

|  |
| --- |
| MineWar |
| 单体测试报告 |
| 软件开发系列文档 |
| Ver 1.0.0  zhaoyg  2016/9/20 |

|  |
| --- |
| All rights reserved, 2016 |

变更履历

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 版本 | 变更内容 | 变更人 | 日期 |
| 1 | 0.7.0 | 追加测试系统说明 | zhaoyg | 2016-7-18 |
| 2 | 0.8.0 | 追加双向链表测试用例及结果 | XB | 2016-7-22 |
| 3 | 0.8.1 | 追加ViewStack测试用例及结果 | XC | 2016-7-25 |
| 4 | 0.8.2 | 追加Map测试用例及结果 | XA | 2016-8-29 |
| 5 | 1.0.0 | 追加\_T\_ASSERT\_GOTO说明 | zhaoyg | 2016-9-20 |
| 6 |  |  |  |  |
| 7 |  |  |  |  |
| 8 |  |  |  |  |
| 9 |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[目录 3](#_Toc533364235)

[1 测试系统 4](#_Toc533364236)

[1.1 测试框架 4](#_Toc533364237)

[1.2 测试API 4](#_Toc533364238)

[1.3 用例示例 6](#_Toc533364239)

[2 测试用例及结果 7](#_Toc533364240)

[2.1 DoubleLinkList 7](#_Toc533364241)

[2.2 Map 12](#_Toc533364242)

[2.3 ViewStack 17](#_Toc533364243)

图表目录

[图 1‑1 测试框架 4](#_Toc533360901)

[表 1‑1 测试用API 4](#_Toc533360902)

# 测试系统

## 测试框架

测试系统包括用例执行器，测试用例和测试API，测试结果由API自动输出。

* 用例执行器：

调用测试用例的模块，可以在main函数中实现；

* 测试用例：

每一个用例对应一个用例函数，在用例中调用测试API，实现测试过程。包括测试目的、前提、步骤、操作参数返回值、操作结果判定；

* 测试API：

测试用接口函数，[测试API](#_测试API)。



图 ‑1 测试框架

## 测试API

本文档为MineWar应用相关设计，包括开发环境、应用框架、UI库、描画库、Log功能、游戏核心数据结构和算法、主要流程和时序等的设计。

表 ‑1 测试用API

|  |  |  |
| --- | --- | --- |
| API | 参数 | 用途 |
| \_T\_PURPOSE(str) | str，字符串 | 用于输出测试目的 |
| \_T\_PRECOND(str) | str，字符串 | 用于输出前提条件 |
| \_T\_PARAM(format, ...) | format及动态参数，与printf类似 | 用于输出操作使用的参数 |
| \_T\_RET(format, ...) | format及动态参数，与printf类似 | 用于输出操作返回值 |
| \_T\_INFO(format, ...) | format及动态参数，与printf类似 | 用于输出额外的信息 |
| \_T\_OPER(str) | str，操作 | 用于输出操作 |
| \_T\_STEP(n) | n，整型 | 用于表示测试步骤序号 |
| \_T\_END\_OK() | 无 | 用于表示用例判定OK |
| \_T\_ASSERT(exp) | exp，表达式 | 用于执行判定，表达式值为FALSE时，用例判定NG，之后退出用例函数 |
| \_T\_ASSERT\_GOTO(exp,lable) | exp，表达式  label，表达式值为FALSE时，goto到该label | 用于在用例判定NG时执行必要的处理，例如释放动态内存 |

## 用例示例

static void case\_03\_001(void)

{

dlist\_t \*list = NULL;

tdata\_t \*data1 = NEW\_T(tdata\_t);

tdata\_t \*pdata;

int ret;

dlist\_t \*p;

int i;

\_T\_CASE("DLIST-03-001");

\_T\_PURPOSE("validate dlist\_ins.");

\_T\_PRECOND("list is empty.");

\_T\_STEP(1);

data1->a = 1;

\_T\_PARAM("data1->a=%d", data1->a);

\_T\_STEP(2);

\_T\_OPER("ret = dlist\_ins(&list, NULL, data1, cmp\_func)");

ret = dlist\_ins(&list, NULL, data1, cmp\_func);

\_T\_ASSERT\_GOTO(ret != 0, end);

\_T\_ASSERT\_GOTO(dlist\_get\_size(list) == 1, end);

p = list;

i = 0;

while(p != NULL){

pdata = p->data;

if(i == 0){

\_T\_ASSERT\_GOTO(pdata->a == 1, end);

}

i++;

p = p->next;

}

\_T\_END\_OK();

end:

dlist\_clear(&list);

}

# 测试用例及结果

为了便于维护及减少维护工作量，粘贴测试Log即可。

## DoubleLinkList

----------------------------------------

<Case> DLIST-01-001

<Purpose> validate dlist\_push\_front.

