iterator（迭代器模式）又称Cursor（游标模式）

简述：

provide a way to access the elements of aggregate object sequentially without exposing its underlying representation（提供一种方式去访问容器的每一个节点，在不暴露容器的底层表示方式的前提下）

原理：

建立一个iterator接口，并利用不同容器的API（list，array等）实现这套接口，既可以自如的对容器里面的内容做相应的操作。

接口：

```
struct _Iterator;
typedef struct _Iterator Iterator;

typedef Ret  (*IteratorSetFunc)(Iterator* thiz, void* data);
typedef Ret  (*IteratorGetFunc)(Iterator* thiz, void** data);
typedef Ret  (*IteratorNextFunc)(Iterator* thiz);
typedef Ret  (*IteratorPrevFunc)(Iterator* thiz);
typedef Ret  (*IteratorAdvanceFunc)(Iterator* thiz, int offset);
typedef int  (*IteratorOffsetFunc)(Iterator* thiz);
typedef Ret  (*IteratorCloneFunc)(Iterator* thiz, Iterator** cloned);
typedef void (*IteratorDestroyFunc)(Iterator* thiz);

struct _Iterator
{
        IteratorSetFunc     set;
        IteratorGetFunc     get;
        IteratorNextFunc    next;
        IteratorPrevFunc    prev;
        IteratorAdvanceFunc advance;
        IteratorCloneFunc   clone;
        IteratorOffsetFunc  offset;
        IteratorDestroyFunc destroy;

        char priv[0];
};
```

应用：

同过应用不同容器提供的接口实现上面表示的接口，就可以利用这套接口操作容器的内容了（不同的容器，实现不同的iterator）。一下例子，用iterator实现链表数据的倒序操作。

```
#include "iterator.h"
```

```
Ret invert(Iterator* forward, Iterator* backward)
{
	void* data1 = NULL;
	void* data2 = NULL;
	return_val_if_fail(forward != NULL && backward != NULL, RET_INVALID_PARAMS);

	for(; iterator_offset(forward) < iterator_offset(backward); iterator_next(forward), iterator_prev(backward))
	{
		iterator_get(forward, &data1);
		iterator_get(backward, &data2);
		iterator_set(forward, data2);
		iterator_set(backward, data1);
	}

	return RET_OK;
}

#ifdef INVERT_TEST
#include "dlist.h"
#include "dlist_iterator.h"
#include "test_helper.c"

int main(int argc, char* argv[])
{
	int i = 0;
	int n = 101;
	int last = n - 1;
	DList* dlist = dlist_create(NULL, NULL);

	for(i = 0; i < n; i++)
	{
		dlist_append(dlist, (void*)i);			/* 创建列表 */
	}

	Iterator* forward = dlist_iterator_create(dlist);	/* 创建从头往后走的光标（iterator） */
	Iterator* backward = dlist_iterator_create(dlist);	/* 创建从后往前走的光标（iterator） */

	iterator_advance(backward, last);			/* 把从后往前走的光标移动到最后 */
	invert(forward, backward);
	dlist_foreach(dlist, check_and_dec_int, &last);
	iterator_destroy(forward);
	iterator_destroy(backward);
	dlist_destroy(dlist);

	return 0;
}
#endif/*INVERT_TEST*/
```