

《分布式系统》

实验指导书

余腊生

信息科学与工程学院

2019 年 03 月

目 录

(选做 3-4 题)

实验一	数据包 socket 应用	1
实验二	流式 socket 应用	4
实验三	客户/服务器应用开发	6
实验四	RMI API	8
实验五	CORBA 系统编程	14
实验六	支持选举的 Java IDL 应用	19
实验七	Internet 应用	20
实验八	实现一个基本的 Web 服务器程序	22
实验八	实现一个简单的 Web 应用程序(2)	26
实验九	Google AppEngine 程序设计	27
实验十	安装 Xen 熟悉常用命令	30
实验十	基于 Xen 虚拟机实现 tomcat cluster	33
实验十一	GFS 的安装与配置	34
实验十二	XAMPP 的安装与使用	37
实验十三	虚拟机的使用与 Linux 系统的安装	40
实验十四	Hadoop 的安装与部署	42
实验十五	Windows Azure 云平台搭建和部署云平台服务	44
开放式课题	(相对需求不是很固定, 可以自由发挥, 至少 1 题)	45

实验一 数据包 socket 应用

实验目的

1. 理解数据包 socket 的应用
2. 实现数据包 socket 通信
3. 了解 Java 并行编程的基本方法

预习与实验要求

1. 预习实验指导书及教材的有关内容，了解数据包 socket 的通信原理；
2. 熟悉一种 java IDE 和程序开发过程；
3. 了解下列 Java API: Thread、Runnable；
4. 尽可能独立思考并完成实验。

实验环境

- a) 独立计算机或计算机网络；
- b) Windows 操作系统。
- c) Jdk 工具包
- d) JCreator or others

实验原理

1. 分布式计算的核心是进程通信。操作系统、网卡驱动程序等应用从不同抽象层面提供了对进程通信的支持，例如

Winsock、java.net.*。Socket API 是一种作为 IPC 提供对系统低层抽象的机制。尽管应用人员很少需要在该层编写代码，但理解 socket API 非常重要，因为：1，高层设施是构建于 socket API 之上的，即他们是利用 socket API 提供的操作来实现；2，对于以响应时间要求较高或运行于有限资源平台上的应用来说，socket API 可能是最适合的。

在 Internet 网络协议体系结构中，传输层上有 UDP 和 TCP 两种主要协议，UDP 允许在传送层使用无连接通信传送，被传输报文称为数据包。（是否存在面向连接的数据包 socket？）因此数据包 socket 是基于 UDP 的不可靠 IPC。Java 为数据包 socket API 提供两个类：

- （1）针对 socket 的 datagramSocket 类
- （2）针对数据包交换的 datagramPacket 类

希望使用该 API 发送和接收数据的进程须实例化一个 datagramSocket 对象，每个 socket 被绑定到该进程所在及其的某个 UDP 端口上。为了向其他进程发送数据包，进程必须创建

一个代表数据包本身的对象。该对象通过实例化一个 `datagram socket` 对象创建。在接收者进程中，`datagramPacket` 对象也必须被实例化并绑定到一个本地端口上，该端口必须与发送者数据包的定义一致。接收进程创建一个指向字节数组的 `DatagramPacket`，并调用 `datagramSocket` 对象的 `receive` 方法，将 `DatagramPacket` 对象指针作为参数定义。

2. 并行编程（以 Java 为例¹）

一个线程是比进程更小的执行粒度。Java 虚拟机允许应用程序有多个执行线程同时运行。有两种方法来创建一个新线程的执行。一个是声明一个类是一个线程的子类。这个子类应重写 `Thread` 类的 `run` 方法。一个子类的实例可以被分配和启动。另一种方法创建一个线程，并同时声明一个类实现了 `Runnable` 接口（这个类要实现 `run` 方法）。一个类的实例可以被分配并作为参数传递给创建的线程，并启动线程。例如：

◆ 创建一个类是 `Thread` 的子类：

```
class SomeThread extends Thread {  
    SomeThread() {  
    }  
  
    public void run() {  
        ...  
    }  
}
```

```
SomeThread p = new SomeThread();  
p.start();
```

◆ 创建一个实现 `Runnable` 接口的类并传递给线程：

```
class SomeRun implements Runnable  
{ SomeRun() {  
    }  
  
    public void run() {  
        ...  
    }  
}
```

```
SomeRun p = new SomeRun(143);  
new Thread(p).start();
```

当一个实现 `Runnable` 接口的类被执行时，可以没有子类。实例化一个 `Thread` 实例，并通过自身作为目标线程。在大多数情况下，如果你只打算重写的 `run()` 方法，并没有其它

¹ java.sun.com

的线程方法，应使用 `Runnable` 接口。因为类不应该被继承，除非程序员有意修改或增强类的基本行为。

实验内容

1. 构建客户端程序

- (1) 构建 `DatagramSocket` 对象实例
- (2) 构建 `DatagramPacket` 对象实例，并包含接收者主机地址、接收端口号等信息
- (3) 调用 `DatagramSocket` 对象实例的 `send` 方法，将 `DatagramPacket` 对象实例作为参数发送。

2. 构建服务器端程序

- (1) 构建 `DatagramSocket` 对象实例，指定接收的端口号。
- (2) 构建 `DatagramPacket` 对象实例，用于重组接收到的消息。
- (3) 调用 `DatagramSocket` 对象实例的 `receive` 方法，进行消息接收，并将 `DatagramPacket` 对象实例作为参数。

实验报告

1. 客户端和服务端程序的伪代码；
2. 试验过程中的问题和解决途径；
3. 写实验报告。

思考题

1. 如何避免数据包丢失而造成的无限等待问题？
2. 如何实现全双工的数据包通信？

实验二 流式 socket 应用

实验目的

1. 理解流式 socket 的原理
2. 实现流式 socket 通信

预习与实验要求

3. 预习实验指导书及教材的有关内容，了解流式 socket 的通信原理；
4. 熟悉 java 环境和程序开发过程；
5. 尽可能独立思考并完成实验。

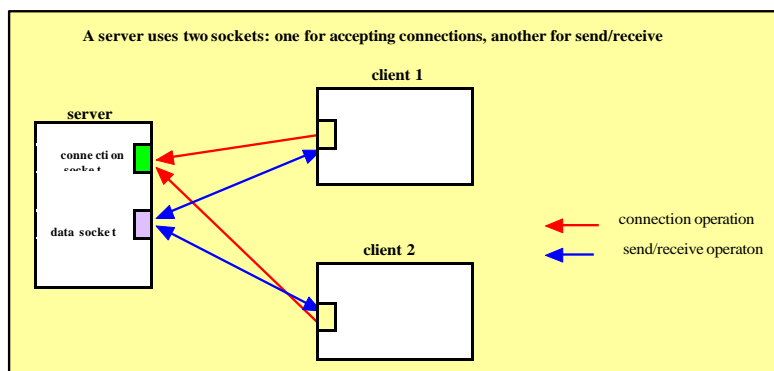
实验环境

- a) 独立计算机；
- b) Windows 操作系统；
- c) Jdk 工具包

实验原理

Socket API 是一种作为 IPC 提供低层抽象的机制。尽管应用人员很少需要在该层编写代码，但理解 socket API 非常重要，因为：1，高层设施是构建于 socket API 之上的，即他们利用 socket API 提供的操作来实现；2，对于以响应时间要求较高或运行于有限资源平台上的应用来说，socket API 可能是最适合的。

在 Internet 网络协议体系结构中，传输层上有 UDP 和 TCP 两种主要协议，UDP 允许使用无连接通信传送，被传输报文称为数据包。而 TCP 则允许面向连接的可靠通信，这种 IPC 称为流式 socket。Java 为流式 socket API 提供两类 socket（1）式用于连接，连接 socket（2）式用于数据交换的数据 socket。



实验内容

1. 构建客户端程序和服务器端程序都需要的 `MystreamSocket` 类，定义继承自 `java Socket` 的 `sendMessage` 和 `receiveMessage` 方法
2. 构建客户端程序
 - (1) 创建一个 `MyStreamsocket` 的实例对象，并将其指定接收服务器和端口号
 - (2) 调用该 `socket` 的 `receiveMessage` 方法读取从服务器端获得的消息
3. 构建服务器端程序
 - (1) 构建连接 `socket` 实例，并与指定的端口号绑定，该连接 `socket` 随时侦听客户端的连接请求
 - (2) 创建一个 `MyStreamsocket` 的实例对象
 - (3) 调用 `MyStreamsocket` 的实例对象的 `sendMessage` 方法，进行消息反馈。

实验报告

1. 应用程序的结构图，说明程序之间的关系；
2. 程序的伪代码。

思考题

1. 如何实现全双工的流式 `socket` 通信？
2. 如何实现安全 `socket API`？
3. 如何实现 1 对多的并发？

实验三 客户/服务器应用开发

实验目的

1. 验证 daytime 和 echo 程序，
2. 实现包 socket 支撑的 C/S 模式 IPC 机制
3. 实现流式 socket 支撑的 C/S 模式 IPC 机制

预习与实验要求

1. 预习实验指导书及教材的有关内容，了解 daytime 和 echo 要提供的具体服务内容；
2. 复习包 socket 和流式 socket 的实现原理；
3. 实验前认真听讲，服从安排。尽可能独立思考并完成实验。

实验环境

- a) 独立计算机；
- b) Windows 操作系统。
- c) Jdk 工具包

实验原理

C/S 模式是主要的分布式应用范型，其设计的目的是提供网络服务。网络服务指如 daytime、telnet、ftp 和 WWW 之类的允许网络用户共享资源的服务。要构建 C/S 范型的应用就必须解决以下一些关键问题：

- (1) 如何通过会话实现多个用户的并发问题
- (2) 如何定义客户和服务会话期间必须遵守的协议
- (3) 服务定位问题
- (4) 进程间通信和事件同步问题：语法、语义和响应
- (5) 数据表示问题

在解决了这些问题的基础上，C/S 范型必须遵从 3 层结构的软件体系结构：

- (1) 表示层，提供与客户端进行交互的界面
- (2) 应用逻辑层，定义服务器和客户端要处理的主要事务的业务逻辑
- (3) 服务层，定义应用逻辑层所需要的底层支持技术，例如定义其 IPC 机制里的 receive 方法和 send 方法等。

实验内容

1. 构建用数据包 socket 实现的 daytime 客户端程序
 - (1) 构建表示层程序 DaytimeClient1.java
 - (2) 构建应用逻辑层程序 DaytimeHelper1.java
 - (3) 构建服务层程序 MyClientDatagramSocket.java
2. 构建用数据包 socket 实现的 daytime 服务器端程序
 - (1) 构建表示层和应用逻辑层程序 DaytimeServer1.java
 - (2) 构建服务层程序 MyServerDatagramSocket.java
 - (3) 构建服务层程序 MyServerDatagramSocket.java 所需要的下层程序 DatagramMessage.java (它封装了客户端的消息和地址)
3. 构建用流式 socket 实现的 daytime 应用程序包
4. 构建用数据包 socket 实现的 echo 应用程序包
5. 构建用流式 socket 实现的 echo 应用程序包

实验报告

1. 用数据包 socket 实现的 daytime 应用程序包的构架, 列明各程序之间的关系;
2. 用流式 socket 实现的 daytime 应用程序包的构架, 列明各程序之间的关系;
3. 用数据包 socket 实现的 echo 应用程序包的构架, 列明各程序之间的关系;
4. 用流式 socket 实现的 echo 应用程序包的构架, 列明各程序之间的关系。

思考题

1. 如何实现有状态服务器的状态信息的维护?