<Precondition> list is empty.

<Step> 1

<Param> data=00000000

<Step> 2

<Operation> ret = dlist\_push\_front(&list, data)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 1 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-01-002

<Purpose> validate dlist\_push\_front.

<Precondition> list is not empty.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_push\_front(&list, data1)

<Step> 2

<Param> data2->a=2

<Operation> ret = dlist\_push\_front(&list, data2)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 2 -> TRUE

<Assert> pdata->a == 2 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-02-001

<Purpose> validate dlist\_push\_back.

<Precondition> list is empty.

<Step> 1

<Param> data=00000000

<Step> 2

<Operation> ret = dlist\_push\_back(&list, data)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 1 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-02-002

<Purpose> validate dlist\_push\_back.

<Precondition> list is not empty.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_push\_back(&list, data1)

<Step> 2

<Param> dataa->a=2

<Operation> ret = dlist\_push\_back(&list, data2)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 2 -> TRUE

<Assert> pdata->a == 2 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-03-001

<Purpose> validate dlist\_ins.

<Precondition> list is empty.

<Step> 1

<Param> data1->a=1

<Step> 2

<Operation> ret = dlist\_ins(&list, NULL, data1, cmp\_func)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 1 -> TRUE

<Assert> pdata->a == 1 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-03-002

<Purpose> validate dlist\_ins.

<Precondition> list is not empty, data match.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_ins(&list, NULL, data1, cmp\_func)

<Step> 2

<Param> dataa->a=1

<Operation> ret = dlist\_ins(&list, data2, data3, cmp\_func)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 2 -> TRUE

<Assert> pdata->a == 3 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-03-003

<Purpose> validate dlist\_ins.

<Precondition> list is not empty, data not match.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_ins(&list, NULL, data1, cmp\_func)

<Step> 2

<Param> dataa->a=2

<Operation> ret = dlist\_ins(&list, data2, data3, cmp\_func)

<Assert> ret == 0 -> TRUE

<Assert> dlist\_get\_size(list) == 1 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-04-001

<Purpose> validate dlist\_del.

<Precondition> list is empty.

<Step> 1

<Operation> ret = dlist\_del(&list, NULL, cmp\_func)

<Assert> ret == 0 -> TRUE

<Assert> dlist\_get\_size(list) == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-04-002

<Purpose> validate dlist\_del.

<Precondition> list is not empty, data match.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_push\_back(&list, data1)

<Step> 2

<Param> dataa->a=2

<Operation> ret = dlist\_push\_back(&list, data2)

<Step> 3

<Param> data3->a=2

<Operation> ret = dlist\_del(&list, data3, cmp\_func)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 1 -> TRUE

<Step> 4

<Param> data3->a=1

<Operation> ret = dlist\_del(&list, data3, cmp\_func)

<Assert> ret != 0 -> TRUE

<Assert> dlist\_get\_size(list) == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-04-003

<Purpose> validate dlist\_del.

<Precondition> list is not empty, data not match.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_push\_back(&list, data1)

<Step> 2

<Param> dataa->a=2

<Operation> ret = dlist\_push\_back(&list, data2)

<Step> 3

<Param> data3->a=3

<Operation> ret = dlist\_del(&list, data3, cmp\_func)

<Assert> ret == 0 -> TRUE

<Assert> dlist\_get\_size(list) == 2 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-05-001

<Purpose> validate dlist\_find.

<Precondition> list is empty.

<Step> 1

<Operation> ret = dlist\_find(list, NULL, cmp\_func)

<Assert> ret == NULL -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-05-002

<Purpose> validate dlist\_find.

<Precondition> list is not empty, data match.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_push\_back(&list, data1)

<Step> 2

<Param> dataa->a=2

<Operation> ret = dlist\_push\_back(&list, data2)

<Step> 3

<Param> data3->a=2

<Operation> pdata = dlist\_find(list, data3, cmp\_func)

<Assert> pdata != NULL -> TRUE

<Assert> pdata->a == 2 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-05-003

<Purpose> validate dlist\_find.

