的性别更改为入参性别
- 获取入参工号对应节点指针
- 更改性别

INFO_data_SetAge
将入参工号的年龄更改为入参年龄
- 获取入参工号对应节点指针
- 更改年龄

INFO_data_SetHeight
将入参工号的身高更改为入参身高
- 获取入参工号对应节点指针
- 更改身高

info_data_Get
返回入参工号对应的链表节点指针
若该工号不存在则返回 NULL

INFO_data_Destroy
删除入参工号对应的链表节点
- 获取待删除节点地址
若该工号无信息则返回
- 从链表上摘除该节点
- 释放内存空间

info_data_Delete
将入参指针指向的链表节点从链表结构当中摘除（并不释放内存）

info_data_Free
释放入参指针指向的内存空间

INFO_data_IsEmpty
判断整个数据组织是否为空
（返回值为 BOOL_T 类型）

INFO_data_GetFirst
返回第一个有数据的工号
（即链表中第一个节点的工号）

INFO_data_GetNext
返回下一个有数据的工号
（即链表中所有比入参工号大的工号中最小的一个）。
此接口获取下一个工号不依赖于入参工号本身在链表中是否存在

INFO_data_GetData
将入参工号的数据返回到出参结构体
- 获取入参工号对应节点指针
- 将入参工号的数据复制到出参结构体当中

INFO_data_Init
模块初始化：数据组织初始化

INFO_data_Fini
模块退出：释放所有资源

proc

INFO_proc_Add
处理 Add 操作
- 调用 parse 接口解析输入
- 判断参数是否全部有效
- 判断入参工号是否已存在
- 向内部链表中加入该工号
- 设置该工号的各项数据

INFO_proc_Modify
处理 Modify 操作
- 调用 parse 接口解析输入
- 判断入参工号是否有效
- 判断入参工号是否存在
- 将入参中有效的数据覆盖到该工号的相应数据项

INFO_proc_Delete
处理 Delete 操作
- 调用 parse 接口解析输入
- 判断入参工号是否有效
- 删除该工号的数据

INFO_proc_Display
处理 Display 操作
- 没有任何数据
if( )
{
显示” No info.”
}
else
{
- 显示所有数据
}

INFO_proc_OpenDebug
打开调试开关

INFO_proc_CloseDebug
关闭调试开关

INFO_proc_Exit
处理 Exit 操作
无操作，直接返回

parse

info_parse_Each
将入参单个数据项字符串中的信息
赋给出参结构体的相应项

INFO_parse_InputStr
解析入参用户输入字符串得到出参配置数据结构体

main

INFO_Run
运行
do
{
- 显示菜单
- 获取“菜单选择”
- 获取“输入字符串”
- 执行相应操作
- 输出错误提示信息

}while(用户没有选择 Exit);

