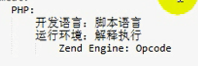
Java技术

\*\*\*php是一种开发语言 平时安装的php也是运行环境

Php开发出来的网站程序怎么运行？ Php是一种脚本语言 也是一种动态语言 开发出来 直接解释器解释执行 ----- 但是 大量用户的时候 每一个都要解释执行 会非常慢 ----- 在php4的时候 两个以色列学员为php引入了zend引擎 ----- 两段式 第一步编译成opcode 而后执行



Apache每一个进程响应一个用户的响应 ----- 每一个进程有自己的进程空间 ----- 这样 每一个用户编译的opcode无法被公用 ---- 这时候 就引入了Xcache和APC eAccelarator ------生产环境中 用的很多的是XCache

Php实现mvc ----- 实现大规模的应用程序的时候 应该分成三成：数据层 业务层 和 表示层

---- 这个和TCP IP一样 分层 就是 每一层进行修改 不会妨碍到别的层的

就是下层进行封装 向上层提供接口 上层直接调用接口

----- 这样 招人的时候 负责数据层 负责业务层 和最后负责展示 ----- 精细化



Php要达到这种功能 要借助很多框架 比如smarty ---- 这样的框架可以方便讲数据 业务 展示进行分层

这种适合开发web应用 ----- 浏览器中看到的是展示层

----- 门户站点 有很多模块

数据库查询 要展示多少条 要进行过滤 ----- 通过网页展示给用户 多大的字体 什么颜色

通常展示给用户的都是静态的资源 html -- css – javascript



由此 使用C语言开发一个网站 和PHP相比 有什么坏处？

为C引入一个模板 让数据 业务 展示分开 但是 后端数据 业务分开非常难

并且C和C++移植性差 --- 和本地CPU OS结合太紧密

----- 最大缺陷就是一直困难

----- 虽然比汇编移植性好 但是 对于互联网不合适

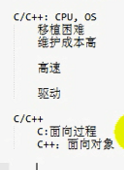
----- 维护非常困难 发现某一个字符写错了 都要重新编译

C有一个致命的优势 ------- 比汇编差很多 但是 比Java什么的 高30% -50%

所以 C 适合开发底层os 数据库

现在大型系统 多语言互相操作

C比C++简单很多



C programming language ----- 100 – 200页 足以介绍

但是 C++ primer 洋洋洒洒写1000页 也不说明问题

Linux内核 就是C写的

但是无论如何 C和C++写出来的程序

OS为高级语言提供API ----- application programming interface ----应用编程接口

基本意义上 就是把系统调用 system call 封装起来 ---- 通过接口提供出来 ----- 便于程序员调用

OS底层把底层不同的硬件通过软件层做了一次抽象 ---- 把抽象的结果 表现成程序员容易接受的System call提供给程序员 ----- 这样程序员就能够编程了

System call 过于底层 简陋 ---- 但是 这样 system call的数目很少

由于system call简陋 所以有了二次封装 --- API

并非所有的API都有system call

如果OS不一样 有些事windows 有些事linux

所以 windows的c API拿到linux上面编译 就是不OK

编译中间最重要的就是链接过程 ----- 就是检查我们开发过程中依赖的库是不是存在的 –调用的库遵循的语法是否相同 ----- 所以 使用Windows API开发的程序 很难到linux上面运行



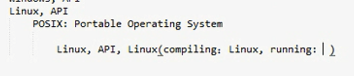
所以 就有了POSIX ---- POSIX ----- portable operating system**可移植操作系统**POS 只不过为了兼容UNIX的脚法 所以 POS -> POSIX ----- 是**一种操作系统编程接口规范**

只要双方都遵循POSIX规范 ---- 这样 在windows上面开发的程序---- 就可以在linux上面编译就可以通过了 ---- 因为编程接口一样 【注意 只是说 我再windows上面开发完的源代码 可以拿到linux上面编译通过】

所以规范是非常重要的 ----- 规范是实现互操作的重要前提 -------- 所以 就有那么多标准出现了

只要双方都遵循posix规范 那么**跨平台的编译**就称为现实了【**说的是跨平台编译 没有说跨平台运行！！！！**】

我们在linux上面使用linux api上开发的程序 编译完成 -🡪 实现到windows上面运行 可不可以？

可以否？

刚才说的是 在linux上面 开发的 都遵循posix 到windows上面可以完成编译 ------ 但是 不能运行！为什么？

尽管说大家都遵循同一个编程接口规范 ----但是 windows 的是 dll 动态链接库 但是 linux是so 共享对象 在windows上运行是依赖于dll的 那么到linux上是无法找到这个dll的 ----- 更重要的一点是 二进制程序的格式都不一样

