**测试报告**

|  |  |
| --- | --- |
| **小 组 成 员** | |
| **202021090xx3** | **陈x** |
| **202022090xx3** | **肖xx** |

目录

[1 测试概述 1](#_Toc58094615)

[1.1 测试目的 1](#_Toc58094616)

[1.2 测试范围 1](#_Toc58094617)

[1.3 测试环境 1](#_Toc58094618)

[2 测试策略 2](#_Toc58094619)

[2.1 测试方法 2](#_Toc58094620)

[2.2 测试流程 2](#_Toc58094621)

[3 测试用例设计 3](#_Toc58094622)

[4 测试执行 4](#_Toc58094623)

[5 测试结论及分析 8](#_Toc58094624)

[5.1 总体分析 8](#_Toc58094625)

# 测试概述

## 测试目的

系统测试旨在通过测试找出系统在设计和实现时可能潜在的缺陷与错误，对测试结果进行分析，验证系统是否符合需求分析、系统设计要求；具体为测试各种结构实现的服务器的性能，结构包括迭代、多进程、多线程、I/O复用、线程池等，通过分析各种结构服务器的性能表现，找出各种服务器的特点。

## 测试范围

测试范围为：用迭代、多进程、多线程、I/O复用、线程池等多种技术设计的Web服务器。

## 测试环境

* 系统：[Ubuntu16.04.2 + Windows 10(VM 虚拟机](https://blog.csdn.net/q1007729991/article/details/69224317))
* 处理器：Intel 酷睿i5 9400
* 内存：2.00G

# 测试策略

## 测试方法

本次测试主要采用黑盒测试的方法，对其功能和性能进行测试。

黑盒测试又称为功能测试，主要检测软件的每一个功能是否能够正常使用。在测试过程中，将程序看成不能打开的黑盒子，不考虑程序内部结构和特性的基础上通过程序接口进行测试，检查程序功能是否按照设计需求以及说明书的规定能够正常打开使用。

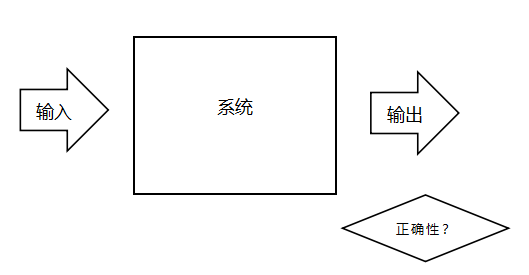


图2.1 黑盒测试示意图

## 测试流程

* 启动服务器端
* 启动客户端，输入相应参数
* 连接服务器，访问，记录数据

首先编写一个简单的测试工具，该工具能够通过读取用户输入的参数，创建若干个进程同时对服务器进行访问，每个进程都可以与服务器建立多次连接。该测试工具的使用方法为：工具名 服务器ip地址 服务器端口号 要创建的子进程数 每个子进程建立的连接数。

编写完该测试工具后，首先使用该工具创建同样的子进程数和连接数访问不同网络模型的Web服务器，记录测试时间和平均响应时间进行比较。

# 测试用例设计

根据各个体系结构的web服务器，分别设计测试用例。设计的测试用例如表3-1所示。

表3-1 测试用例表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试用例ID** | **场景** | **测试步骤** | **预期结果** | **备注** |
| TC1 | 模拟10个用户访问web服务器，每个用户建立5个连接 | 打开终端  运行迭代式服务器  运行客户端测试工具 | 正确访问  用户平均操作响应时间不超过1秒 | 无 |
| TC2 | 模拟10个用户访问web服务器，每个用户建立5个连接 | 打开终端  运行I/O复用式服务器  运行客户端测试工具 | 正确访问  用户平均操作响应时间不超过1秒 | 无 |
| TC3 | 模拟10个用户访问web服务器，每个用户建立5个连接 | 打开终端  运行多进程式服务器  运行客户端测试工具 | 正确访问  用户平均操作响应时间不超过1秒 | 无 |
| TC4 | 模拟10个用户访问web服务器，每个用户建立5个连接 | 打开终端  运行多线程式服务器  运行客户端测试工具 | 正确访问  用户平均操作响应时间不超过1秒 | 无 |
| TC5 | 模拟10个用户访问web服务器，每个用户建立5个连接 | 打开终端  运行线程池式服务器  运行客户端测试工具 | 正确访问  用户平均操作响应时间不超过1秒 | 无 |