用户根据菜单输入选择

用户输入数据信息
（如工号等）

info_main_DisplayMenu
显示菜单

info_main_GetMenuChoice
和用户交互，
获取“菜单选择” enMenu

info_main_GetInputStr
根据菜单选择和用户交互，
获取 " 输入字符串 "pcInputStr

info_main_DisplayErrInfo
输出错误提示信息

INFO_Init
初始化

Main
- 初始化
- 运行
- 去初始化

INFO_Fini
去初始化

dbg
保存了一个调试开关 g_bInfoDbg，用来标记是否要打印各条调试信息

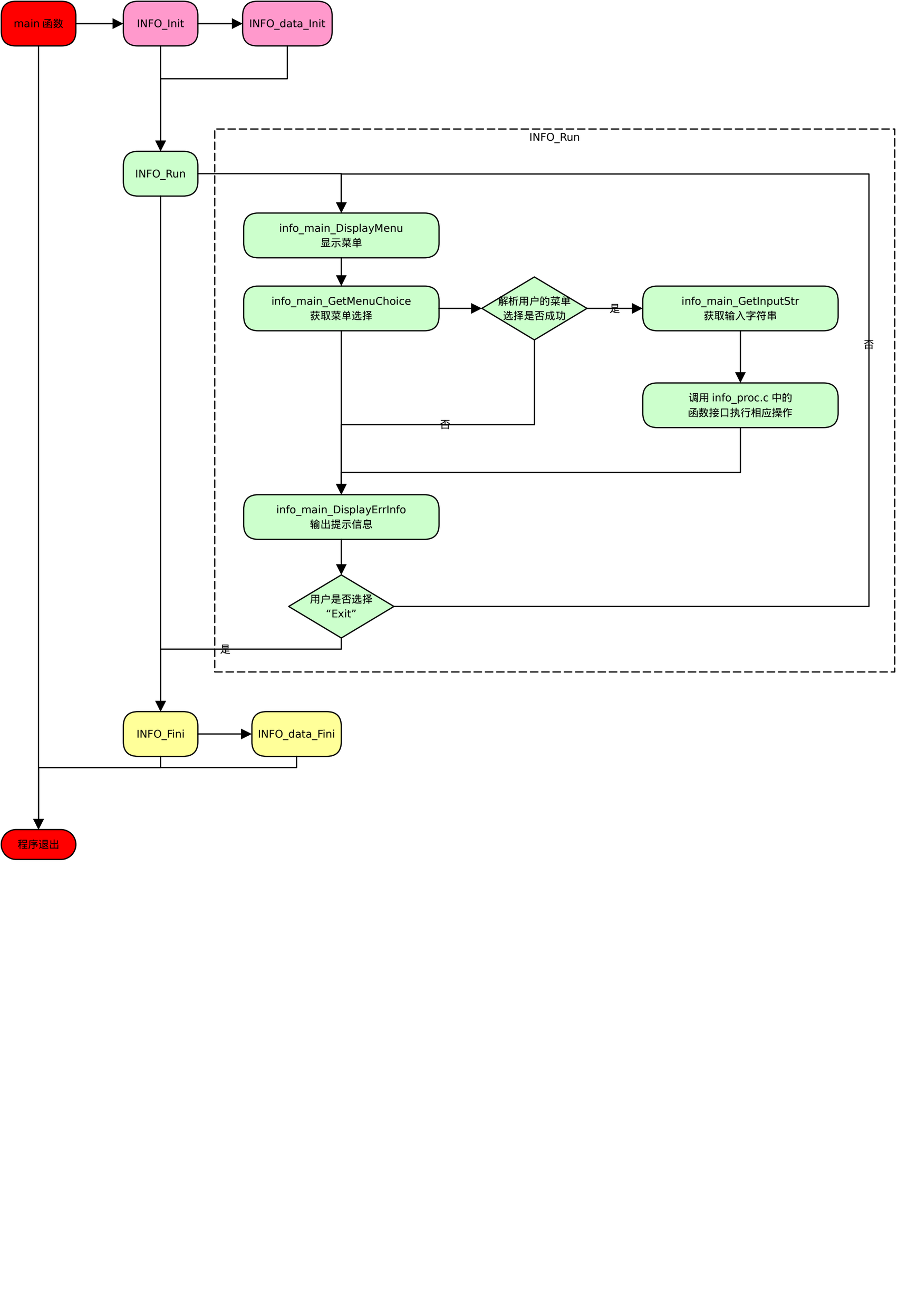
INFO_dbg_Set
设置调试开关

其他函数通过
该宏来打印调试信息

INFO_DBG_PRINT 宏

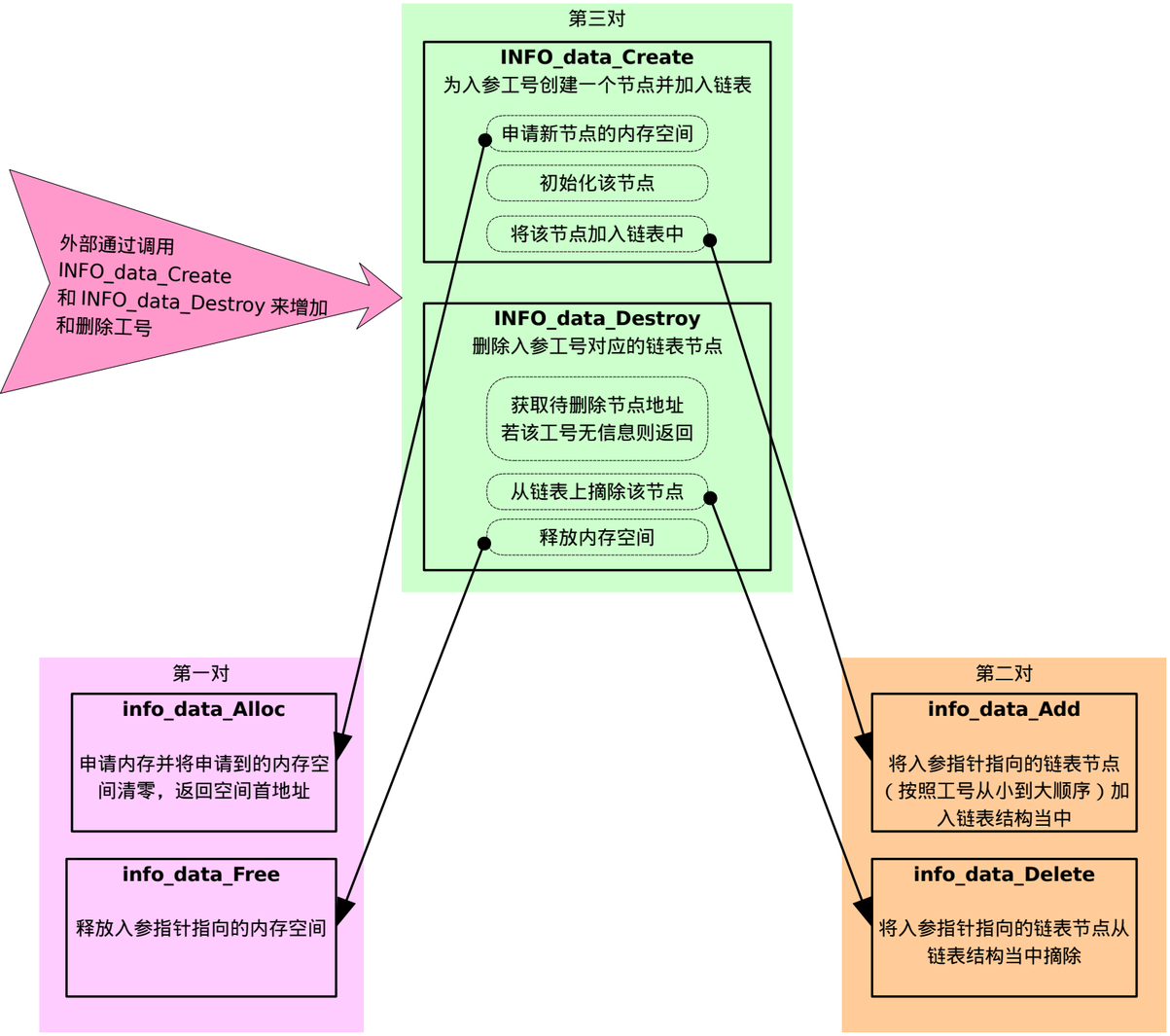
INFO_dbg_Get
获取调试开关

INFO_dbg_Print
打印调试信息


main 函数
INFO_Init
INFO_data_Init
INFO_Run
INFO_Run
info_main_DisplayMenu
显示菜单
info_main_GetMenuChoice
获取菜单选择
解析用户的菜单
选择是否成功
是
info_main_GetInputStr
获取输入字符串
否
调用 info_proc.c 中的
函数接口执行相应操作
否
info_main_DisplayErrInfo
输出提示信息
用户是否选择
“Exit”
是
INFO_Fini
INFO_data_Fini
程序退出

/* INFO-Q 以下 6 个函数是数据组织的典型封装，为什么提供这些接口，分成几对，层次如何 */

（图中省略了 info_data.c 中的其他函数）



/* INFO-Q 请整理使用 DTQ 所遇到的问题和知识点 */

Doubly-linked Tail queue

A doubly-linked tail queue is headed by a pair of pointers, one to the head
of the list and the other to the tail of the list. The elements are doubly linked so that an arbitrary element can be removed without a need to traverse the list. New elements can be added to the list before or after an existing element, at the head of the list, or at the end of the list. A doubly-linked tail queue may be traversed in either direction.