Linux上面的二进制 通常告诉你是  ELF格式 这就是可执行程序的格式 windows上面不一样

尽管大家的API都一样 但是 ABI（Application Binary Interface）---- 应用二进制接口是不一样的 ----- 尽管可以跨平台编译 但是 不能跨平台运行

除非 把不同平台的ABI磨平 才可能实现跨平台运行

---- 这个就是Java

Java开始叫OAK ---- 当时的Sun公司开发智能电视 ----- 要求开发出来的程序在各种平台上面都可以运行 ----- 而这个程序的作者 当时窗外的就有一个棵高大的橡树 所以叫oak

Oak之前 还叫green

Green --- oak ----

后来大家觉得这个语言好玩 怎么优化 都在一起喝咖啡 这个咖啡就是印尼的爪哇岛生产的 所以 green --- oak –java

Java实现的最初目的就是为了在不同硬件上运行同一个编译好的程序的 ----- 所以 Java的**基本的着眼点** **就是 跨平台性**

不同的os 就算api兼容 但是 abi也不兼容

---- sun公司当时的智能电视太超前了 90年代初

所以Java当时在智能电视上 没有更多的作为

但是 随着http协议的出现 互联网爆炸式发展 ---- web应用 ------ 当时特别需求一种跨平台的语言 ------ 为什么呢？就是 你的页面是html 静态网页 ----- **所以 要开发动态网页 要求你的程序到任何一种客户端 都能够运行** ---- 很可能用户使用的是不同的OS ---- 就算是同一种OS OS所在的硬件也未必一样的 ----- 那就必须跨平台 ---- Java就在这种场景下 无心插柳柳成荫

--------- 此后不久 Java就广泛用到**开发动态小程序上**

-------- Java 包含独立但是又相关的技术

(1). Java的程序设计语言 ---- 本身是一种开发语言 ---- 要编译好才能运行 ----- Java早起跨平台 所以 不能编译成abi 早起也是编译成opcode --- Java这里叫字节码 byte code

这个字节码 不是能够直接在os运行的二进制 ---- 必须用自己的解释器解释执行 ---- 对于Java来讲 这个解释器不叫解释器 而叫做 jvm Java虚拟机

(2). JVM Java虚拟机

Java编译源程序之后 有了第二种格式 字节码格式 ---- Java class --- java类格式

Java字节码格式遵循哪种规范 就是Java字节码规范定义的

(3). Java class 字节码

Java class需要运行在Jvm上

(4). Java API

既然Java是一种程序设计语言 为了加速开发 Java各方提供了许许多多的库 这个就是Java应用程序编程接口(同样 c语言有自己的c库) ==== java api