实验四 RMI API

实验目的

1. 了解 Java RMI 体系结构,
2. 学会用 RMI API 开发 C/S 模式的应用程序

预习与实验要求

1. 预习实验指导书及教材的有关内容, 了解 RMI 技术原理;
2. 尽可能独立思考并完成实验。

实验环境

- a) 独立计算机;
- b) Windows 操作系统。
- c) Jdk 工具包

实验原理

RMI 技术是分布式对象范型的一个实例, 是 RPC 技术的扩展。在 Java RMI 体系结构中, 客户及服务器都提供三层抽象。

1. 客户端体系结构

- (1) stub 层: 负责解释客户程序发出的远程方法调用; 然后将其转发到下一层
- (2) 远程引用层: 解释和管理客户发出的到远程服务对象的引用, 并向下一层即传输层发起 IPC 操作, 从而将方法调用传送给远程主机
- (3) 传输层: 基于 TCP 协议

2. 服务器端体系结构

- (1) skeleton 层: 负责与客户端 stub 进行交互
- (2) 远程引用层: 管理源于客户端的远程引用, 并将其转换成能被 skeleton 层理解的本地引用
- (3) 传输层: 与客户端体系结构一样, 面向传输层。

实验内容

1. 开发服务器端软件

- (1) 为该应用的所有待生成文件创建一个目录
- (2) 在 someInterface.java 中定义远程服务器接口。编译并修改程序, 直到不再有任何

何语法错误

- (3) `SomeImpl.java` 中实现接口，编译并修改程序，直到不再有任何语法错误
- (4) 使用 RMI 编译器 `rmic` 处理实现类，生成远程对象的 `stub` 文件和 `skeleton` 文件

`Rmic SomeImpl`

- (5) 创建对象服务器程序 `SomeServer.java`，编译并修改程序，直到不再有任何语法错误
- (6) 激活对象服务器

`java SomeServer`

2. 开发客户端软件

- (1) 为该应用的所有待生成文件创建一个目录
- (2) 获取远程接口类文件的一个拷贝，使用 `javac` 编译，生成接口类文件
- (3) 获取接口实现 `stub` 文件 `SomeImpl_stub.class` 的一个拷贝
- (4) 开发客户程序 `SomeClient.java`，编译程序，生成客户类
- (5) 激活客户

`Java SomeClient`

实验报告

1. RMI 应用文件列表及放置情况
2. 每个文件主要实现的功能和文件相互间的关系
3. 每个文件的伪代码
4. 试验运行情况和结果，在试验中遇到的问题和分析

思考题

1. 如何使用 RMI 传递参数？
2. 如何使得被动提供远程服务的服务器能够主动发起数据请求？
3. 如何避免在多个客户端同时发起远程服务调用时产生不一致的情况？

示例：

一、首先编写一个接口中的参数类型，此类型也是一个接口

Java 代码

```
1. package remote.test;
2.
3. public interface HelloTask {
4.     public String execute();
5. }
```

二、实现这个接口，记住，实现的时候一定要实现

Serializable 接口

Java 代码

```
1. package remote.test;
2.
3. import java.io.Serializable;
4.
5. public class HelloTaskImpl implements HelloTask,Serializable{
6.     private static final long serialVersionUID = 1L;
7.
8.     public String execute() {
9.         return "Hello World";
10.    }
11. }
```

三、编写 Remote 接口

Java 代码

```
1. package remote.test;
2.
3. import java.rmi.RemoteException;
4.
5. public interface IHello extends java.rmi.Remote{
6.     String say(HelloTask task) throws RemoteException;
7. }
```

四、编写 Server 类

Java 代码

```
1. package remote.test.server;
2.
3. import java.rmi.RemoteException;
4. import java.rmi.registry.LocateRegistry;
5. import java.rmi.registry.Registry;
6. import remote.test.IHello;
7. import remote.test.HelloTask;
8.
9. public class HelloServer extends java.rmi.server.UnicastRemoteObject implements
10.     IHello{
11.     private static final long serialVersionUID = 2279096828129284306L;
```

```

12.
13.     public HelloServer() throws RemoteException {
14.         super();
15.     }
16.
17.     public String say>HelloTask task) throws RemoteException {
18.         String result = task.execute();
19.         System.out.println("execute say,task.execute : " + result);
20.         return result;
21.     }
22.
23.     public static void main(String[] args) {
24.         try {
25.             HelloServer h = new HelloServer();
26.             Registry registry = LocateRegistry.createRegistry(2500);
27.             registry.bind("hello", h);
28.             System.out.println("Start...");
29.         } catch (Exception e) {
30.             e.printStackTrace();
31.         }
32.     }
33. }

```

您看到在 `HelloServer` 将会实现 `IHello` 中的接口，并执行 `HelloTask` 的接口，`HelloTask` 是从 客户端那边传送过来，所以执行任务的逻辑将在 `say` 方法中由服务端来实现，客户端是不知道 服务端如何执行这个 `HelloTask`，他只需要知道执行的结果。 对于服务端代码的编写，这里需要注意的是 `Registry`，`Registry` 是由 `Java Security` 管理，如果 没有权限 `getRegistry` 将会报错，如果使用如下方法获得

Java 代码

```

1. Registry registry = LocateRegistry.getRegistry(2500);

```

就需要注意，在 `Jar` 中需要有个 `*.policy` 策略文件，如果没有设置这些将会抛出 `"Access Denied "` 异常。此方式的编写这里不详细讲解，请搜索网络需找资料。

接下来 `registry` 只需要绑定一个名字空间和对象即可开启 `2500` 端口开始监听。

五、生成Stub 对象

在 `java` 环境中生成这个对象需要使用 `rmic` 命令以及 `rmiregistry`，这两个命令的用法其实也 比较简单，但是无论如何使用起来还是会觉得有点麻烦，在 `Spring` 中也有 `rmi` 的功能，其实他 也就是包装了一下 `rmi` 的类，但是他最大的优点在于它可以自动生成这些托管类，那么使用 `RMIC - Eclipse - Plugin` 也可以帮助我们自动生成。

装好 `RMIC - Eclipse` 之后，对着项目的根目录点击右键，您将会看到 **"Rmic Marcar/Desmarcar para Auto-Generar Stubs"** 的选项，点击它，这时你将会在 `Bin` 文件夹下看到 `Hello_Stub` 类，如果有接口继承了 `Remote` 类，那么他将会自动搜索并生成。

六、编写Client 类

Java 代码

```
1. package remote.test.client;
2.
3. import java.rmi.Naming;
4. import remote.test.HelloTask;
5. import remote.test.HelloTaskImpl;
6. import remote.test.IHello;
7.
8. public class HelloClient {
9.     public static void main(String[] args) {
10.         try {
11.             IHello hi = (IHello) Naming.lookup("rmi://127.0.0.1:2500/hello");
12.
13.             HelloTask task = new HelloTaskImpl();
14.             for (int i = 0; i < 10; i++) {
15.                 System.out.println(hi.say(task));
16.             }
17.         } catch (Exception e) {
18.             e.printStackTrace();
19.         }
20.     }
21. }
```

在 Client 类中，首先需要使用 JNDI 的 Naming 来需找服务，并从服务中返回对象。URL 的格式可以参考 JDK 中 RMI URL。

实验五 CORBA 系统编程

【试验环境】

JXTA、Eclipse

【实验目的】

1. 理解 JXTA 的工作原理;
2. 掌握如何利用 JXTA 编写网络程序;
3. 掌握 Eclipse 编程环境等。

【实验要求】

1. 编写一个简单的 Hello World 的 CORBA 程序
2. 程序包含 IDL 接口定义文件;
3. 将接口定义文件编译为相应高级语言源代码, 产生服务器框架与客户端存根;
4. 基于服务器框架, 编写服务对象实现程序;
5. 基于客户端存根, 编写客户对象调用程序;

【预备知识】

CORBA (Common Object Request Broker Architecture, 公共对象请求代理体系结构, 通用对象请求代理体系结构)是由 OMG 组织制订的一种标准的面向对象应用程序体系规范。或者说 CORBA 体系结构是对象管理组织 (OMG) 为解决分布式处理环境(DCE)中, 硬件和软件系统的互连而提出的一种解决方案; OMG 组织是一个国际性的非盈利组织, 其职责是为应用开发提供一个公共框架, 制订工业指南和对象管理规范, 加快对象技术的发展。

【实验内容】

1. 编写 IDL 接口定义文件

文件名: Hello.idl
<pre>module HelloApp{ interface Hello{ string sayHello(in string sayHello); } }</pre>

程序解释: 定义一个模块 HelloApp, 这个相当于 JAVA 里面的包. 然后定义了一个接口 Hello, 该接口包含一个方法 sayHello, 用来显示 Hello, CORBA

2. 将接口定义文件编译为相应高级语言源代码, 产生服务器框架与客户端存根; Java IDL 即是 CORBA 的一个实现, 它是 JDK1.3 或更高版本的核心软件包之一, 定义在 org.omg.CORBA 及其子包中。我们利用 JAVA 提供的 IDL 编译工具对 IDL 文件进行编译. idlj -oldImplBase -fall Hello.idl

先解释一下参数的意思.-oldImplBase 表示生成与旧（1.4 版之前）JDK ORB 兼容的框架.-f 是定义要发出的绑定。-fall 表示绑定所有.包括客户端和服务端. 编译后会产生六个文件,文件的意义从别的网站 COPY 过来了

