version.c

提供两个函数获取当前libuv库的版本：

1.uv_version,获取数字版本
2.uv_version_string获取字符串版本

tree.h

定义伸展树（splay tree）和红黑树（RB tree）

1.伸展树在windows平台未发现使用
2.红黑树相关定义
3.#define RB_HEAD(name, type) 红黑树结构体，内部包含一个根节点指针（比如uvwin.h中的uv_timer_s，其中uv_timer_s中又包含UV_TIMER_PRIVATE_FIELDS宏，该宏展开后有RB_ENTRY(uv_timer_s) tree_entry）

```
#define RB_HEAD(name, type)                                     \
struct name {                                                   \
        struct type *rbh_root; /* root of the tree              \
*/ }
```

b. #define RB_INIT(root) 初始化根节点

```
#define RB_INIT(root) do {                                      \
        (root)->rbh_root = NULL;                                \
    } while (/*CONSTCOND*/ 0)
```

c. #define RB_ENTRY(type) 红黑树节点，包含左、右子树，父节点以及节点颜色
d. 红黑树相关函数的宏定义，主要包括左旋、右旋、删除、插入、上一个节点、下一个节点、最大节点、最小节点、查找等方法，可以参考stl的红黑树（stl通过模板实现支持树节点包含不同类型的数据，共用同一套红黑树的逻辑，而libuv通过宏定义来实现，不同类型的树节点都包含RB_ENTRY宏，而树相关的操作基本上都是针对该宏定义的结构体）

uv-win.h
定义平台相关的内容

1. 定义重叠模型（Overlapped I/O）用到的socket函数指针
   a. typedef int (WSAAPI* LPFN_WSARECV)
   b. typedef int (WSAAPI* LPFN_WSARECVFROM)
2. typedef struct uv_buf_t结构体，能够转换为WSABUF
3. RB_HEAD(uv_timer_tree_s, uv_timer_s); 定义一颗红黑树 uv_timer_tree_s，节点为uv_timer_s

4.#define UV_LOOP_PRIVATE_FIELDS 定义loop(循环)的私有变量

```
/* The loop's I/O completion port * /                                  \
//iocp完成端口句柄
HANDLE iocp;                                                          \
/* The current time according to the event loop. in msecs.            \
*/ //事件循环的当前时间 毫秒
uint64_t time;                                                        \
/* Tail of a single-linked circular queue of pending reqs. If the queue */  \
 /* is empty, tail_ is NULL. If there is only one item, */            \
/* tail_->next_req == tail_ */                                        \
//单向循环队列的尾指针
uv_req_t* pending_reqs_tail;                                          \
/* Head of a single-linked list of closed handles */                  \
//已经关闭的句柄的单向列表的首指针
uv_handle_t* endgame_handles;                                         \
 /* The head of the timers tree */                                    \
//定时器红黑树
struct uv_timer_tree_s timers;                                        \
 /* Lists of active loop (prepare / check / idle) watchers */         \
//活动的循环观察者列表
 uv_prepare_t* prepare_handles;                                       \
uv_check_t* check_handles;                                            \
uv_idle_t* idle_handles;                                              \
/* This pointer will refer to the prepare/check/idle handle whose     \
*/     /* callback is scheduled to be called next. This is needed to  \
allow */    /* safe removal from one of the lists above while that list  \
being */     /* iterated over. */                                     \
//
uv_prepare_t* next_prepare_handle;                                    \
uv_check_t* next_check_handle;                                        \
uv_idle_t* next_idle_handle;                                          \
/* This handle holds the peer sockets for the fast variant of uv_poll_t */  \
SOCKET                                                                \
poll_peer_sockets[UV_MSAFD_PROVIDER_COUNT]; /*                        \
Counter to keep track of active tcp streams */
//活动的tcp流数量                                                     \
unsigned int active_tcp_streams;                                      \
/* Counter to keep track of active udp streams */
//活动的udp流数量                                                     \
unsigned int active_udp_streams;                                      \
/* Counter to started timer */
//已开始的定时器的数量                                                \
uint64_t timer_counter;                                               \
 /* Threadpool */                                                     \
 void* wq[2];                                                         \
uv_mutex_t wq_mutex;
uv_async_t wq_async;
```