# 测试执行

按照设计的测试用例，依次执行，得到的测试结果如表3-2所示。

表3-2 测试结果表

|  |  |
| --- | --- |
| **测试用例编号** | **测试结果** |
| TC1 | 功能正确执行 |
| TC2 | 功能正确执行 |
| TC3 | 功能正确执行 |
| TC4 | 功能正确执行 |
| TC5 | 功能正确执行 |

在测试过程中，记录的测试数据如表3-3所示。

表3-3 测试数据表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **测试用例编号** | **用户数** | **连接数** | **测试时间** | **平均响应时间** |
| TC1 | 10 | 5 | 17.94 | 0.35 |
| TC2 | 10 | 5 | 8.73 | 0.17 |
| TC3 | 10 | 5 | 2.23 | 0.04 |
| TC4 | 10 | 5 | 5.42 | 0.1 |
| TC5 | 10 | 5 | 8.62 | 0.17 |

测试过程截图如下：

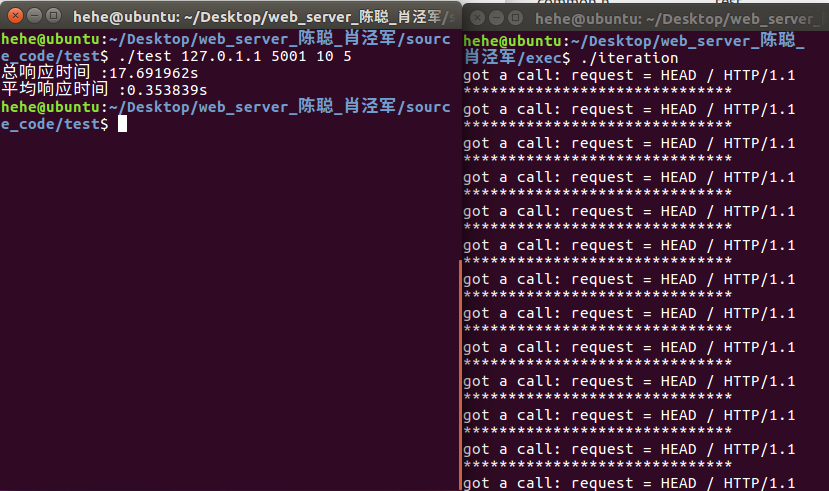


图4-1 迭代式测试截图

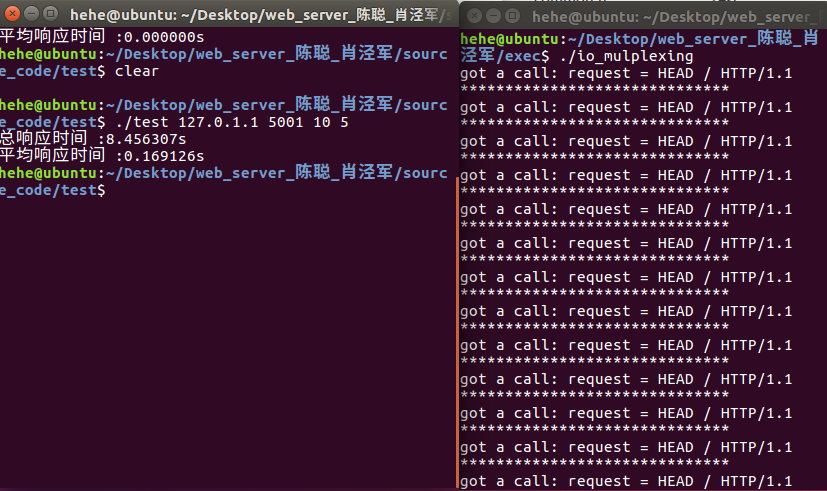


图4-2 I/O复用式测试截图

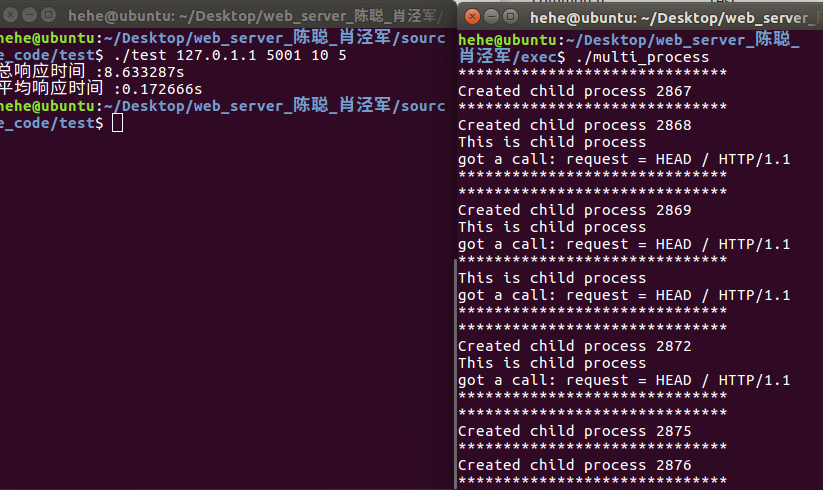