- **_HelloImplBase.java**
该抽象类是一个服务器 skeleton，它可为服务器提供基本的 CORBA 功能。它实现 Hello.java 接口。服务器类 HelloServant 扩展 _HelloImplBase。
- **_HelloStub.java**
该类是客户机 stub，可为客户机提供 CORBA 功能。它实现 Hello.java 接口。
- **Hello.java** 该接口含有 IDL 接口的 Java 版本。Hello.java 接口扩展 org.omg.CORBA.Object 并提供标准的 CORBA 对象功能。
- **HelloHelper.java**
这是一个终态类，可以提供辅助功能，特别是提供将 CORBA 对象引用转换为适当类型所需的 narrow() 方法。
- **HelloHolder.java**
这是一个终态类，其中含有 Hello 类型的公有实例成员。它可为“out”和“inout”变量提供操作。CORBA 有这些变量，但不容易映射为 Java 的语义。
- **HelloOperations.java**
这是一个接口类，其中含有方法 sayHello()。生成了相应的代码后,接下来对 Hello 接口进行实现,该类继承 _HelloImplBase 类.因为 _HelloImplBase 类是对 HELLO 接口的一个抽象的实现.新建一个 HelloImpl.java 文件,用来实

现 HELLO 接口

```

HelloImpl.java
import HelloApp.*;
public class HelloImpl extends _HelloImplBase
{
    HelloImpl()
    {
        super();
    }
    public String sayHello(String message)
    {
        System.out.println("It's server,clint is
invoking..");
        System.out.println("Hello"+message);
        return message;
    }
}

```

实现 Hello 接口后, 接下来就是对服务器端和客户端分别编写相应的代码

3. 基于服务器框架，编写服务对象实现程序； 服务器端程序编写的步骤

1. 创建一个 ORB 实例
2. 创建一个 HelloImpl 实例（CORBA Hello 对象的实现）并通知 ORB
3. 获取一个命名上下文的 CORBA 对象引用，在该命名上下文中注册新 CORBA 对象
4. 在命名上下文中将新对象注册在“Hello”名下
5. 等待对新对象的调用

按此步骤写一个 HelloServer.java 的程序

HelloServer.java
<pre> import org.omg.CosNaming.*; import org.omg.CORBA.*; public class HelloServer { public static void main(String args[]) { try{ //初始化 ORB ORB orb=ORB.init(args,null); System.out.println("ORB initial..."); //创建一个实例,并向 ORB 注册 HelloImpl helloImpl = new HelloImpl(); orb.connect(helloImpl); System.out.println("将实例注册到 ORB"); //获取一个命名上下文 org.omg.CORBA.Object obRef = orb.resolve_initial_references("NameService"); NamingContext ncRef = NamingContextHelper.narrow(obRef); //绑定命名中的对象引用 NameComponent nc = new NameComponent("Hello",""); NameComponent path[] = { nc }; ncRef.rebind(path, helloImpl); //等待客户机的调用 java.lang.Object sync = new java.lang.Object(); synchronized (sync) { sync.wait(); } System.out.println("等待 CORBA 客户端调用..."); } catch (Exception e) { System.err.println("错误: " + e); e.printStackTrace(System.out); } } } </pre>

4. 基于客户端存根，编写客户对象调用程序；

HelloClient.java

//·创建一个 ORB

```
// · 获取一个指向命名上下文的引用
// · 在命名上下文中查找 "Hello" 并获得指向该 CORBA 对象的引用
// · 调用对象的 sayHello() 操作并打印结果

import org.omg.CosNaming.*;
import org.omg.CORBA.*;
import HelloApp.*;

public class HelloClient
{
    public static void main(String args[]) {
        try{
            ORB orb = ORB.init(args, null);
            System.out.println("ORB initial...");
            //获取一个命名上下文
            org.omg.CORBA.Object obRef =
orb.resolve_initial_references("NameService");
            NamingContext ncRef = NamingContextHelper.narrow(obRef);

            NameComponent nc = new NameComponent("Hello", "");
            NameComponent path[] = { nc };
            Hello h = HelloHelper.narrow(ncRef.resolve(path));
            System.out.println("我在客户端, 开始调用 CORBA 服务器端的 'sayHello'
方法");
            System.out.println("欢迎, "
+ h.sayHello("javamxj blog"));
        }catch (Exception e) {
            System.out.println("错误 : " + e);
            e.printStackTrace(System.out);
        }
    }
}
```

5. 分别编译客户对象和服务对象程序;

程序写好后,接下来就是对程序的编译了,先做如下的准备工作.将上面编写的四个文件复制到一个新文件夹,如 d:/HelloCorba 编译 Hello.idl

```
D:\HelloCorba>idlj -oldImplBase -fall Hello.idl
```

这会生成一个 HelloApp 的目录 再对所有的 JAVA 文件进行编译

```
D:\HelloCorba>javac *.java HelloApp/*.java
```

编译中如果有问题,那代表程序中有些地方不正确,根据相应提示修改一下就可以了 接下来就可以运行程序了.为了看执行效果,将服务器端和客户端代码分开 新建一个 SERVER 的文件夹,再建一个 CLIENT 的文件夹用来表示服务器和客户. 分别在 Client 和 Server 目录下建立 HelloApp 子目录, 将

D:\CorbaSample\HelloApp 目录中的

_HelloStub.class
Hello.class
HelloHelper.class
HelloHolder.class HelloOperations.class

复制到 D:\CorbaSample\Client\HelloApp 目录下，再将 D:\CorbaSample 目录中的
HelloClient.class 复制到 D:\CorbaSample\Client 目录下。

将 D:\CorbaSample\HelloApp 目录中的

_HelloImplBase.class
Hello.class HelloOperations.class

复制到 D:\CorbaSample\Server\HelloApp 目录下，再将 D:\CorbaSample 目录中的 HelloServer.class
和 HelloImpl.class 复制到 D:\CorbaSample\Server 目录中 确保名字服务器处于运行状态：

D:\HelloCorba\Server>tnameserv -ORBInitialPort 1050

实验六 支持选举的 Java IDL 应用

1. 实验目的

掌握和运用支持 CORBA 的接口和类的 Java 基本包，熟悉支持开发 CORBA 应用的工具包 idlj 和 orbd，通过一个简单应用的构建，熟悉 Java IDL 语法，理解 CORBA 的体系结构、ORB 及其功能、IOP 及其重要性、CORBA 对象引用和互操作对象引用协议、CORBA 名字服务和互操作名字服务、标准 CORBA 对象服务及如何提供这些服务等内容。

2. 实验内容

创建一个支持选举的 Java IDL 应用。服务器输出两个方法：

(1) getList，返回一个列表，列出所有候选人及其得票数；

(2) castVote，以参数形式接受一个包含候选人姓名的字符串，该方法没有返回值；

先在一台及其上运行所有的进程来测试应用，然后通过在不同机器上运行客户和服务
器来测试系统。

4、实验环境

(1) Windows XP/NT/2000

(2) Eclipse

3. 实验要求

(1) 编写一个基于命令行交互的客户程序来访问选举服务。客户程序通过命令行参数访问
远程服务。

(2) 严格按照编码规范进行编码实现。（请到服务器上下载）；

(3) 按照面向对象的编程思想来设计和实现；

(4) 提交的结果包括：

⌚ 源代码

⌚ 编译运行与调试有关的批处理文件

i. 编译 idl 文件的脚本文件 build.bat；

ii. 在服务器端启动命名服务的脚本文件 runOrbd.bat；

iii. 在服务器端启动服务器的脚本文件 runServer.bat； iv. 在客户端启动客户端
的脚本文件 runClient.bat；

⌚ 其他文档：

i. 在 doc 子目录中存放上述源代码生成的 javadoc 文档；

实验七 Internet 应用

实验目的

1. 了解基于 http 协议的 web 应用的工作原理
2. 了解构建基于 web 的分布式应用范型的主要技术和手段

预习与实验要求

1. 预习实验指导书及教材的有关内容，了解 http 协议工作原理；
2. 了解 html 和 XML 语言；
3. 尽可能独立思考并完成实验。

实验环境

- a) 独立计算机；
- b) Windows 操作系统。
- c) Tomcat
- d) J2EE 工具包
- e) J2sdk 工具包

实验原理

基于 http 的 web 应用是分布式系统的另一个重要应用范型，与 C/S 模式不同的是 B/S 模式的应用没有独立的客户端软件，B/S 模式中统一对服务器端响应进行解释的就只有浏览器。因为浏览器的通用性的简单性，使得 B/S 应用中数据传递也不可能很复杂，因此就决定了它在应用层的支持协议只能是面向文本的 http 协议。

最初的 http 协议只能支持简单的静态页面（对象）的获得，但是随着网络应用的发展，提供与客户的接口，使得客户与服务器端的通信能够动态进行，并由此动态生成页面（对象）是必要的。解决这一问题的主要手段是使用表单技术。而保存与客户交互的结果也是必须的，解决的技术是 cookie。

实验内容

1. 使用 socket API 实现简单的 HTTP 客户
 - (1) 使用 sendMessage 方法发送符合 http 协议定义的消息给 web 服务器以获得想要的页面
 - (2) 使用 receiveMessage 方法接收返回的页面源文件并显示。
2. 使用隐式表单域传输会话状态数据

- (1) 编写客户端请求页面 `form.html`
 - (2) 编写 `form.html` 所触发的脚本 `hiddenform.c`
 - (3) 编写 `hiddform.cgi` 动态生成页面中触发的脚本 `hidden.form2.c`
3. 使用 `cookie` 传递状态数据
- (1) 编写请求页面 `cookie.html`
 - (2) 编写 `cookie.html` 所触发的脚本 `cookie.c`
 - (3) 编写由 `cookie.c` 动态生成页面中触发的脚本 `cookie2.c`

实验报告

1. 列明文件清单
2. 写明各文件的作用和文件间的关系
3. 写出各文件的伪代码

思考题

1. 考虑隐式表单域与 `cookie` 技术之间的区别
2. 考虑 `cookie` 的安全性问题

实验八 实现一个基本的 Web 服务器程序

【实验目的及要求】

采用 Socket API 知识和对 HTTP 协议, CGI 的理解, 实现一个基本的 WEB 服务器程序, 要求服务器能成功响应客户程序发来的 GET 命令 (传送文件), 进一步实现响应 POST 和 GET 命令的 CGI 程序调用请求。

要求: 要求独立完成。

【实验原理和步骤】

1. 实验原理

(1) 服务器主要监听来至客户浏览器或是客户端程序的连接请求, 并且接收到客户请求后对客户请求作出响应。如果请求是静态的文本或是网页则将内容发送给客户。如果是 CGI 程序则服务器调用请求的 CGI 程序, 并发送结果给客户。