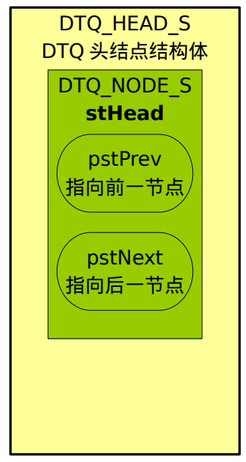
双向连接尾队列（ Doubly-linked Tail queue, DTQ ）

DTQ 以一对指针开头，一个指向链表头节点，一个指向链表尾节点。
DTQ 节点是双向连接的，因此删除一个节点时不必遍历链表。
新节点可以在一个现有节点的前面或后面插入，也可以在链表头或链表尾加入。
DTQ 可以按两个方向遍历。

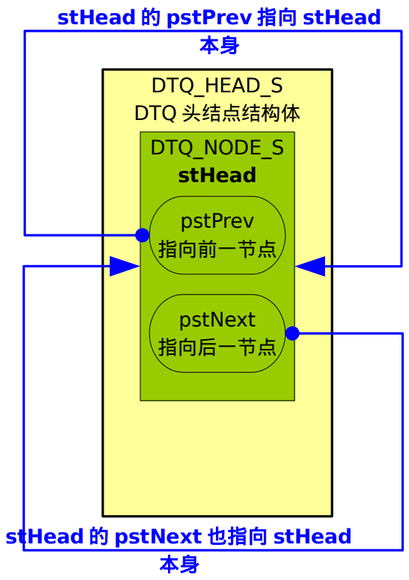
用户声明一个 DTQ_HEAD_S 变量作为链表的哨兵节点，并在用户自定义的数据结构体中包含一个 DTQ_NODE_S 类型成员，就可以通过 DTQ 提供的接口将自定义的数据结构体节点组织成一个双向链表，并进行各种操作。

这样的一个数据组织，对于 DTQ_NODE_S 型节点进行遍历时，它是一个环状链表；但对于 user structure 节点进行遍历时，它的对外表现并不是环形的，例如对最后一个 user structure 节点取后一个节点时，会返回空，而非第一个 user structure 节点。

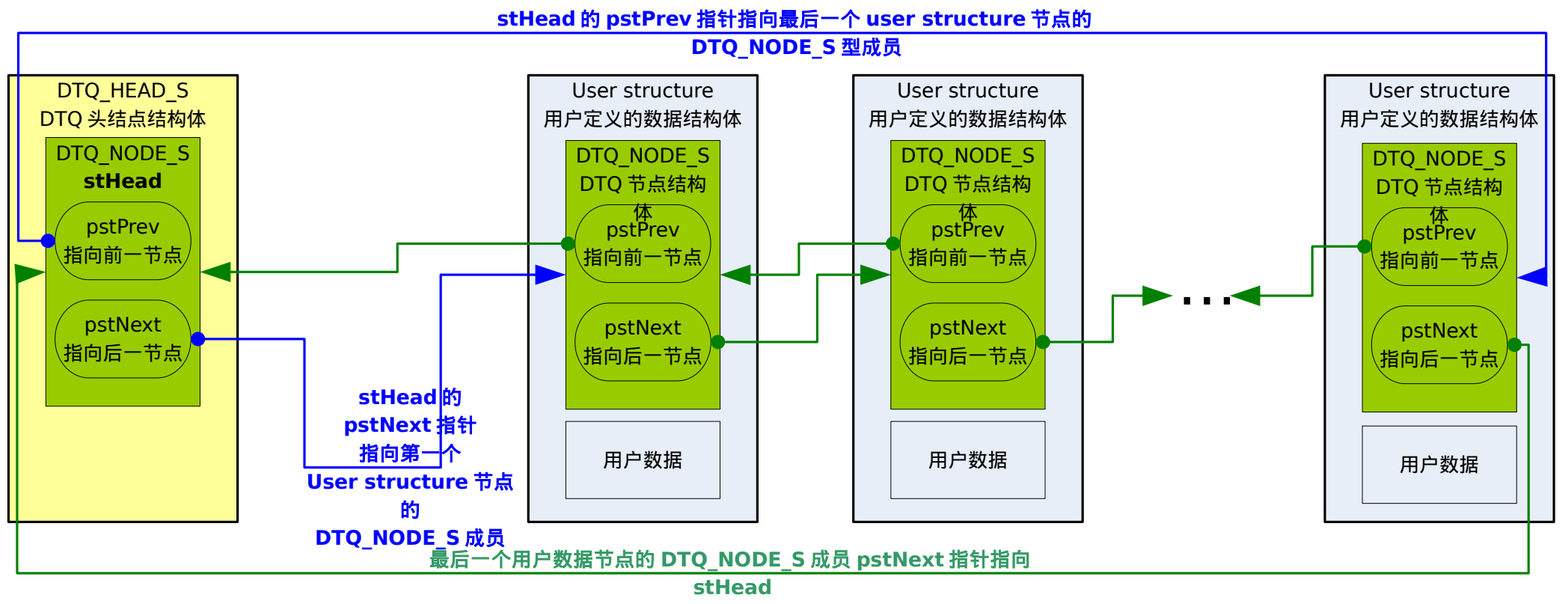
数据组织为空的情况（未执行 DTQ_Init 函数时）
stHead 的 pstPrev 、 pstNext 指针都为空



