# plist

## plist.h

1. enum plist\_type ,显示plist node的类型；
2. plist\_t **plist\_new\_dict**(void); 创建一个PLIST\_DICT类型的plist的对象；
3. plist\_t **plist\_new\_array**(void); 创建一个PLIST\_ARRAY类型的plist的对象；
4. plist\_t **plist\_new\_string**(const char \*val); 创建一个PLIST\_STRING类型的对象；
5. plist\_t **plist\_new\_bool**(uint8\_t val); 创建一个PLIST\_BOOLEN类型的对象；
6. plist\_t **plist\_new\_uint**(uint64\_t val); 创建一个PLIST\_UINT类型的对象；
7. plist\_t **plist\_new\_real**(double val); 创建一个PLIST\_REAL 类型的对象；
8. plist\_t **plist\_new\_data**(const char \*val, uint64\_t length); 创建一个PLIST\_DATA类型的对象；
9. plist\_t **plist\_new\_date**(int32\_t sec, int32\_t usec); 创建一个PLIST\_DATE类型的对象；
10. plist\_t **plist\_new\_uid**(uint64\_t val); 创建一个PLIST\_UID类型的对象；
11. void **plist\_free**plist\_t plist); 释放一个plist对象；
12. plist\_t **plist\_copy**(plist\_t node); 拷贝node中的数据，创建一个新的plist\_t；

### Array

1. uint32\_t **plist\_array\_get\_size**(plist\_t node); 获取array的大小；
2. plist\_t **plist\_array\_get\_item**(plist\_t node, uint32\_t n); 获取第n个node；
3. uint32\_t **plist\_array\_get\_item\_index**(plist\_t node); 获得node对应的index；
4. void **plist\_array\_set\_item**(plist\_t node, plist\_t item, uint32\_t n); node的第n个节点设置为item;
5. void **plist\_array\_append\_item**(plist\_t node, plist\_t item); 将item添加到node上；
6. void **plist\_array\_insert\_item**(plist\_t node, plist\_t item, uint32\_t n); 将item插入到node的第n个结点；
7. void **plist\_array\_remove\_item**(plist\_t node, uint32\_t n); 在node中删除第n个结点；
8. void **plist\_array\_new\_iter**(plist\_t node, plist\_array\_iter \*iter); 创建一个node的迭代器iter，iter使用结束使用标准free释放；
9. void **plist\_array\_next\_item**(plist\_t node, plist\_array\_iter iter, plist\_t \*item); 将迭代器前进至item的下一个结点位置；

### dictionary

1. uint32\_t **plist\_dict\_get\_size**(plist\_t node); 获取字典node的大小；
2. void **plist\_dict\_new\_iter**(plist\_t node, plist\_dict\_iter \*iter); 创建一个字典node的迭代器iter；
3. void **plist\_dict\_next\_item**(plist\_t node, plist\_dict\_iter iter, char \*\*key, plist\_t \*val); dict迭代器前进；
4. plist\_t **plist\_dict\_get\_item**(plist\_t node, const char\* key); 通过key获取到node中的value;
5. plist\_t **plist\_dict\_item\_get\_key**(plist\_t node); 获取node的key；
6. void **plist\_dict\_set\_item**(plist\_t node, const char\* key, plist\_t item); 更新key对应的item；
7. void **plist\_dict\_remove\_item**(plist\_t node, const char\* key); 在dict node中删除key对应的item;
8. void **plist\_dict\_merge**(plist\_t \*target, plist\_t source); 将source中的键值与target中的合并，出现冲突则使用source中的重写；