(2) HTTP 协议是基于 TCP/IP 协议之上的协议, 是 Web 浏览器和 Web 服务器之间的应用层协议, 是通用的、无状态的、面向对象的协议。

(3) HTTP 的请求一般是 GET 或 POST 命令 (POST 用于 FORM 参数的传递)。GET 命令的格式为

GET 路径/文件名 HTTP/1.0

文件名指出所访问的文件, HTTP/1.0 指出 Web 浏览器使用的 HTTP 版本。

(4) Web 浏览器提交请求后, 通过 HTTP 协议传送给 Web 服务器。Web 服务器接到后, 进行事务处理, 处理结果又通过 HTTP 传回给 Web 浏览器, 从而在 Web 浏览器上显示出所请求的页面。

在发送内容之前 Web 服务器首先传送一些 HTTP 头信息:

HTTP 1.0 200 OK

WEBServer: 1.0 // 服务器类型

content_type:类型 content_length:长度
值

(5) 响应 POST 和 GET 命令的 CGI 程序调用请求需要服务器执行外部程序, Java 执行外部可执行程序的方法是: 首先通过 `Runtime run = Runtime.getRuntime()` 返回与当前 Java 应用程序相关的运行时对象; 然后调用 `Process CGI = run.exec(ProgramName)` 另启一个进程来执行一个外部可执行程序。

2. Web 服务器的实现步骤:

- (1) 创建 `ServerSocket` 类对象, 监听端口 8080。这是为了区别于 HTTP 的标准 TCP/IP 端口 80 而取的;
- (2) 等待、接受客户机连接到端口 8080, 得到与客户机连接的 `socket`;
- (3) 创建与 `socket` 字相关联的输入流和输出流

(4) 从与 socket 关联的输入流 instream 中读取一行客户机提交的请求信息, 请求信息的格式为: GET 路径/文件名 HTTP/1.0

(5) 从请求信息中获取请求类型。如果请求类型是 GET, 则从请求信息中获取所访问的文件名。没有 HTML 文件名时, 则以 index.html 作为文件名;

(6) 如果请求文件是 CGI 程序则调用它, 并把结果通过 socket 传回给 Web 浏览器, (此处只能是静态的 CGI 程序, 因为本设计不涉及传递环境变量) 然后关闭文件。否则发送错误信息给 Web 浏览器;

(7) 关闭与相应 Web 浏览器连接的 socket 字。

【实验任务】

1. 提交源代码以及实验报告。

实验方案二、

1. 编写一个基于 Web Service 的应用;
2. 调试代码并测试运行结果;
3. 使用客户端对 Web Service 进行调用。

【预备知识】

Web Service 也叫 XML Web Service WebService 是一种可以接收从 Internet 或者 Intranet 上的其它系统中传递过来的请求, 轻量级的独立的通讯技术。是通过 SOAP 在 Web 上提供的软件服务, 使用 WSDL 文件进行说明, 并通过 UDDI 进行注册。


XML: (Extensible Markup Language)扩展型可标记语言。面向短期的临时数据处理、面向万维网络, 是 Soap 的基础。

Soap: (Simple Object Access Protocol)简单对象存取协议。是 XML Web Service 的通信协议。当用户通过 UDDI 找到你的 WSDL 描述文档后, 他通过可以 SOAP 调用你建立的 Web 服务中的一个或多个操作。SOAP 是 XML 文档形式的调用方法的规范, 它可以支持不同的底层接口, 像 HTTP(S)或者 SMTP。

WSDL: (Web Services Description Language) WSDL 文件是一个 XML 文档, 用于说明一组 SOAP 消息以及如何交换这些消息。大多数情况下由软件自动生成和使用。

UDDI (Universal Description, Discovery, and Integration) 是一个主要针对 Web 服务供应商和使用者的新项目。在用户能够调用 Web 服务之前, 必须确定这个服务内包含哪些商务方法, 找到被调用的接口定义, 还要在服务端来编制软件, UDDI 是一种根据描述文档来引导系统查找相应服务的机制。UDDI 利用 SOAP 消息机制 (标准的 XML/HTTP) 来发布, 编辑, 浏览以及查找注册信息。它采用 XML 格式来封装各种不同类型的数据, 并且发送到注册中心或者由注册中心来返回需要的数据。

【实验内容】

 在.NET 中创建 Web Service

```
<% @ WebService Language="C#" class="SecurityWebService" %>
```

 通知编译器运行 Web Service 模式, 还有 c#类名

```
using System;  
using System.Web.Services;
```

SecurityWebService 应该继承了 Web Service 类的功能

```
public class SecurityWebService : WebService
```

将默认生成的服务进行改写

```

[WebMethod]
public string ReverseString(string message)
{
    char[] arr = message.ToCharArray();
    Array.Reverse(arr);
    message = new string(arr);
    return message;
}

```

运行该程序，并验证结果 在运行的过程中查看 Soap 协议

Service

单击[此处](#)，获取完整的操作列表。

ReverseString

测试

若要使用 HTTP POST 协议对操作进行测试，请单击“调用”按钮。

参数	值
message:	<input type="text"/>

调用

SOAP 1.1

以下是 SOAP 1.2 请求和响应示例。所显示的占位符需替换为实际值。

```

POST /WebSite1/Service.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/ReverseString"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  <soap:Body>
    <ReverseString xmlns="http://tempuri.org/"
      <message>string</message>
    </ReverseString>
  </soap:Body>
</soap:Envelope>

```

ReverseString

测试

若要使用 HTTP POST 协议对操作进行测试，请单击“调用”按钮。

参数	值
message:	dwcxssw

调用

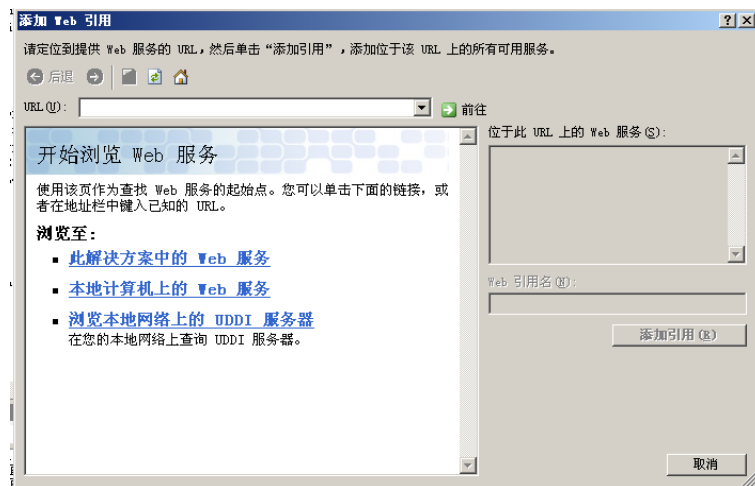
```

<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">wssxcwd</string>

```

在 Windows Forms 中调用 Web 服务

新建 Windows Forms 工程，注意上面服务不要关。项目-添加服务引用，地址中输入 Web 服务的地址，上例：http://localhost:3765/WebSite1/Service.asmx，如果 Web 服务已经发布，请填写发布的地址。



在 Form 上加入两个 TextBox，一个 Button，双击 Button，编写事件。

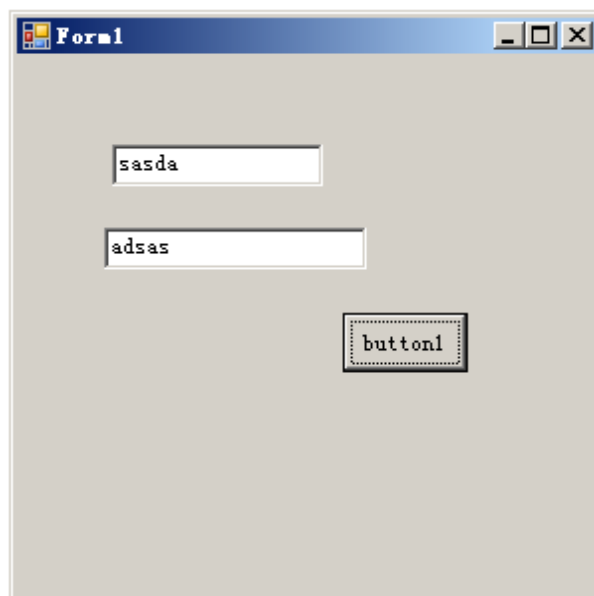
```
using System.Text;
using System.Windows.Forms;

namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            localhost.Service ws = new localhost.Service();

            textBox2.Text = ws.ReverseString(textBox1.Text);
        }
    }
}
```

运行结果：



实验八 实现一个简单的 Web 应用程序(2)

【实验目的及要求】

该实验内容为通过 WEB 程序实现数据库的增、删、改和查询功能，并能使用 HttpSession 或 Cookie 实现会话状态数据管理，整个实验可以选择 J2EE/JSP 或.NET 实现。该实验内容为通过 WEB 程序实现数据库的增、删、改和查询功能，并能使用 HttpSession 或 Cookie 实现会话状态数据管理。

要求：独立完成或合作完成。

【实验原理和步骤】

1.实验原理

采用 J2EE/JSP 的技术实现，难度分成 3 个等级：a.采用基本 JSP 实现；b. 采用 JSP +STRUTS 框架实现；c. 采用 JSP+STRUTS+iBtais/ Hibernate 框架实现。

2.WEB 应用的实现步骤：

- (1) 建立数据库，可以采用 MYASQL、SQL Server 或 ACCESS;
- (2) 搭建 WEB 应用框架;
- (3) 开发网页和服务程序，实现业务功能;
- (4) 部署系统;
- (5) 测试系统。

【实验任务】

- 1.提交源代码以及实验报告。

实验九 Google AppEngine 程序设计

试验环境

计算机、AppEngine 软件开发包、Python 2.5

实验目的

1. 理解云计算的基本理论知识;
2. 掌握 AppEngine 软件包的基本使用;
3. 运用 AppEngine 进行简单的留言本的开发。

实验要求

1. 下载 AppEngine 和 Python2.5, ;
2. 申请 AppEngine 帐号;
3. 编写程序, 并在调试完毕后上传到服务器上。

预备知识

✚ Google App Engine 用于让开发者进行网站应用程序开发或上传已经完成的应用。该开发包包含: 一个 web 服务程序, 用来模拟 App Engine 应用环境; 一个本地版的数据存储方案; 本地模拟的 Google 帐号集成; 支持使用 Api 来分析 URL 和发送邮件。这个开发包可以运行在所

有安装了 Python2.5 的机器上, 并且支持 Windows, Mac OS X 和 Linux 系统。

✚ Google App Engine 应用通过 CGI 标准协议与服务器通讯。这是一个标准的 Http 处理流程, Web 服务接受到客户端发来的 Get 或 Post 请求, web 服务器把请求转发给你的应用程序, 由应用程序来处理要输出的内容。

✚ webapp 应用包含三部分: 一个或多个 RequestHandler 类用来处理 http 请求和应答; 一个 WSGIApplication 实例, 根据不同的 URL 请求, 将处理交给不同的 RequestHandler 类实例; 一个主过程, 通过 CGI adaptor 方式运行 WSGIApplication

实验内容

✚ 创建一个简单的 Request Handler

首先创建一个名为 helloworld 的文件夹。除非特殊说明, 以后所有关于这个应用程序的文件都将放在这个文件夹里面。在 helloworld 文件夹里, 创建一个新文件 helloworld.py, 文件内容如下:

```
class="prettyprint"print 'Content-Type: text/plain' print "  
print 'Hello, world!'
```

这个 Python 脚本处理一个 request 请求, 并且设置一个 Http header, 输出一个空行和一段信息 Hello, world!.