图4-3 多进程式测试截图

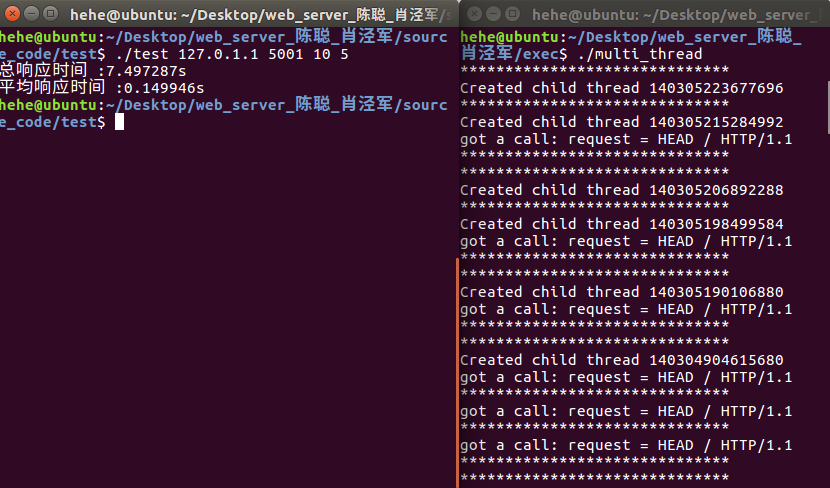


图4-4 多线程式测试截图

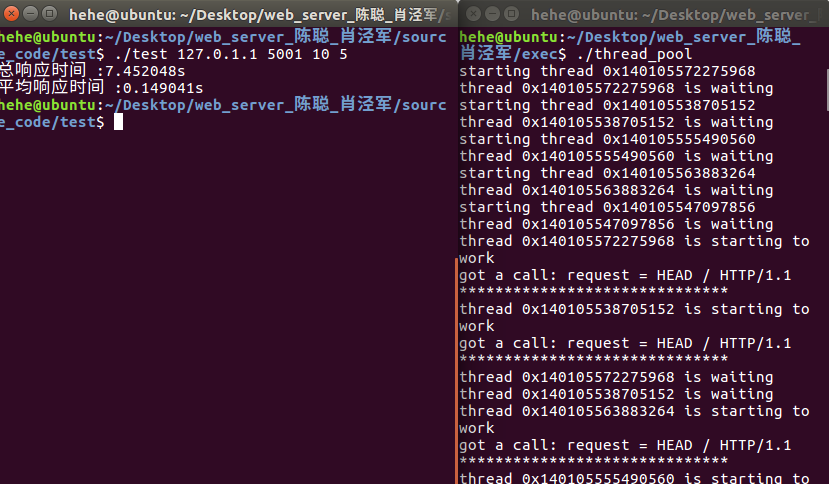


图4-5 线程池式测试截图

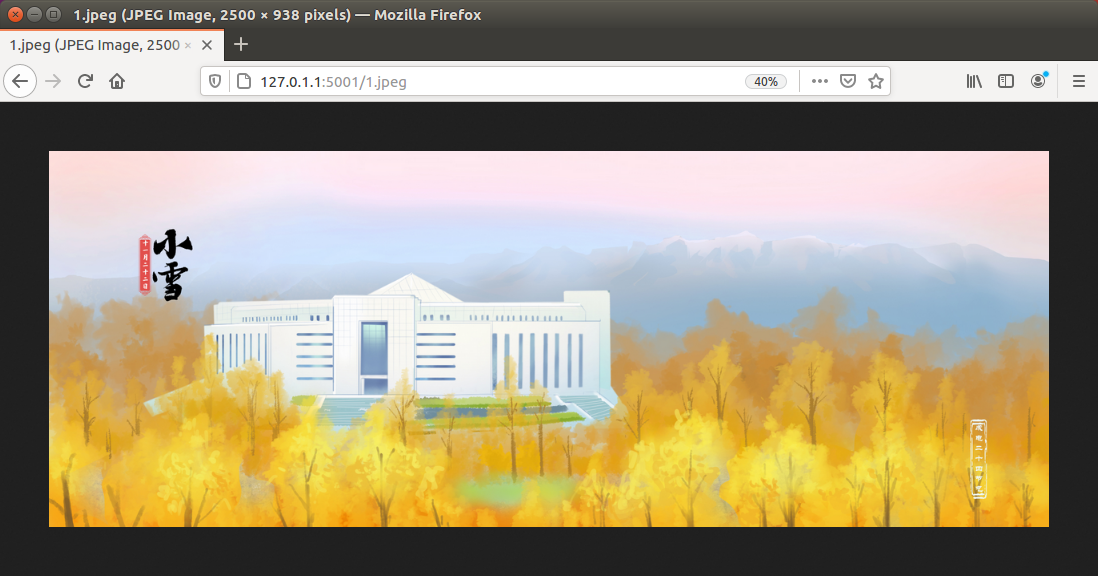


图4-6 服务器资源访问截图

# 测试结论及分析

## 总体分析

|  |  |  |
| --- | --- | --- |
| **体系结构** | **优点** | **缺点** |
| 多进程 | 1）每个进程互相独立，不影响主程序的稳定性，子进程崩溃没关系； 2）通过增加CPU，就可以容易扩充性能； 3）可以尽量减少线程加锁/解锁的影响，极大提高性能，就算是线程运行的模块算法效率低也没关系； 4）每个子进程都有2GB地址空间和相关资源，总体能够达到的性能上限非常大。 | 1）逻辑控制复杂，需要和主程序交互； 2）需要跨进程边界，如果有大数据量传送，就不太好，适合小数据量传送、密集运算 ； 3）多进程调度开销比较大。 |
| 多线程 | 1）无需跨进程边界； 2）程序逻辑和控制方式简单； 3）所有线程可以直接共享内存和变量等； 4）线程方式消耗的总资源比进程方式好； | 1）每个线程与主程序共用地址空间，受限于2GB地址空间； 2）线程之间的同步和加锁控制比较麻烦； 3）一个线程的崩溃可能影响到整个程序的稳定性； 4）到达一定的线程数程度后，即使再增加CPU也无法提高性能， 5）线程能够提高的总性能有限，而且线程多了之后，线程本身的调度也是一个麻烦事儿，需要消耗较多的CPU 。 |
