public:

//static void RegisterSetUpTestCase( testing::Test::SetUpTestCaseFunc set\_up\_tc ) {}

//static void RegisterSetUpTestCase( testing::AssertionResult\* set\_up\_tc ) {}

//static void RegisterTearDownTestCase( testing::Test::TearDownTestCaseFunc tear\_down\_tc ) {}

TestCaseFuncList

void RunSetUpTestCase() { (\*set\_up\_tc\_)(); }

void RunTearDownTestCase() { (\*tear\_down\_tc\_)(); }

CPPUNIT\_TEST\_SUITE和CPPUNIT\_TEST\_SUITE\_END所做的事情包括：

1. 声明并实现了一个叫做getTestNamer的静态方法

static const CPPUNIT\_NS::TestNamer &getTestNamer\_\_()

{

static CPPUNIT\_TESTNAMER\_DECL( testNamer, ATestFixtureType ); \

return testNamer;

}

CPPUNIT\_NS::TestNamer类提供了两个方法，在构造Test Tree的时候，各个树节点的名称（字符串）就是根据Namer的方法返回的

virtual std::*string* getFixtureName() const; // 根节点

virtual std::*string* getTestNameFor( const std::*string* &testMethodName ) const; // 叶子结点

1. 还声明并实现了一个叫做addTestsToSuite的静态方法，这个方法的作用就是在定义当前Fixture结点包含哪些Test方法，根节点包含哪些叶子结点
2. CPPUNIT\_TEST用来完成把添加叶子结点的操作
3. 每一个叶子结点对应了一个叫做TestCaller的类型

CPPUNIT\_TEST\_SUITE\_REGISTRATION所做的事情：

1.定义了一个AutoRegisterSuite类型的全局变量

AutoRegisterSuite的作用：

TestFactoryRegistry \*m\_registry;

TestSuiteFactory<TestCaseType> m\_factory;

先说TestFactoryRegistry，它内部维护了一个TestFactory\*类型的std::set

而所有TestFactoryRegistry又是由一个叫做TestFactoryRegistryList的单例对象持有的。

TestFactoryRegistryList通过一个name to TestFactory\*的std::map

CPPUNIT\_TEST\_SUITE\_REGISTRATION会产生一个测试树的根节点，这个根节点对应了一个AutoRegisterSuite类型的对象，AutoRegisterSuite知道根节点对应的具体TestFixture类型，但它并不负责TestFixture实例的创建，而是通过TestSuiteFactory<TestCaseType> m\_factory;来访问具体TestFixture::suite()静态方法。

具体TestFixture::Suite()返回一个TestSuite\*