数据组织为空的情况（执行 DTQ_Init 函数之后）



数据组织不为空的情况



DTQ 提供的函数接口

DTQ 链表提供的函数接口，入参都是 DTQ_HEAD_S 节点指针及 user structure 节点中的 DTQ_NODE_S 成员指针。
仅使用这些函数接口，只能通过对 DTQ_NODE_S 成员指针来间接操作用户数据组织的结构，
不能直接获得 user structure 节点指针，因此也不能直接操作用户数据内容。

初始化 / 去初始化

DTQ_Init
链表结构初始化

DTQ_NodeInit
节点初始化

DTQ_FreeAll
用入参函数清空入参链表

事实上这个函数只是通过 DTQ_FOREACH_SAFE 宏遍历了所有 user structure 的 DTQ_NODE_S 成员，并在遍历后调用 DTQ_Init 将 DTQ_HEAD_S 节点初始化。释放内存及其他处理是需要由入参函数来实现的。传递给入参函数的指针是链表中所有 user structure 中的 DTQ_NODE_S 成员指针。（不会传入 stHead 的指针。）

获取链表状态

DTQ_IsEmpty
链表结构是否为空

DTQ_IsEndOfQ
判断入参节点是否为 stHead

获取特定节点地址

DTQ_First
返回入参链表中第一个 user structure 节点的 DTQ_NODE_S 成员指针

DTQ_Last
返回入参链表中最后一个 user structure 节点的 DTQ_NODE_S 成员指针

DTQ_Prev
返回入参 DTQ_NODE_S 节点的 pstPrev 指针

DTQ_Next
返回入参 DTQ_NODE_S 节点的 pstNext 指针

增加节点

DTQ_AddBefore
将一个 DTQ_NODE_S 节点插入到入参指定的 DTQ_NODE_S 节点前面

DTQ_AddAfter
将一个 DTQ_NODE_S 节点插入入参指定的 DTQ_NODE_S 节点后面

DTQ_AddHead
将入参 DTQ_NODE_S 节点插入到入参链表的 DTQ_HEAD_S 节点后面第一个位置

DTQ_AddTail
将入参 DTQ_NODE_S 节点插入到入参链表的 DTQ_HEAD_S 节点前面第一个位置

DTQ_Append
将一个链表连接到另一个链表尾

删除节点
（只摘除相应节点，并不释放内存）

DTQ_Del
将入参 DTQ_NODE_S 节点从链表结构中摘除

DTQ_DelHead
将入参链表 DTQ_HEAD_S 后面第一个节点删除，并返回该节点指针。如果链表为空则返回 NULL

DTQ_DelTail
将入参链表 DTQ_HEAD_S 后面第一个节点删除，并返回该节点指针。如果链表为空则返回 NULL

DTQ 提供的宏

(1) 名字中带有 ENTRY 的宏进行的是对 user structure 节点的操作，根据 user structure 的结构体类型名 type 、该结构体中 DTQ_NODE_S 结构的成员名 member ，得到某个 DTQ_NODE_S 节点所在的 user structure 节点地址。这些宏使得调用者能够遍历并操作各 user structure 节点，例如释放内存，获取 user structure 中的用户数据等。
(2) 名字中带有 SAFE 的宏在遍历时，除了循环变量以外，还会保存链表中该循环变量的下一个节点指针（从后向前遍历时为上一个节点指针）。每次循环时，都先保存下一个（上一个）节点指针，因此对循环变量的操作并不会影响下一次循环。例如清空链表时，每次循环都 free 循环变量指向的地址，这并不会影响到下一次循环。