创建配置文件: 每个 App Engine application 都包含一个名为 app.yaml 的配置文件。在这个配置文件中, 可以设置具体的某个 URL 需要用哪个 Python 脚本来处理。现在, 在 helloworld 文件夹中, 创建一个新的 app.yaml 文件, 输入以下内容:

```

application: helloworld
version: 1 runtime:
python api_version: 1
handlers:
- url: /* script:
helloworld.py

```

这个配置文件描述了以下内容：这个应用程序的标识是 helloworld. 这个标识需要和你在 App Engine 网站上创建的应用程序标识保持一致。在开发期间你可以使用任何你喜欢的名字，但是上传的时候，必须要和你在 App Engine 注册的标识保持一致。现在，我们把它设置为 helloworld. 你的应用程序的版本号为 1，如果你在上传应用之前修改了这个编号，App Engine 将会自动保留前一个版本的副本，以方便你可以在管理平台中将当前版本恢复成原来的版

本。该应用运行在 python 环境, 环境版本是 1. 目前只有 Python 可选，将来会提供更多的运行环境和开发语言. 所有符合正则表达式/* (所有 URL) 的请求，都由 helloworld.py 脚本来处理. 该配置文件使用 YAML 语法. 可以在本地 App Engine SDK 环境中进行模拟运行测试。首先，指定应用路径为 helloworld 目录，使用下面的命令启动测试环境 Web 服务程序，

google_appengine/dev_appserver.py helloworld/ 这个 Web 服务程序将监听 8080 端口. 你可以在浏览器中输入以下地址进行测试: <http://localhost:8080/> 创建一个网络记事本

创建 guestbook 文件夹，包含文件如下：app.yaml; guestbook.py

```

application: networkdevelop
version: 1
runtime: python
api_version: 1
handlers:
- url: /*
script: guestbook.py

```

图表 app.yaml

```

1 #!/usr/bin/env python
2 #
3 # Copyright 2007 Google Inc.
4 #
5 # Licensed under the Apache License, Version 2.0 (the "License");
6 # you may not use this file except in compliance with the License.
7 # You may obtain a copy of the License at
8 #
9 #     http://www.apache.org/licenses/LICENSE-2.0
10 #
11 # Unless required by applicable law or agreed to in writing, software
12 # distributed under the License is distributed on an "AS IS" BASIS,
13 # WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
14 # See the License for the specific language governing permissions and
15 # limitations under the License.
16 #
17 import cgi
18 import datetime
19 import wsgiref.handlers
20
21 from google.appengine.ext import db
22 from google.appengine.api import users
23 from google.appengine.ext import webapp
24
25 class Greeting(db.Model):
26     author = db.UserProperty()
27     content = db.StringProperty(multiline=True)
28     date = db.DateTimeProperty(auto_now_add=True)
29
30
31 class MainPage(webapp.RequestHandler):
32     def get(self):
33         self.response.out.write('<html><body>')
34

```

```

35 greetings = db.GqlQuery("SELECT * "
36                          "FROM Greeting "
37                          "ORDER BY date DESC LIMIT 10")
38
39 for greeting in greetings:
40     if greeting.author:
41         self.response.out.write('<b>%s</b> wrote:' % greeting.author.nickname())
42     else:
43         self.response.out.write('&#xA0;An anonymous person wrote:')
44         self.response.out.write('<blockquote>%s</blockquote>' %
45                                 cgi.escape(greeting.content))
46
47 self.response.out.write("""
48     <form action="/sign" method="post">
49         <div><textarea name="content" rows="3" cols="60"></textarea></div>
50         <div><input type="submit" value="Sign Guestbook"></div>
51     </form>
52 </body>
53 </html>""")
54
55
56 class Guestbook(webapp.RequestHandler):
57     def post(self):
58         greeting = Greeting()
59
60         if users.get_current_user():
61             greeting.author = users.get_current_user()
62
63         greeting.content = self.request.get('content')
64         greeting.put()
65         self.redirect('/')
66
67 application = webapp.WSGIApplication([
68
69     ('/', MainPage),
70     ('/sign', Guestbook)
71 ], debug=True)
72
73
74 def main():
75     wsgiref.handlers.CGIHandler().run(application)
76
77
78 if __name__ == '__main__':
79     main()
80

```

图表 2guestbook.py

注册用户

访问 <http://appengine.google.com/>, 使用你的 Google 帐号登录到 App Engine 管理平台。(如果你还没有 Google 帐号, 请先申请一个)为了创建一个新的 GAE 应用, 请点击按钮"Create an Application", 按照提示注册应用程序 ID,应用程序 ID 的名字必须是唯一的。创建 ID 后, 你就可以拥有一个 <http://applicationid.appspot.com/> 这样的 URL 地址来访问你的 WEB 应用了.当然, 如果你拥有自己的域名的话, 也可以将其绑定到你自己的应用。上传应用程序 执行命令行程序:appcfg.py update helloworld/按照提示, 输入您自己的 Google 用户名和密码. 现在你已经可以使用如下地址访问您刚刚上传的 WEB 应用了 <http://application-id.appspot.com>

实验十 安装 Xen 熟悉常用命令

【实验目的】

- 1,掌握在 Linux Xen 的安装方法
- 2, 熟悉 Xen 的常用命令

【相关的知识与要点】

1. Xen 简介

XEN 一开始是由英国剑桥大学的实验室催生出来的，目前已经交由 XEN 的社区所管理。通常 Linux 的软件都会交由社区来协助发展，一方面是人数较多，另一方面也是可以直接推广。因为社区的支持一直都是 Linux 的一项优势，通过社区可以实现许多一般公司无法轻易达到的目标，如跨国的技术支持，以及众多技术人员的支持，或是快速的更新问题。

XEN 和其他的虚拟机软件有一个最大的不同点，在于它提供了一项 para mode（para virtualization），只要当初系统有使用 XEN 专有的虚拟机技术去开发，那么，在 XEN 之下就可以得到非常高的运行性能。其实这都与“para”有关，代表着通过特有的技术，该系统在 XEN 之下不需要仿真太多的硬件，因为仿真越多硬件，就一定会消耗越多的系统资源。

目前，XEN 虚拟机有两种运行方式：完全虚拟化（full virtualization）和半虚拟化(para virtualization)（图 -1）。

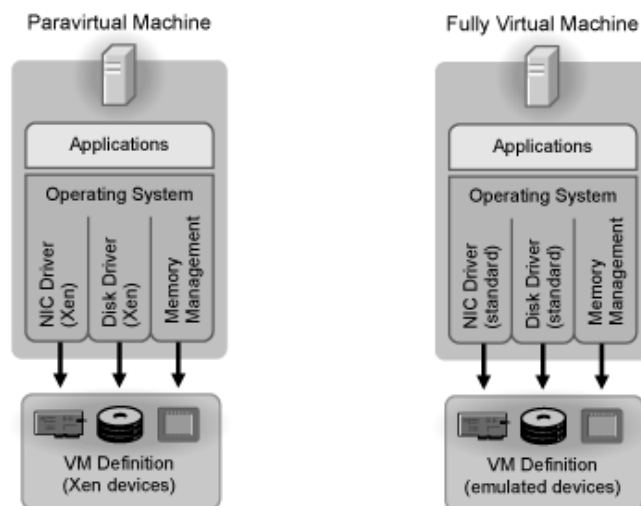


图 1 XEN 两种运行方式

1、完全虚拟化提供底层物理系统的全部抽象化，且创建一个新的虚拟系统，客户机操作系统可以在里面运行。不需要对客户机操作系统或者应用程序进行修改（客户机操作系统或者应用程序像往常一样运行，意识不到虚拟环境的存在）。可以创建 Linux、FreeBSD 和 Windows 客户机。

2、半虚拟化需要对运行在虚拟机上的客户机操作系统进行修改（这些客户机操作系统会意识到他们运行在虚拟环境里）并提供相近的性能，但半虚拟化的性能要比完全虚拟化更优越，支持 RHEL4.5 以上版本的客户机

在每个客户虚拟机支持到 32 个虚拟 CPU，通过 VCPU 热插拔），支持 PAE 指令集的 x86/32, x86/64 平台，通过 Intel 虚拟支持 VT 的支持来用虚拟原始操作系统（未经修改的）支持（包括 Microsoft Windows） 优秀的硬件支持.支持几乎所有的 Linux 设备驱动

【实验环境】

Linux enterprise Redhad 5 or SUSE

【实验内容】

安装 Xen 虚拟机

启动 Xen 服务器

列出所有正在运行的虚拟操作系统 创建新的虚拟机并查看所有虚拟系统运行的状态 调整虚拟平台及虚拟操作系统的虚拟 CPU 个数 调整虚拟平台及虚拟操作系统的虚拟内存大小 停止虚拟的系统

参考： Xen 服务器的启动：

xend 服务器的启动/停止/重启/状态查询，请用下面的命令：

[root@localhost ~]# /etc/init.d/xend start 启动 xend，如果 xend 没有运行）

[root@localhost ~]# /etc/init.d/xend stop 停止 xend，如果 xend 正在运行）

[root@localhost ~]# /etc/init.d/xend restart 重启正在运行的 xend，如果 xend 没有运行，

则启动

```
[root@localhost ~]# /etc/init.d/xend status 查看 xend 状态
```

启动 xendomains 服务器的启动/停止/重启/状态查询, 请用下面的命令; 一般的情况下, xend 服务器启动了, xendomains 也会自动启动。 [root@localhost ~]# /etc/init.d/xend start 启动 xend, 如果 xend 没有运行)

```
[root@localhost ~]# /etc/init.d/xend stop 停止 xend, 如果 xend 正在运行)
```