| I/O复用 | 1）相比于多线程和多进程，I/O多路复用是在单一进程的上下文中的，当有多个并发连接请求时，多线程或者多进程模型需要为每个连接创建一个线程或者进程，而这些进程或者线程中大部分是被阻塞起来的。由于CPU的核数一般都不大，比如4个核要跑1000个线程，那么每个线程的时间槽非常短，而线程切换非常频繁。这样是有问题的。而使用I/O多路复用时，处理多个连接只需要1个线程监控就绪状态，对就绪的每个连接开一个线程处理（由线程池支持）就可以了，这样需要的线程数大大减少，减少了内存开销和上下文切换的CPU开销。 2）整个过程只在调用select、poll、epoll这些调用的时候才会阻塞，收发客户消息是不会阻塞的，整个进程或者线程就被充分利用起来，这就是事件驱动。 | 单线程模型不能有阻塞，一旦发生任何阻塞（包括计算机计算延迟）都会使得这个模型不如多线程。另外，单线程模型不能很好的利用多核cpu。 |
| 迭代 | 1. 编写简单，对程序员的要求不高 2. 对于单次请求，响应数据快 | 1. 一次只能处理一个客户端的请求，容易造成阻塞，无法应付并发情况 |
| 线程池 | 减少每次获取资源的消耗，提高对资源的利用率，从而极大地提升了速度。 | 如果线程缺乏统一管理，可能无限制新建线程，相互之间竞争，及可能占用过多系统资源导致死机或oom（Out Of Memory） |