<Precondition> list is not empty, data not match.

<Step> 1

<Param> data1->a=1

<Operation> ret = dlist\_push\_back(&list, data1)

<Step> 2

<Param> dataa->a=2

<Operation> ret = dlist\_push\_back(&list, data2)

<Step> 3

<Param> data3->a=3

<Operation> pdata = dlist\_find(list, data3, cmp\_func)

<Assert> pdata == NULL -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-06-001

<Purpose> validate dlist\_clear.

<Precondition> list is empty.

<Step> 1

<Operation> dlist\_clear(&list)

<Assert> list == NULL -> TRUE

<Assert> dlist\_get\_size(list) == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> DLIST-06-002

<Purpose> validate dlist\_clear.

<Precondition> list is not empty.

<Step> 1

<Param> data->a=1

<Operation> ret = dlist\_push\_back(&list, data)

<Assert> list != NULL -> TRUE

<Assert> dlist\_get\_size(list) == 1 -> TRUE

<Step> 2

<Operation> dlist\_clear(&list)

<Assert> list == NULL -> TRUE

<Assert> dlist\_get\_size(list) == 0 -> TRUE

<Result> OK

----------------------------------------

DLIST-01-001 OK

DLIST-01-002 OK

DLIST-02-001 OK

DLIST-02-002 OK

DLIST-03-001 OK

DLIST-03-002 OK

DLIST-03-003 OK

DLIST-04-001 OK

DLIST-04-002 OK

DLIST-04-003 OK

DLIST-05-001 OK

DLIST-05-002 OK

DLIST-05-003 OK

DLIST-06-001 OK

DLIST-06-002 OK

## Map

----------------------------------------

<Case> MAP-01-001

<Purpose> validate map\_save and map\_load.

<Precondition> -

<Step> 1

<Operation> map\_save("testmap-01.dat", test\_map1)

[Map/I] map save, file=testmap-01.dat

<Step> 2

<Operation> map\_load("testmap-01.dat", tmp\_map)

[Map/I] map load, file=testmap-01.dat

<Assert> memcmp(test\_map1, tmp\_map, sizeof(test\_map1)) == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-002

<Purpose> validate map\_get.

<Precondition> -

<Step> 1

<Operation> map\_save("Map1.dat", test\_map1)

[Map/I] map save, file=Map1.dat

<Step> 2

<Operation> map\_load("Map1.dat", &all\_map\_data[0\*MAP\_H\*MAP\_W])

[Map/I] map load, file=Map1.dat

<Step> 3

<Operation> map\_get(0, tmp\_map)

<Assert> memcmp(test\_map1, tmp\_map, sizeof(test\_map1)) == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-003

<Purpose> validate map\_fill(1 entrance).

<Precondition> -

<Step> 1

<Operation> map\_save("Map1.dat", test\_map1)

[Map/I] map save, file=Map1.dat

<Step> 2

<Operation> map\_load("Map1.dat", &all\_map\_data[0\*MAP\_H\*MAP\_W])

[Map/I] map load, file=Map1.dat

<Step> 3

<Operation> map\_fill(0, tmp\_map, entrances, &entrance\_total)

<Assert> memcmp(filled\_map1, tmp\_map, sizeof(filled\_map1)) == 0 -> TRUE

<Assert> entrance\_total == 1 -> TRUE

<Assert> (entrances[0].x == 0)&&(entrances[0].y == 5) -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-004

<Purpose> validate map\_fill(4 entrances).

<Precondition> -

<Step> 1

<Operation> map\_save("Map2.dat", test\_map2)

[Map/I] map save, file=Map2.dat

<Step> 2

<Operation> map\_load("Map2.dat", &all\_map\_data[1\*MAP\_H\*MAP\_W])

[Map/I] map load, file=Map2.dat

<Step> 3

<Operation> map\_fill(1, tmp\_map, entrances, &entrance\_total)

<Assert> memcmp(filled\_map2, tmp\_map, sizeof(filled\_map2)) == 0 -> TRUE

<Assert> entrance\_total == 4 -> TRUE

<Assert> (entrances[0].x == 0)&&(entrances[0].y == 0) -> TRUE

<Assert> (entrances[1].x == 13)&&(entrances[1].y == 0) -> TRUE

<Assert> (entrances[2].x == 0)&&(entrances[2].y == 8) -> TRUE

<Assert> (entrances[3].x == 13)&&(entrances[3].y == 8) -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-005

<Purpose> validate map\_get\_move\_direction(1 entrance/1 end).