[root@localhost ~]# /etc/init.d/xend restart 重启正在运行的 xend, 如果 xend 没有运行, 则启动

```
[root@localhost ~]# /etc/init.d/xend status 查看 xend 状态
```

```
[root@localhost ~]# /etc/init.d/xendomains start
```

```
[root@localhost ~]# /etc/init.d/xendomains stop
```

```
[root@localhost ~]# /etc/init.d/xendomains restart
```

```
[root@localhost ~]# /etc/init.d/xendomains status
```

Xen 管理工具 xm:

列出所有正在运行的虚拟操作系统:

```
[root@localhost ~]# /usr/sbin/xm list
```

```
Name ID Mem(MiB) VCPUs State Time(s)
```

```
Domain-0 0 450 1 r----- 5377.0
```

```
fc5 4 256 1 -b---- 0.1
```

关闭被虚拟的系统:

```
[root@localhost ~]# /usr/sbin/xm shutdown 虚拟操作系统的 Name 或 DomID
```

```
[root@localhost ~]# /usr/sbin/xm destroy 立即停止虚拟的系统 (重要)
```

调整虚拟平台及虚拟操作系统的虚拟 CPU 个数;

```
[root@localhost ~]# /usr/sbin/xm vcpu-set
```

举例:

```
[root@localhost ~]# /usr/sbin/xm list
```

```
Name ID Mem(MiB) VCPUs State Time(s)
```

```
Domain-0 0 458 1 r----- 260.3
```

```
fc5 2 256 1 -----6.5
```

```
[root@localhost ~]# /usr/sbin/xm vcpu-set 2 4
```

查看虚拟系统运行的状态:

```
[root@localhost ~]# xm top 或 [root@localhost ~]# xentop
```

实验十 基于 Xen 虚拟机实现 tomcat cluster

【实验目的】

- 1, 掌握 tomcat 在 linux 上的安装
- 2, 掌握构建 tomcat cluster 的过程
- 3, 实现 tomcat cluster 中 session 同步

【相关的知识与要点】

Apache Tomcat 是由 Apache 软件基金会下属的 Jakarta 项目开发的一个 Servlet 容器, 按照 Sun Microsystems 提供的技术规范, 实现了对 Servlet 和 JavaServer Page (JSP) 的支持, 并提供了作为 Web 服务器的一些特有功能, 如 Tomcat 管理和控制平台、安全域管理和 Tomcat 阀等。由于 Tomcat 本身也内含了一个 HTTP 服务器, 它也可以被视作一个单独的 Web 服务器。但是, 不能将 Tomcat 和 Apache Web 服务器混淆, Apache Web Server 是一个用 C 语言实现的 HTTP web server; 这两个 HTTP web server 不是捆绑在一起的。Apache Tomcat 包含了一个配置管理工具, 也可以通过编辑 XML 格式的配置文件来进行配置。

【实验环境】

安装有 Xen 的 linux 系统

【实验内容】

1. 利用 Xen 建立 2 个或 2 个以上虚拟机
2. 在每一台虚拟机种安装 Tomcat
3. 测试 tomcat 是否安装成功
4. 配置 Tomcat 修改 works.properties 文件和 conf/server.xml 文件
5. 配置负载均衡器 apache httpd.conf
6. 启动集群服务器 a 和 b 测试集群是否建立
7. 进行 session 同步实验

实验十一 GFS 的安装与配置

1. GFS 简介

GFS (Global File System), 是 Minnesota 大学开发的基于 SAN 的共享存储的机群文件系统, 后来 Sistina 公司将 GFS 产品化。GFS 在很长一段时间都是以源代码开放软件的形式出现的, 后来由于 Sistina 希望通过向用户提供支持和服务的计划未能取得成功, 为了要促进自己的财务收入, Sistina 在 2001 年将 GFS 变成了一种“专有软件”。

GFS 文件系统其实是一个网络日志文件系统, 通常被用作多台计算机共享同一存储设备。由于 GFS 是日志文件系统, 所以, 如果将其应用到单独的一台计算机上, 即完全等同于本地日志文件系统, 享受日志文件系统带来的好处。

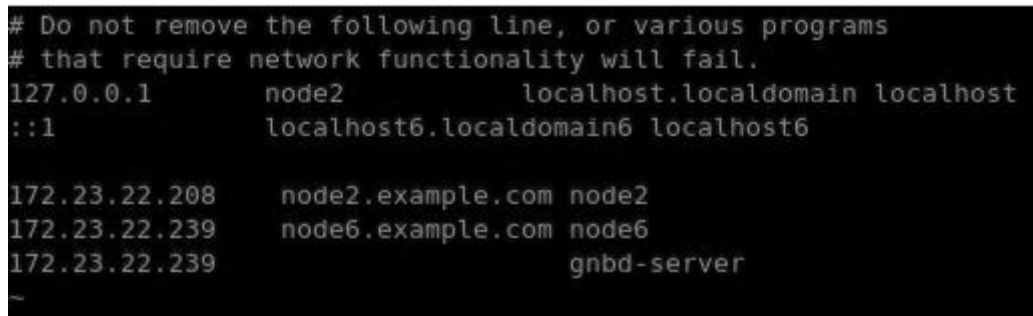
2. GFS 需要的安装包

GFS 采用 rpm 包安装模式进行安装, 采用命令 `rpm -ivh *.rpm`, 需要下载并安装的包有:

- 1、rgmanager — 管理集群系统的服务与资源;
- 2、system-config-cluster — 图形化的 cluster 配置工具, 可以配置本机上的集群名字, 资源, fence 和服务等;
- 3、ccsd — Cluster Configuration Services Daemon, 集群配置服务守护进程和相关文件;
- 4、magma — 包括集群锁管理在内的界面库;
- 5、cman — 包括控制集群成员、信息、通知的集群管理器;
- 6、dlm — 包括分布式锁管理库;
- 7、fence — 管理集群节点之间网络切换、光纤网络切换和集成电源管理接口的 I/O 防护系统;
- 8、iddev — 包括用来识别设备格式化的文件系统的库;
- 9、GFS — Global File System, 红帽全局文件系统模块;
- 10、gnbd — Red Hat GFS 网络块设备模块;
- 11、lvm2-cluster — 逻辑卷管理的集群扩展模块。

3. GFS 配置安装过程

- 1、修改每个节点上/etc/hosts 设置, 如图 2-1 所示;



```
# Do not remove the following line, or various programs
# that require network functionality will fail.
127.0.0.1      node2          localhost.localdomain localhost
::1           localhost6.localdomain6 localhost6

172.23.22.208  node2.example.com node2
172.23.22.239  node6.example.com node6
172.23.22.239  gnbd-server
```

图 12-1 修改每个节点上/etc/hosts 设置

- 2、通过 system-config-cluster 对 cluster.conf 文件进行配置, 加载各个节点, 配置 fence 配置, 以及设置 failover domains 和 resources, 并将配置好的 cluster.conf 文件通过 scp 发送到其他节点上 (图 2-2);

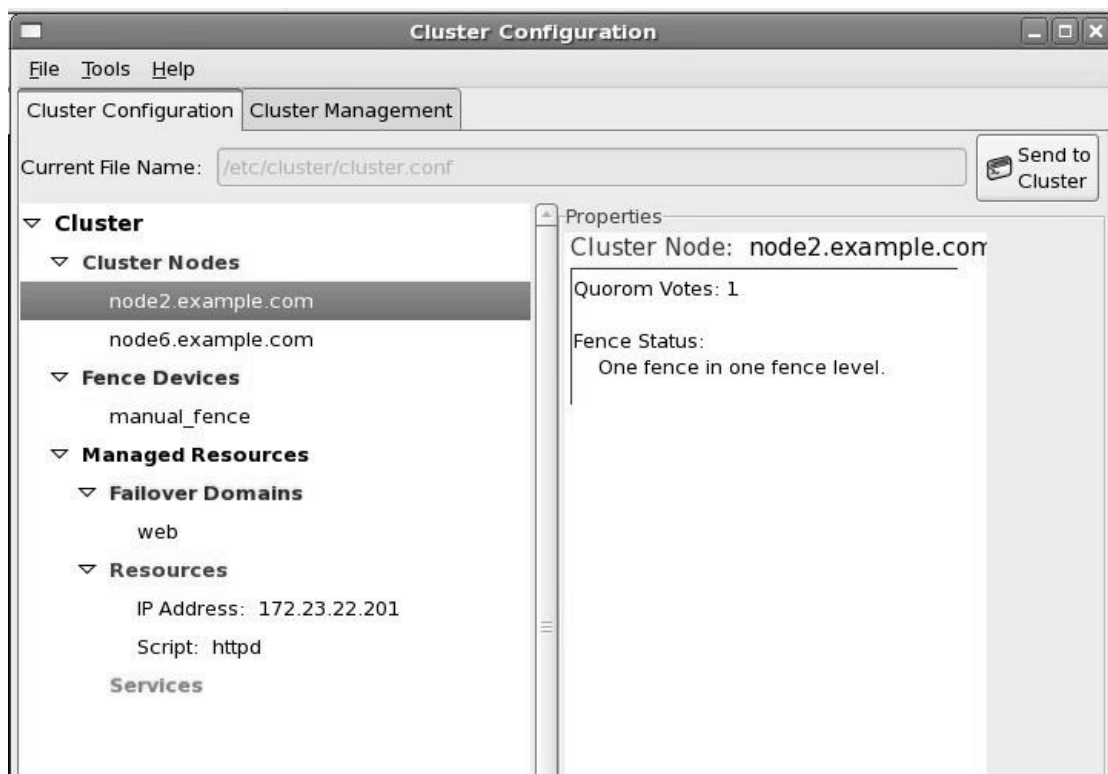


图 12-2 使用 system-config-cluster 配置节点等

3、在各个节点上启动 dlm、ccsd、cman、fence 服务（图 2-3），在启动这些服务之前最好关掉防火墙 iptables(如果已经开放了相应的端口，则不需要停止防火墙)；

```
[root@node6 ~]# modprobe lock_dlm
[root@node6 ~]# service iptables stop
Flushing firewall rules: [ OK ]
Setting chains to policy ACCEPT: filter [ OK ]
Unloading iptables modules: [ OK ]
[root@node6 ~]# service cman start
Starting cluster:
  Loading modules... done
  Mounting configfs... done
  Starting ccscd... done
  Starting cman... done
  Starting daemons... done
  Starting fencing... done
[ OK ]
[root@node6 ~]#
```