循环遍历 DTQ_NODE_S 节点

以 DTQ_NODE_S * pstNode 为循环变量，
遍历 DTQ_HEAD_S * pstList 指向的 DTQ 链表。

DTQ_FOREACH(pstList, pstNode)
以 pstNode 作为循环变量，从前向后遍历所有 user structure 中的 DTQ_NODE_S 成员。循环结束后 pstNode 指向 stHead 。

DTQ_FOREACH_SAFE(pstList, pstNode, pstNext)
以 pstNode 作为循环变量，从前向后遍历所有 user structure 中的 DTQ_NODE_S 成员。循环时 pstNext 始终指向 pstNode 的后一个 DTQ_NODE_S 节点，循环结束后 pstNode 和 pstNext 都指向 stHead 。

DTQ_FOREACH_REVERSE(pstList, pstNode)
以 pstNode 作为循环变量，从后向前遍历所有 user structure 中的 DTQ_NODE_S 成员。循环结束后 pstNode 指向 stHead 。

DTQ_FOREACH_REVERSE_SAFE(pstList, pstNode, pstPrev)
以 pstNode 作为循环变量，从后向前遍历 pstList 指向的 DTQ 链表。循环时 pstPrev 始终指向 pstNode 的前一个 DTQ_NODE_S 节点，循环结束后 pstNode 和 pstPrev 都指向 stHead 。

获取特定的 user structure 节点指针

DTQ_ENTRY(ptr, type, member)
返回 DTQ_NODE_S * ptr 所在的 user structure 节点地址

DTQ_ENTRY_FIRST(pstList, type, member)
返回链表中第一个 user structure 节点地址

DTQ_ENTRY_LAST(pstList, type, member)
返回链表中最后一个 user structure 节点地址

DTQ_ENTRY_NEXT(pstList, pstEntry, member)
返回入参 user structure 型节点 pstEntry 的后一个 user structure 地址。如果 pstEntry 已经是最后一个 user structure ，则返回空值。

DTQ_ENTRY_PREV(pstList, pstEntry, member)
返回入参 user structure 型节点 pstEntry 的前一个 user structure 地址。如果 pstEntry 已经是第一个 user structure ，则返回空值。

循环遍历 user structure 节点

DTQ_FOREACH_ENTRY(pstList, pstEntry, member)
以 user structure 型指针 pstEntry 作为循环变量，从前向后遍历 pstList 指向的 user structure 链表。循环结束后 pstEntry 为空。

DTQ_FOREACH_ENTRY_SAFE(pstList, pstEntry, pstNextEntry, member)
以 user structure 型指针 pstEntry 作为循环变量，从前向后遍历 pstList 指向的 user structure 链表。循环时 pstNextEntry 始终指向 pstEntry 的后一个 user structure 节点， pstEntry 为最后一个 user structure 节点时 pstNextEntry 为空。循环结束后 pstEntry 和 pstNextEntry 都为空。

DTQ_FOREACH_ENTRY_REVERSE(pstList, pstEntry, member)
以 user structure 型指针 pstEntry 作为循环变量，从后向前遍历 pstList 指向的 user structure 链表。循环结束后 pstEntry 为空。

DTQ_FOREACH_ENTRY_REVERSE_SAFE(pstList, pstEntry, pstPrevEntry, member)
以 user structure 型指针 pstEntry 作为循环变量，从后向前遍历 pstList 指向的 user structure 链表。循环时 pstPrevEntry 始终指向 pstEntry 的前一个 user structure 节点， pstEntry 为第一个 user structure 节点时 pstPrevEntry 为空。循环结束后 pstEntry 和 pstPrevEntry 都为空。

/* INFO-Q 为什么使用函数指针数组，为什么采用这种初始化的方法 */

以菜单选项枚举常量作为数组下标，函数指针数组提供了菜单选项名称字符串和对应的菜单功能处理函数指针之间的 一 一 映射。