<Precondition> -

<Step> 1

<Operation> map\_save("Map1.dat", test\_map1)

[Map/I] map save, file=Map1.dat

<Step> 2

<Operation> map\_load("Map1.dat", &all\_map\_data[0\*MAP\_H\*MAP\_W])

[Map/I] map load, file=Map1.dat

<Step> 3

<Operation> map\_get\_move\_direction(0, tmp\_direction)

[Map/I] get direction, index=0

<Assert> memcmp(direction\_1, tmp\_direction, sizeof(direction\_1)) == 0 -> T

RUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-006

<Purpose> validate map\_get\_move\_direction(4 entrances/2 ends).

<Precondition> -

<Step> 1

<Operation> map\_save("Map2.dat", test\_map2)

[Map/I] map save, file=Map2.dat

<Step> 2

<Operation> map\_load("Map2.dat", &all\_map\_data[1\*MAP\_H\*MAP\_W])

[Map/I] map load, file=Map2.dat

<Step> 3

<Operation> map\_get\_move\_direction(1, tmp\_direction)

[Map/I] get direction, index=1

<Assert> memcmp(direction\_2, tmp\_direction, sizeof(direction\_2)) == 0 -> T

RUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-007

<Purpose> validate map\_is\_in\_range.

<Precondition> -

<Step> 1

<Operation> ret = map\_is\_in\_range(0, 0)

<Assert> ret == 1 -> TRUE

<Step> 2

<Operation> ret = map\_is\_in\_range(-1, 0)

<Assert> ret == 0 -> TRUE

<Step> 3

<Operation> ret = map\_is\_in\_range(1, 0)

<Assert> ret == 1 -> TRUE

<Step> 4

<Operation> ret = map\_is\_in\_range(7, 0)

<Assert> ret == 1 -> TRUE

<Step> 5

<Operation> ret = map\_is\_in\_range(7, -1)

<Assert> ret == 0 -> TRUE

<Step> 6

<Operation> ret = map\_is\_in\_range(7, 1)

<Assert> ret == 1 -> TRUE

<Step> 7

<Operation> ret = map\_is\_in\_range(12, 0)

<Assert> ret == 1 -> TRUE

<Step> 8

<Operation> ret = map\_is\_in\_range(13, 0)

<Assert> ret == 1 -> TRUE

<Step> 9

<Operation> ret = map\_is\_in\_range(14, 0)

<Assert> ret == 0 -> TRUE

<Step> 10

<Operation> ret = map\_is\_in\_range(7, 7)

<Assert> ret == 1 -> TRUE

<Step> 11

<Operation> ret = map\_is\_in\_range(7, 8)

<Assert> ret == 1 -> TRUE

<Step> 12

<Operation> ret = map\_is\_in\_range(7, 9)

<Assert> ret == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-008

<Purpose> validate map\_is\_mining\_enable.

<Precondition> -

<Step> 1

<Info> road-mine ae

<Operation> ret = map\_is\_mining\_enable(1, 5, MINE\_AE, map1)

<Assert> ret == 1 -> TRUE

<Step> 2

<Info> road-mine tk

<Operation> ret = map\_is\_mining\_enable(1, 5, MINE\_AT, map1)

<Assert> ret == 1 -> TRUE

<Step> 3

<Info> roadside-mine ae

<Operation> ret = map\_is\_mining\_enable(1, 4, MINE\_AE, map1)

<Assert> ret == 0 -> TRUE

<Step> 4

<Info> roadside-mine tk

<Operation> ret = map\_is\_mining\_enable(1, 4, MINE\_AT, map1)

<Assert> ret == 0 -> TRUE

<Step> 5

<Info> road-mine rs

<Operation> ret = map\_is\_mining\_enable(1, 5, MINE\_RS, map1)

<Assert> ret == 0 -> TRUE

<Step> 6

<Info> roadside-mine rs

<Operation> ret = map\_is\_mining\_enable(3, 1, MINE\_RS, map1)

<Assert> ret == 1 -> TRUE

<Step> 7

<Info> water-mine rs

<Operation> ret = map\_is\_mining\_enable(4, 7, MINE\_RS, map1)

<Assert> ret == 0 -> TRUE

<Step> 8

<Info> house-mine rs

<Operation> ret = map\_is\_mining\_enable(12, 2, MINE\_RS, map1)

<Assert> ret == 0 -> TRUE

<Step> 9

<Info> forest-mine rs

<Operation> ret = map\_is\_mining\_enable(5, 7, MINE\_RS, map1)

<Assert> ret == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> MAP-01-009

<Purpose> validate map\_is\_road\_side.