5.#define UV_REQ_TYPE_PRIVATE 定义REQ(请求)的类型

6. #define UV_REQ_PRIVATE_FIELDS 定义REQ的私有变量

```
#define UV_REQ_PRIVATE_FIELDS
union {
        /* Used by I/O operations */
        struct {
                OVERLAPPED overlapped;
                size_t  queued_bytes;
        } io;
} u;
struct uv_req_s* next_req;
```

7. #define UV_WRITE_PRIVATE_FIELDS 定义write的私有变量

```
int ipc_header;
uv_buf_t write_buffer;
HANDLE event_handle;
HANDLE wait_handle;
```

8. uv_pipe_accept_s
9. uv_tcp_accept_s
10. uv_read_s
11. uv_stream_connection_fields
12. uv_stream_server_fields
13. UV_STREAM_PRIVATE_FIELDS
14. uv_tcp_server_fields
15. uv_tcp_connection_fields
16. UV_TCP_PRIVATE_FIELDS
17. uv_pipe_server_fields
18. uv_pipe_connection_fields
19. UV_PIPE_PRIVATE_FIELDS
20. UV_TTY_PRIVATE_FIELDS
21. UV_POLL_PRIVATE_FIELDS
22. UV_TIMER_PRIVATE_FIELDS
23. UV_ASYNC_PRIVATE_FIELDS
24. UV_PREPARE_PRIVATE_FIELDS
25. UV_CHECK_PRIVATE_FIELDS
26. UV_IDLE_PRIVATE_FIELDS
27. UV_HANDLE_PRIVATE_FIELDS
28. UV_GETADDRINFO_PRIVATE_FIELDS
29. UV_GETNAMEINFO_PRIVATE_FIELDS
30. UV_PROCESS_PRIVATE_FIELDS
31. UV_FS_PRIVATE_FIELDS
32. UV_WORK_PRIVATE_FIELDS
33. UV_FS_EVENT_PRIVATE_FIELDS
34. UV_SIGNAL_PRIVATE_FIELDS

uv.h

主要是一些宏定义以及函数的声明
1._WIN32平台下，如果定义了BUILDING_UV_SHARED就是导出函数（libuv工程在工程配置"c/c++——预处理器"中定义）：

```
# if defined(BUILDING_UV_SHARED)
/* Building shared library. */
#   define UV_EXTERN __declspec(dllexport)
```

如果定义了USING_UV_SHARED（使用者定义），则是导入函数

2.#define UV_ERRNO_MAP(XX) 定义了错误名以及对应的错误信息的列表，其中XX可以是另一个宏，比如在uv-common.h中的：

```
#define UV_ERR_NAME_GEN(name, _) case UV_ ## name: return #name;
```

配合switch语句使用，返回错误名。其中错误的代码通过枚举定义：

```
typedef   enum {
            #define   XX(code, _) UV_ ## code = UV__ ## code,
            UV_ERRNO_MAP(XX)
            #undef   XX
            UV_ERRNO_MAX = UV__EOF - 1
} uv_errno_t;
```

3.#define UV_HANDLE_TYPE_MAP(XX) ，句柄类型列表，同时参考：

```
typedef   enum {
            UV_UNKNOWN_HANDLE = 0,
            #define   XX(uc, lc)   UV_##uc,
            UV_HANDLE_TYPE_MAP(XX)
            #undef XX
            UV_FILE,
            UV_HANDLE_TYPE_MAX
} uv_handle_type;
```

4.#define UV_REQ_TYPE_MAP(XX)，定义请求类型列表，同时参考：

```
typedef   enum {
            UV_UNKNOWN_REQ = 0,
            #define   XX(uc, lc)   UV_##uc,
            UV_REQ_TYPE_MAP(XX)
            #undef XX
            UV_REQ_TYPE_PRIVATE
            UV_REQ_TYPE_MAX
} uv_req_type;
```

5.定义一些结构体类型，比如typedef struct uv_loop_s uv_loop_t;，主要有句柄结构体以及请求结构体
6.定义一些配置：

```
typedef enum {
UV_LOOP_BLOCK_SIGNAL }
uv_loop_option;
typedef enum {


            UV_RUN_DEFAULT = 0,
            UV_RUN_ONCE,
            UV_RUN_NOWAIT
} uv_run_mode;
```

7. 函数声明，以及声明各种函数指针变量
8. 定义各种结构体