图 12-3 在各个节点上启动 dlm、ccsd、cman、fence 服务

4、在 gnbd-server 上导出设备（图 2-4），并在各节点导入设备（图 2-5）；

```
[root@node6 ~]# gnbd_export -v -e gfs -d /dev/sda3 -c
gnbd_export: created GNBD gfs serving file /dev/sda3
[root@node6 ~]#
```

图 12-4 在 gnbd-server 上导出设备

```
[root@node6 ~]# modprobe gnbd
[root@node6 ~]# gnbd_import -v -i gnbd-server
gnbd_import: created gnbd device gfs
gnbd_recvd: gnbd_recvd started
[root@node6 ~]#
```

图 12-5 在各节点导入设备

5、加载 GFS 服务，并用 `gfs_mkfs` 命令在 `gnbd-server` 上建立 `gfs` 文件系统（图 2-6）。

```
[root@node6 gnbd]# gfs_mkfs -p lock_dlm -t GFS:gfs -j 2 /dev/gnbd/gfs
This will destroy any data on /dev/gnbd/gfs.
  It appears to contain a gfs filesystem.

Are you sure you want to proceed? [y/n] y

Device:                /dev/gnbd/gfs
Blocksize:             4096
Filesystem Size:       1259688
Journals:              2
Resource Groups:       20
Locking Protocol:      lock_dlm
Lock Table:            GFS:gfs

Syncing...
All Done
```

图 12-6 GFS 创建完成

为了让以后机器启动以后自动加载 GFS 文件系统，需要改写 `/etc/fstab` 文件，加以下内容：
`/dev/hda8 /GFS default 0 0`
以后系统启动时就会自动加载 GFS 文件系统了。

实验十二 XAMPP 的安装与使用

【实验目的】

- (1) 了解常用服务器软件的工作原理与工作机制。
- (2) 掌握 XAMPP 的下载、安装、配置与使用；
- (3) 掌握 Apache、MySQL、Filezilla 软件的配置与使用方法。

【实验内容和步骤】

1 下载

XAMPP 官方网址: <http://www.apachefriends.org/> 可以下载最新版本的 XAMPP

2.1 安装

先将 xampp 压缩包解压, 根据解压放置的位置, 分为两种方式。若将 xampp 解压在非根目录下, 就需要运行.\xampp\setup_xampp.bat 进行安装配置; 若将 xampp 解压在分区根目录下, 则不需要运行.\xampp\setup_xampp.bat 进行安装配置。

虽然将 xampp 解压在根目录下, 不需要运行.\xampp\setup_xampp.bat 进行配置, Apache、MySQL 和 Mercury 邮件服务器能够正确启动, 但 FileZilla FTP 服务器不会启动, 因为它需要绝对路径。

2.2 启动

.\xampp\xampp-control.exe

其它的服务启动/停止脚本

启动 Apache 和 MySQL: .\xampp\xampp_start.exe

停止 Apache 和 MySQL: .\xampp\xampp_stop.exe

启动 Apache: .\xampp\apache_start.bat

停止 Apache: .\xampp\apache_stop.bat

启动 MySQL: .\xampp\mysql_start.bat

停止 MySQL: .\xampp\mysql_stop.bat

启动 Mercury 邮件服务器: .\xampp\mercury_start.bat

(Mercury 邮件服务器只能通过 XAMPP 控制面板的图形界面停止)

设置 FileZilla FTP 服务器: .\xampp\filezilla_setup.bat

启动 FileZilla FTP 服务器: .\xampp\filezilla_start.bat

停止 FileZilla FTP 服务器: .\xampp\filezilla_stop.bat

2.3 安装服务

您可以在 NT4、2000 和 XP 平台中将特定的服务器配置为系统服务。请使用以下脚本:

安装 Apache 服务器为系统服务: .\xampp\apache\apache_installservice.bat

卸载 Apache 服务器的系统服务: .\xampp\apache\apache_uninstallservice.bat

安装 MySQL 服务器为系统服务：.\xampp\mysql\mysql_installservice.bat
卸载 MySQL 服务器的系统服务：.\xampp\mysql\mysql_uninstallservice.bat
安装及卸载 FileZilla FTP 服务器为系统服务：.\xampp\filezilla_setup.bat
Mercury 邮件服务器：目前还不能配置为系统服务！

3 XAMPP 的配置

3.1 默认安全配置

下面是 XAMPP 默认配置的安全问题列表：

- MySQL 管理员（root）未设置密码。
- MySQL 服务器可以通过网络访问。
- PhpMyAdmin 可以通过网络访问。 样例可以通过网络访问。

Mercury 邮件服务器和 FileZilla FTP 服务器的用户是公开的。

安全配置地址：<http://127.0.0.1/security>，MySQL、PhpMyAdmin 的管理员密码和 XAMPP 的目录保护可以在这里设置。对于 Mercury 邮件服务器和 FileZilla FTP 服务器，请记得更改配置设置（比如用户名和密码）。如果您不需要这些服务，那就不要启动它们——这样也是安全的。

3.2 配置文件位置

您可以通过文本编辑器来更改 XAMPP 的各种配置文件。这些文件存在于以下路径：

- Apache 基本配置：.\xampp\apache\conf\httpd.conf
- Apache SSL：.\xampp\apache\conf\ssl.conf
- Apache Perl（仅限插件）：.\xampp\apache\conf\perl.conf
- Apache Tomcat（仅限插件）：.\xampp\apache\conf\java.conf
- Apache Python（仅限插件）：.\xampp\apache\conf\python.conf
- PHP：.\xampp\php\php.ini
- MySQL：.\xampp\mysql\bin\my.ini
- phpMyAdmin：.\xampp\phpMyAdmin\config.inc.php
- FileZilla FTP 服务器：.\xampp\FileZillaFTP\FileZilla Server.xml
- Mercury 邮件服务器基本配置：.\xampp\MercuryMail\MERCURY.INI
- Sendmail：.\xampp\sendmail\sendmail.ini

3.3 目录说明

路径	内容
.\xampp\anonymous	匿名 FTP 的样例文件夹
.\xampp\apache	Apache 服务器

\xampp\cgi-bin	可执行的 CGI 脚本
\xampp\FileZillaFTP	FileZilla FTP 服务器
\xampp\htdocs	http 文档的主文件夹
\xampp\install	用于 XAMPP 的安装（请勿删除！）
\xampp\licenses	同上
\xampp\MercuryMail	Mercury 邮件 SMTP POP3 IMAP 服务器
\xampp\mysql	MySQL 服务器
\xampp\perl	Perl
\xampp\php	PHP（4 和 5）
\xampp\phpmyadmin	phpMyAdmin
\xampp\security	额外的安全配置
\xampp\tmp	临时文件夹
\xampp\webalizer	Webalizer 网络状态
\xampp\webdav	WebDAV 样例

实验十三 虚拟机的使用与 Linux 系统的安装

【实验目的】

- (1) 理解虚拟机软件的工作原理与运行机制;
- (2) 掌握 VMware Workstation 的下载、安装、配置与使用;
- (3) 掌握 Linux 系统 (Ubuntu) 的配置与使用方法。

【实验内容和步骤】

- 1 可以到官方网站下载 VMware9.0 虚拟机安装软件, 直接点 next 下一步直接安装完成即可
- 2 添加设备, 点击 add, 添加过程就像刚开始配置安装虚拟机过程一样。
- 3 创建一块磁盘。

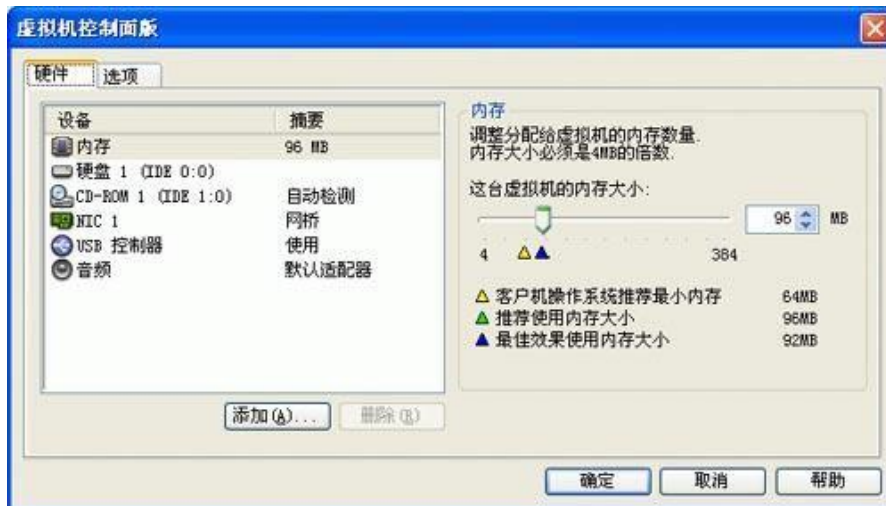
3.1 当第一次建立虚拟机时, 请选择第一项, 第二项适用于建立第二个或更多虚拟机, 即使用已经建立好的虚拟机磁盘, 这样可以减少虚拟机占用的真实磁盘空间。第三项则允许虚拟机直接读写磁盘空间, 比较危险, 所以适合熟悉使用磁盘的高级用户, 如果操作失误会把真实磁盘里的内容删掉的。

3.2 设置虚拟机磁盘容量。第一项可以定义磁盘大小。第二项允许虚拟机无限使用磁盘空间, 但需要真实磁盘足够大。第三项则限制了每块虚拟磁盘的最大容量为 2G。

3.3 这一步是最后一步了, 虚拟磁盘即将被创建, advanced 里可以更改虚拟磁盘的接口是 SCSI 或是 IDE

4 虚拟机共享上网

第一步: 点击 VMware 菜单“虚拟→设置”, 再点选网卡 NIC 并设置它的属性, 可在“网桥”、“NAT”、“仅是主机”中任选一项, 但一定要记住所选的项目。 第二步: 把宿主电脑网络连接的 IP 地址设为自动获取。 第三步: 在宿主电脑“网络和拨号连接”窗口中, 右击 ADSL 虚拟拨号连接, 选择“属性”命令, 在“共享”标签页内选中“启用此连接的 Internet 连接共享”, 然后根据虚拟机网卡的工作方式选择一个网络连接(见图 6), 具体对应如下:



网桥—本地连接

NAT—VMware Network Adapter VMnet8 只是主机—VMware Network Adapter Vmnet1 第四步: 宿主电脑启用网络连接后, 相应网络连接的 IP 地址被自动设置为 192.168.0.1。因此, 需要把虚拟机的 IP 地址设置为 192.168.0.X(X 取值范围为 2~254), 同时要把 DNS 服务器地址、默认网关都设置为 192.168.0.1。这样, 宿主电脑上网后, 虚拟机也就可以通过宿主计算机共享上网了。

实验十四 Hadoop 的安装与部署

【实验目的】

- (1) 理解 Hadoop 的工作原理与工作模式。
- (2) 掌握基于 Linux 平台的 JDK 6.0、Hadoop 的安装与配置；
- (3) 掌握 Hadoop 软件的单机工作模式的配置与使用方法。

【实验内容和步骤】

1 安装 JDK

到 sun 网站下载 JDK 安装包 jdk-6u11-linux-i586.bin, 复制到机器的 usr 目录中, 并在每台机器的 root 用户下面安装.