<Precondition> -

<Step> 1

<Info> road side

<Operation> ret = map\_is\_road\_side(0, 4, map1)

<Assert> ret == 1 -> TRUE

<Step> 2

<Info> road side

<Operation> ret = map\_is\_road\_side(0, 6, map1)

<Assert> ret == 1 -> TRUE

<Step> 3

<Info> road side

<Operation> ret = map\_is\_road\_side(1, 1, map1)

<Assert> ret == 1 -> TRUE

<Step> 4

<Info> road side

<Operation> ret = map\_is\_road\_side(5, 1, map1)

<Assert> ret == 1 -> TRUE

<Step> 5

<Info> road side

<Operation> ret = map\_is\_road\_side(5, 5, map1)

<Assert> ret == 1 -> TRUE

<Step> 6

<Info> forest

<Operation> ret = map\_is\_road\_side(5, 7, map1)

<Assert> ret == 1 -> TRUE

<Step> 7

<Info> house

<Operation> ret = map\_is\_road\_side(13, 2, map1)

<Assert> ret == 1 -> TRUE

<Step> 8

<Info> entrance

<Operation> ret = map\_is\_road\_side(0, 5, map1)

<Assert> ret == 0 -> TRUE

<Step> 8

<Info> road

<Operation> ret = map\_is\_road\_side(1, 5, map1)

<Assert> ret == 0 -> TRUE

<Step> 9

<Info> not in range

<Operation> ret = map\_is\_road\_side(-1, 5, map1)

<Assert> ret == 0 -> TRUE

<Step> 10

<Info> not in range

<Operation> ret = map\_is\_road\_side(16, 9, map1)

<Assert> ret == 0 -> TRUE

<Result> OK

----------------------------------------

MAP-01-001 OK

MAP-01-002 OK

MAP-01-003 OK

MAP-01-004 OK

MAP-01-005 OK

MAP-01-006 OK

MAP-01-007 OK

MAP-01-008 OK

MAP-01-009 OK

## ViewStack

----------------------------------------

<Case> VSTACK-01-001

<Purpose> validate API regView.

<Precondition> view stack and views have be created.

<Step> 1

<Param> view1=00CDE768

<Operation> vStack->regView(vStack, VID\_1, (View\_t \*)view1)

[vStack/I] regView, vid=0

<Assert> vStack->mIndex[VID\_1] == (View\_t \*)view1 -> TRUE

<Step> 2

<Param> view2=00CDEA98

<Operation> vStack->regView(vStack, VID\_2, (View\_t \*)view2)

[vStack/I] regView, vid=1

<Assert> vStack->mIndex[VID\_2] == (View\_t \*)view2 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> VSTACK-01-002

<Purpose> validate API push1stView and getTopView.

<Precondition> views have be registered.

<Step> 1

<Param> view id=0

<Operation> vStack->push1stView(vStack, VID\_1)

[vStack/I] push1stView, vid=0

[View/I] onCreate, id=0

[View/I] onEnter, id=0

[View/I] onShow, id=0

<Assert> topV == (View\_t \*)view1 -> TRUE

<Assert> vStack->mSp == 0 -> TRUE

<Assert> onCreateFlag1 == 1 -> TRUE

<Assert> onEnterFlag1 == 1 -> TRUE

<Assert> onShowFlag1 == 1 -> TRUE

<Assert> onHideFlag1 == 0 -> TRUE

<Assert> onExitFlag1 == 0 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> VSTACK-01-003

<Purpose> validate API callView.

<Precondition> views have be registered.