### Getters

1. plist\_type **plist\_get\_node\_type**(plist\_t node); 获取node的类型；
2. void **plist\_get\_key\_val**(plist\_t node, char \*\*val); 对于PLIST\_KEY的plist而言，获取node结点的value:
3. void **plist\_get\_string\_val**(plist\_t node, char \*\*val); 对于PLIST\_STRING的plist而言，获取其node结点的value;
4. const char\* **plist\_get\_string\_ptr**(plist\_t node, uint64\_t\* length); 返回string node的指针，并其长度；
5. void **plist\_get\_bool\_val**(plist\_t node, uint8\_t \* val); 获取PLIST\_BOOLEN类型node的值；
6. void **plist\_get\_uint\_val**(plist\_t node, uint64\_t \* val); 获取PLIST\_UINT类型node的值；
7. void **plist\_get\_real\_val**(plist\_t node, double \*val); 获取PLIST\_REAL类型node的值；
8. void **plist\_get\_data\_val**(plist\_t node, char \*\*val, uint64\_t \* length); 获取PLIST\_DATA类型node的数据，存储到val中，大小为length；
9. const char\* **plist\_get\_data\_ptr**(plist\_t node, uint64\_t\* length); 获取PLIST\_DATA类型node的指针，并计算其大小存储在length中；
10. void **plist\_get\_date\_val**(plist\_t node, int32\_t \* sec, int32\_t \* usec); 获取PLIST\_DATE类型node的数据，sec为年月日，usec为微秒；
11. void plist\_get\_uid\_val(plist\_t node, uint64\_t \* val); 获取PLIST\_UID类型的node，存储在val中；

### Setters

1. void **plist\_set\_key\_val**(plist\_t node, const char \*val); 在PLIST\_KEY类型的node中设置key；
2. void **plist\_set\_string\_val**(plist\_t node, const char \*val); 设置PLIST\_STRING类型的node数据为val；
3. void **plist\_set\_bool\_val**(plist\_t node, uint8\_t val); 设置PLIST\_BOOLEN类型的node数据为val；
4. void **plist\_set\_uint\_val**(plist\_t node, uint64\_t val); 设置PLIST\_UINT类型的node数据为val；
5. void **plist\_set\_real\_val**(plist\_t node, double val); 设置PLIST\_REAL类型的node数据为val；
6. void **plist\_set\_data\_val**(plist\_t node, const char \*val, uint64\_t length); 设置PLIST\_DATA类型的node数据为val，大小为length；
7. void **plist\_set\_date\_val**(plist\_t node, int32\_t sec, int32\_t usec); 设置PLIST\_DATE类型的node，其日期为sec，时间为usec；
8. void **plist\_set\_uid\_val**(plist\_t node, uint64\_t val); 设置PLIST\_UID类型的node数据为val；

### Imports & Export

1. void **plist\_to\_xml**(plist\_t plist, char \*\*plist\_xml, uint32\_t \* length); 将plist转换为xml格式，xml格式为c\_string, 大小为length；
2. void **plist\_to\_xml\_free**(char \*plist\_xml); 释放xml结构；
3. void **plist\_from\_xml**(const char \*plist\_xml, uint32\_t length, plist\_t \* plist); 将xml转换为plist格式，大小为length；
4. void **plist\_to\_bin**(plist\_t plist, char \*\*plist\_bin, uint32\_t \* length); 将plist格式的plist转换为二进制格式，长度为length；
5. void **plist\_to\_bin\_free**(char \*plist\_bin); 释放二进制结构；
6. void **plist\_from\_bin**(const char \*plist\_bin, uint32\_t length, plist\_t \* plist); 将二进制结构的plist\_bin转换为plist，长度为length；
7. void **plist\_from\_memory**(const char \*plist\_data, uint32\_t length, plist\_t \* plist); 从memory data中提出plist结构数据；
8. int **plist\_is\_binary**(const char \*plist\_data, uint32\_t length); 查看plist\_data时二进制格式还是xml格式；
9. int **plist\_bool\_val\_is\_true**(plist\_t boolnode); 判断boolnode是否是Bool类型；
10. char **plist\_compare\_node\_value**(plist\_t node\_l, plist\_t node\_r); 查看node\_l 与 node\_r是否匹配；