Valgrind官网请参考如下网址：

<http://valgrind.org/>

1. **Valgrind简介**

Valgrind定义为framework，用来构建动态分析工具，这些Valgrind工具可以用来检测内存问题、进行性能分析……Valgrind是开源、自由的，基于GNU GPL v2协议，最新的release版本为2014年11月25日的Valgrind-3.10.1，发布了一些可以在多平台上使用的工具，如Memcheck工具，用来检测内存管理问题。

1. **为什么要使用Valgrind**

在软件开发中，我们经常要花费许多时间去调查bug，想尽一切办法去提成程序性能，有时候煞费苦心折腾了半天却没什么效果，这时我们就可以借助Valgrind，通过Valgrind工具来查找bug、进行性能分析以查找技术瓶颈，最终提高程序的健壮性。

1. **Valgrind工具简介**

Valgrind发布了多个工具，下面作个简单的介绍。

**Memcheck——**

Memcheck主要针对于C/C++程序，用以检测内存管理问题。当读写内存或者是调用malloc/new/free/delete时，如果发生非法操作，Memcheck会检测到并立即报告出来，内容包括代码行号和一些堆栈信息。Memcheck运行程序时要比程序正常运行时慢许多，能够检测到如下几个问题：

**访问非法内存**，如未分配的内存、已经释放的内存，堆上越界访问，访问挥发性的栈等；

**使用未初始化的变量**，未初始化变量的值是不确定的，用起来比较危险；

**内存泄漏**，调用了malloc/new，却没有free/delete；

**释放内存错误**，如释放已经释放的内存即常见的double free，malloc/free、new/delete不匹配等；

**内存块重叠使用**，如memcpy()一类的函数，源地址和目标地址是不能重叠的。

**Cachegrind——**

Cachegrind是一个缓存分析器，用来模拟CPU缓存，这样我们就可以查明源代码中缓存缺失的信息，这些信息包括缓存缺失的数量、内存引用及代码指令的执行，小到每一个函数，大到整个程序，适用于任何语言，只不过使用Cachegrind运行程序时要比正常情况下慢20到100倍。

**Callgrind——**

Callgrind是Cachegrind的一个扩展，既然是扩展，就提供了Cachegrind的所有功能，此外还包括一些图表信息，而可视化工具KCachegrind则用来查看这些数据。

**Massif——**

Massif是一个堆内存分析工具，提供了图表以实时显示堆内存的使用情况，大约比程序正常运行时慢20倍。

**Helgrind——**

Helgrind是一个线程调试器，目前还处于试验阶段，可以在多线程的程序中找出数据竞争信息，当内存被多个线程访问却没有使用同步锁时，Helgrind能够找到这些内存地址，当我们使用了pthreads线程出现问题时，可以使用这个工具排查问题。

**DRD——**

DRD工具用于在多线程pthreads的C/C++程序中检查错误，如上锁的时机有误，对于大多数程序来说DRD不需要太多的内存。

**Lackey、Nulgrind——**

Lackey和Nulgrind不常用，测试用。另外，还有一些其它的工具，可参考如下网址：

<http://valgrind.org/info/tools.html>

1. **简单使用Valgrind**

1）Valgrind安装

Valgrind源码包可在官网下载：

<http://valgrind.org/>

下载成功后，解压源码包，cd进源码包执行如下命令：

$./autogen.sh

$./configure

$make

$sudo make install

安装成功。。。

2）编写测试用例

// valgrindex.cpp

#include <iostream>

void foo()

{

int \*p = new int(10); // 没有delete 内存泄漏

int \*pArray = new int[10]; // 没有delete[] 内存泄漏

pArray[10] = 10; // 数组越界非法操作

}

int main(int argc, char \*\*agrv)

{

foo();

return 0;

}

3）使用Valgrind

$g++ -g –o test valgrindex.cpp

$valgrind ./test

4）结果

==8884== Memcheck, a memory error detector

==8884== Copyright (C) 2002-2013, and GNU GPL'd, by Julian Seward et al.

==8884== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info

==8884== Command: ./test

==8884==

==8884== Invalid write of size 4

==8884== at 0x40078F: foo() (valgrindex.cpp:8)

==8884== by 0x4007AA: main (valgrindex.cpp:13)

==8884== Address 0x5a1b0b8 is 0 bytes after a block of size 40 alloc'd

==8884== at 0x4C2B76A: operator new[](unsigned long) (vg\_replace\_malloc.c:389)

==8884== by 0x400782: foo() (valgrindex.cpp:7)

==8884== by 0x4007AA: main (valgrindex.cpp:13)

==8884==

==8884==

==8884== HEAP SUMMARY:

==8884== in use at exit: 44 bytes in 2 blocks

==8884== total heap usage: 2 allocs, 0 frees, 44 bytes allocated

==8884==

==8884== LEAK SUMMARY:

==8884== definitely lost: 44 bytes in 2 blocks

==8884== indirectly lost: 0 bytes in 0 blocks

==8884== possibly lost: 0 bytes in 0 blocks

==8884== still reachable: 0 bytes in 0 blocks

==8884== suppressed: 0 bytes in 0 blocks

==8884== Rerun with --leak-check=full to see details of leaked memory

==8884==

==8884== For counts of detected and suppressed errors, rerun with: -v

==8884== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)

从上面的结果可以看出，有内存问题的详细堆栈信息，这里检测到了数组越界非法操作的具体位置，还有内存泄漏信息，从我们的测试用例也可以看出泄漏了44字节，如果使用valgrind的参数--leak-check=full，可以看出内存泄漏的详细堆栈信息。