<Step> 1

<Operation> topV = vStack->getTopView(vStack)

<Operation> msg->event = EV\_TEST\_CALL

<Operation> topV->sendEvent(topV, msg)

<Operation> \_self->super.setTransfer(&\_self->super, VID\_2, TRANS\_CALL, 1, 2,

3, 4)

<Operation> topV = vStack->transfer(vStack)

[vStack/I] callView, vid=1

[View/I] onHide, id=0

[View/I] onCreate, id=1

[View/I] onEnter, id=1

[View/I] onShow, id=1

<Assert> topV == (View\_t \*)view2 -> TRUE

<Assert> vStack->mSp == 1 -> TRUE

<Assert> onCreateFlag1 == 0 -> TRUE

<Assert> onEnterFlag1 == 0 -> TRUE

<Assert> onShowFlag1 == 0 -> TRUE

<Assert> onHideFlag1 == 1 -> TRUE

<Assert> onExitFlag1 == 0 -> TRUE

<Assert> onCreateFlag2 == 1 -> TRUE

<Assert> onEnterFlag2 == 1 -> TRUE

<Assert> onShowFlag2 == 1 -> TRUE

<Assert> onHideFlag2 == 0 -> TRUE

<Assert> onExitFlag2 == 0 -> TRUE

<Assert> para2\_p1 == 1 -> TRUE

<Assert> para2\_p2 == 2 -> TRUE

<Assert> para2\_p3 == 3 -> TRUE

<Assert> para2\_p4 == 4 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> VSTACK-01-004

<Purpose> validate API returnView.

<Precondition> views have be registered.

<Step> 1

<Operation> topV = vStack->getTopView(vStack)

<Operation> msg->event = EV\_TEST\_RETURN

<Operation> topV->sendEvent(topV, msg)

<Operation> \_self->super.setTransfer(&\_self->super, -1, TRANS\_RETURN, 1, 2, 3

, 4)

<Operation> topV = vStack->transfer(vStack)

[vStack/I] returnView, vid=1

[View/I] onHide, id=1

[View/I] onExit, id=1

[View/I] onShow, id=0

<Assert> topV == (View\_t \*)view1 -> TRUE

<Assert> vStack->mSp == 0 -> TRUE

<Assert> onCreateFlag1 == 0 -> TRUE

<Assert> onEnterFlag1 == 0 -> TRUE

<Assert> onShowFlag1 == 1 -> TRUE

<Assert> onHideFlag1 == 0 -> TRUE

<Assert> onExitFlag1 == 0 -> TRUE

<Assert> onCreateFlag2 == 0 -> TRUE

<Assert> onEnterFlag2 == 0 -> TRUE

<Assert> onShowFlag2 == 0 -> TRUE

<Assert> onHideFlag2 == 1 -> TRUE

<Assert> onExitFlag2 == 1 -> TRUE

<Assert> para1\_p1 == 5 -> TRUE

<Assert> para1\_p2 == 6 -> TRUE

<Assert> para1\_p3 == 7 -> TRUE

<Assert> para1\_p4 == 8 -> TRUE

<Result> OK

----------------------------------------

----------------------------------------

<Case> VSTACK-01-005

<Purpose> validate API changeView.

<Precondition> views have be registered.

<Step> 1

<Operation> topV = vStack->getTopView(vStack)

<Operation> msg->event = EV\_TEST\_CHANGE

<Operation> topV->sendEvent(topV, msg)

<Operation> \_self->super.setTransfer(&\_self->super, VID\_2, TRANS\_CHANGE, 1, 2

, 3, 4)

<Operation> topV = vStack->transfer(vStack)

[vStack/I] changeView, vid=1

[View/I] onHide, id=0

[View/I] onExit, id=0

[View/I] onCreate, id=1

[View/I] onEnter, id=1

[View/I] onShow, id=1

<Assert> topV == (View\_t \*)view2 -> TRUE

<Assert> vStack->mSp == 0 -> TRUE

<Assert> onCreateFlag1 == 0 -> TRUE

<Assert> onEnterFlag1 == 0 -> TRUE

<Assert> onShowFlag1 == 0 -> TRUE

<Assert> onHideFlag1 == 1 -> TRUE

<Assert> onExitFlag1 == 1 -> TRUE

<Assert> onCreateFlag2 == 1 -> TRUE

<Assert> onEnterFlag2 == 1 -> TRUE

<Assert> onShowFlag2 == 1 -> TRUE

<Assert> onHideFlag2 == 0 -> TRUE

<Assert> onExitFlag2 == 0 -> TRUE

<Assert> para2\_p1 == 1 -> TRUE

<Assert> para2\_p2 == 2 -> TRUE

<Assert> para2\_p3 == 3 -> TRUE

<Assert> para2\_p4 == 4 -> TRUE

<Result> OK

----------------------------------------

VSTACK-01-001 OK

VSTACK-01-002 OK

VSTACK-01-003 OK

VSTACK-01-004 OK

VSTACK-01-005 OK