在 root 用户下:

```
$ cd /usr
```

```
$ chmod +x jdk-6u11-linux-i586.bin
```

 给安装文件增加执行权限.

```
$ ./jdk-6u11-linux-i586.bin
```

, 按提示按几个空格健后, 输入 yes 后开始安装 jdk6.

安装好后, 将目录名修改为 jdk6.

设置 JDK 的环境变量, 考虑到 JDK 可能会有其他系统用户也会用到, 建议将环境变量直接设置在 /etc/profile 中具体内容(如果没有则直接在 profile 文件中添加):

```
export JAVA_HOME=/usr/jdk6
```

```
export CLASSPATH=$CLASSPATH:$JAVA_HOME/lib:$JAVA_HOME/jre/lib
```

```
export PATH=$JAVA_HOME/bin:$JAVA_HOME/jre/bin:$PATH:$HOME/bin
```

 用

```
$ source /etc/profile
```

 使用 java 环境生效.

2 设置目录并安装 Hadoop

用 hadoop 用户登录 namenode, 并新建一个目录, 用于存放所有 hadoop 相关内容。本例中在 /home/hadoop 目录下新建 HadoopInstall

下载 hadoop 安装包并 copy 至 namenode 的 hadoop 用户的 /home/hadoop/HadoopInstall 并解压缩:

```
tar zxvf hadoop-0.16.3.tar.gz
```

 考虑到今后升级以及其他操作的方便性, 建议建一个名称为 hadoop 的链接, 指向 hadoop-0.16.3 目录:

ln -s hadoop-0.16.3 hadoop 新建目录:

/home/hadoop/HadoopInstall/hadoop-conf 将

/home/hadoop/HadoopInstall/hadoop/conf

目录下的 hadoop_site.xml,slaves,hadoop_env.sh,masters 文件拷贝到

/home/hadoop/HadoopInstall/hadoop-conf 目录 在/home/hadoop/.bashrc 文件中设

置环境变量 \$HADOOP_CONF_DIR:

```
export HADOOP_CONF_DIR=$HOME/HadoopInstall/hadoop-conf/
```

3 单机工作模式

单机模式下 Hadoop 使用的是本地文件系统, Hadoop 中有几个示例程序并且已经打包成了 hadoop-0.20.1-examples.jar。其中有一个 WordCount 程序, 功能是统计一批文本文件中各个单词出现的次数

```
$ cd /usr/hadoop/hadoop-0.20.1 $ mkdir test-in
```

```
$ cd test-in
```

```
$ echo "hello world bye world" >file1.txt
```

```
$ echo "hello hadoop goodbye hadoop" >file2.txt
```

#在 test-in 目录下创建两个文本文件, WordCount 程序将统计其中各个单词出现次数

```
$ cd ..
```

```
$ bin/hadoop jar hadoop-0.20.1-examples.jar wordcount test-in test-out
```

注意事项: test-out 目录是程序生成的,运行前必须先删除 test-out 目录 #执行完毕, 下面查看执行结果:

```
$ cd test-out
```

```
$ cat part-00000
```

```
bye 1
```

```
goodbye 1
```

```
hadoop 2
```

```
hello 2
```

```
world 2
```

实验十五 Windows Azure 云平台搭建和部署云平台服务

实验目的

1. 通过微软公司提供的验证码激活账号，登录微软公司的 Windows Azure 云计算平台；
2. 把 Windows Azure 开发环境安装好，为以后的实验作准备；
3. 在 Windows Azure 下开发项目并且发布；

实验设备

1. 安装 Windows 7 Professional Edition or higher 的计算机，推荐用个人的电脑；
2. 稳定高速的 High Speed Internet;

预习要求：

1. 认真预习本实验的要求与实验任务，做好准备。
2. 认真复习第一章和第二章云计算的基本知识；
3. 认真学习课本第五章有关 Windows Azure 云计算平台的基本知识；
4. 要求在做实验之前就对实验的任务和步骤比较清楚；

实验任务

1. 通过微软公司提供的验证码激活账号，登录微软公司的 Windows Azure 云计算平台。然后要按照以下办法在 Windows Azure 下面创建网站并且发布到 Windows Azure 云平台上；
2. 利用提供的 Windows Azure Training Kit 的目录：L1 Cloud Introduction，创建一个 Windows Azure Web Site 网站，然后用 FTP 客户得到一个运行在云中的 ASP Legacy page 网页，记录下 Internet 上能够访问它的网络链接 web link；

开放式课题（相对需求不是很固定，可以自由发挥，至少 1 题）

以下多个开放式课题，可以根据兴趣选择一个，由 1—3 人组成的小组一起完成。

课题一、实现一个 WEB 信息采集程序/系统

功能要求：基于 HTMLParser 等网络采集即时采集 WEB 网站的非结构数据（图片、文件等），可由用户设置采集数据的目标链接和数据源（例如采集 <http://g4c.laho.gov.cn/> 网页中的新建商品房网签数据，见网页中间的表格，数据附属在图片上），且可以由用户设置采集到的文件的命名规则、查看采集文件的下载进度等功能。

采集数据配置的参考界面：



文件采集下载的参考界面：



课题二、实现一个文件同步程序/系统

功能要求：使用 socket 编程实现对设置数据源的非结构数据（图片、文件等）同步功能和传输功能，即，做得比较好的话，还可以设置同步的周期，比如可以设置同步周期为即时同步（每隔 10 秒）、间隔多少小时、间隔多少天等。

设置同步源文件夹和目标文件的参考界面：

数据资源名：	测试文件同步 *	数据资源名：	目标文件目录 *
数据资源类型：	文件	数据资源类型：	文件
数据资源所属IP：	127.0.0.1 *	数据资源所属IP：	127.0.0.1 *
端口号：	1288 *	端口号：	1288 *
路径：	C:\wmpub *	路径：	E:\tongbu *

同步的方式参考（默认为即时同步，即每隔间隔 10 秒左右同步一次）：

调度方式：手动同步

- 手动同步
- 计划同步
- 周历定时
- 周期同步-小时
- 周期同步-天
- 周期同步-月
- 周期同步-每月第一个星期一
- 周期同步-每月第x天

课题三、实现一个分布式主机监控系统

实验目的及要求：

该实验内容为通过 socket 编程技术和 WEB 应用开发技术实现一个分布式主机监控系统，通过服务器监控各客户主机的状态（包括系统启动、系统关机、IE 启动、IE 关闭），

并通过 WEB 网页展示主机监控功能。主机监控的功能包括监控各主机的状态、展示主机的结构关系等，系统功能界面参考如下：

终端地区分布结构

展开全部 | 收起全部

终端地区分布树

- a
- b
- c
- e
- f

主机监控系统

终端IP	终端类型	终端地址	终端状态	机器类型	CPU信息	内存信息	服务器	关机	重启	关闭浏览器	重启浏览器
127.0.0.1	酒店	潮汕	正常开机	lenvon	i9	4GB	a	关机	重启	关闭浏览器	重启浏览器
192.168.1.102	酒店	广州	正常开机	lenvon	i5	4GB	a	关机	重启	关闭浏览器	重启浏览器
125.216.21.49	4s店	广州	正常开机	1	1	1	b	关机	重启	关闭浏览器	重启浏览器
123.12.2.12	123.12.23.12	123.12.23.12	正常开机	1	1	1	c	关机	重启	关闭浏览器	重启浏览器

[首页](#)
[上一页](#)
[下一页](#)
[尾页](#)
 页次:1 共:1页
 [跳转到](#) [页](#) [确定](#)

[返回首页](#)
[服务器树维护](#)

实验原理和步骤： 1. 实验原理

技术路线：socket+jsp+servlet，服务器与客户端使用 socket 通讯，服务器通过 socket 通讯获取客户端状态（包括系统启动、系统关机、IE 启动、IE 关闭），服务器监控类基于 servlet 实现；服务器上的展示程序基于 WEB 实现；客户端基本信息基于 XML 存储和维护。

2. 实现步骤:

(1) 开发 SOCKET 通讯监控程序，并封装成 SERVLET； (2) 搭建 WEB 应用框架，基于 XML 开发客户端主机信息管理功能（设计客户端信息 XML 描述文件，实现客户端信息的增删改功能）；

(3) 开发主机监控系统的 WEB 网页，实现监控功能； (4) 部署系统； (5) 测试系统。

课题四、网络爬虫

实验目标:

本次实验目标为设计一个分布式网络爬虫实现一下功能:

1. 从一个给定的网址中分析其所包含的 URL 并爬取对应的网页，直到爬取完全部不重复的网页为止。

2. 支持分布式爬取，同时记录输出每一个网页的大小。

3. 采用多线程结构设计，实现高性能的网络爬虫。

随着国际互联网的迅速发展，网上的信息越来越多，全球网页数量超过 20 亿，每天新增加 730 万网页。要在如此浩瀚的信息海洋里寻找信息，就像“大海捞针”一样困难。在实际生活中我们经常会使用像百度、Google 这些搜索引擎检索各种信息，搜索引擎正是为了解决这个问题而出现的，而网络爬虫正是搜索引擎所需要的关键部分。本次实验主要的内容就是利用 IO 复用抓取网页，并多线程的分析每个抓取到的网页所包含的 URL 信息，通过消息队列将抓取网页的部分和分析网页部分进行通信，最终记录下 160000 网页中所包含的所有 URL，实现分布式网络爬虫。