所以 是以上四种技术 Java程序设计语言 Java字节码 Java虚拟机 和 Java API



Java api + java程序设计语言 进行Java源程序 开发 ---- 编译成Java class ---- 最后在jvm上面运行

很多时候 大家都提Java ---- Java有一个概念 上面四个东西未必是完全一致的 --- -要有概念

字节码展示的文件是Java类文件的格式

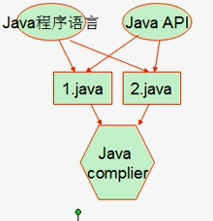
由此 Java 程序在执行的时候 流程编程这样

首先是Java源程序文件 以.java结尾 ----- 依赖java api和程序语言进行开发

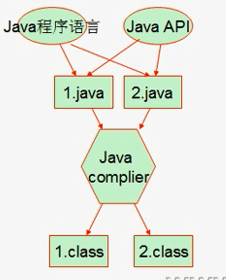
这些原程序要执行 就要依赖编译器

然后 源文件需要javac这个编译器 进行编译 生成字节码文件

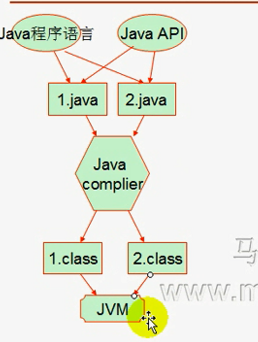
编译完成之后 就要有Java class文件



这样 就生成了1.class和2.class



这些类必须在Java虚拟机上面运行



在linux开发了C程序 要依赖共享库 才能运行起来 否则无法运行

1.class依赖对应的二进制格式才能跑起来 也就是api本身的库才能运行

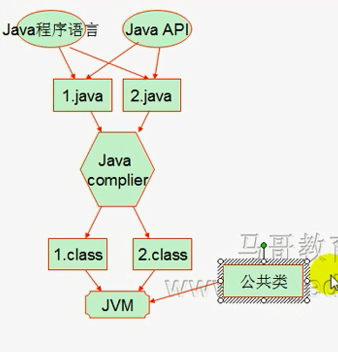
C库开发的时候 需要找头文件 函数名称 参数的格式 参数类型 调用方法 返回值的类型

但是 运行的时候 是要使用库中的某一个函数 ----- 具体是那一个函数？这个函数名叫什么？

比如 shell脚本编程中 source了另外一个文件 表示 把这个文件中的变量引入到当前脚本运行环境中 ----- 当前的脚本就可以用了

脚本本来就是源码 ---- 对于你给的二进制程序要运行得时候 就要依赖对应的库 --- 这个库此时必须在内存中有 ---- 否则无法运行

----- 对于字节码文件 依赖的其他字节码 必须首先装载到内存中（对应的脚本 就意味着 这个要source一下 source过来的过程） ----- 这个叫公共类库



除了公共类以外 还依赖一些私有类

写shell的时候 ---- 库中没有合适的函数 就是私有函数

这些类都要装载到jvm中

Jvm可以理解为运行某个程序的运行环境 ------

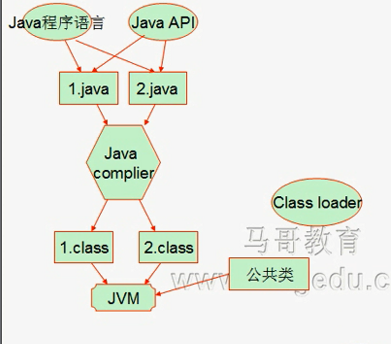
Jvm不能保证都运行起来 ----- 因为无论是公共类 还是私有类 都要装载进去 ---- 自己负责加载

就是 jvm提供一个运行的平台 至于你依赖什么库 OS不管 要你自己管

1.class要明白依赖哪些库 ----- 自己手动都要加载进来 ---- 如果应用程序完成加载这些类 就非常麻烦了

Linux有一个独特的程序 依赖于其他库 可以有一个依赖的组件 把这些依赖的库全部都调用进来 而不用程序自身完成依赖的调用

同样道理 ---- jvm还有一个类加载器（Java中的库就是类 所以 叫类加载器） ---classloader

---- 由此 1.class 2.class就能运行起来了

需要注意 不止一次的讲到 最终代码都放到CPU上面运行的 ------ 这些都要有OS的内核来调度和理解 ------ 想想一下 os本身必须能够理解class ------

也就是真正跑起来 --- 还是依赖于操作系统

Java的最大特点就是 Once for all 一次编译 到处运行

就是因为jvm的存在

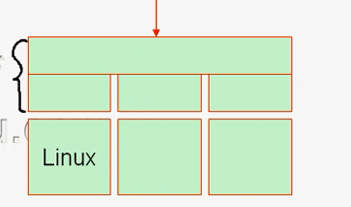
所以 jvm要磨合底层OS的不同

Jvm本身有两段 ---- 一段是特定操作系统相关的 另一段和os无关 ----- 最后 把和os无关的代码转化为和os相关的代码进行执行

所以jvm就是通过这种机制 每一种平台都是一段独有的代码 最上面输出都是和os没有关的代码

Jvm最终还是要将程序转化为和os相关的才能运行

正是不同版本的jvm的存在 才能使得java可以一次编译 到处执行



====== jvm的实现方式 ======

Jvm实现方式有三种

(1). 一次性解释器解释字节码并执行 ----- 类似于shell 解释器 ----- 解释的是字节码 ---- 这个字节码是解释之前已经编译好的

.java -> .class  
 .class本身就是字节码 ---- .class本身不是纯粹的二进制文件

第一次解释并执行完 第二次还要解释 解释的过程 就是把字节码转换成和OS相关的二进制程序

所以 字节码需要jvm转换成和平台相关的代码

每一次执行 都要解释一遍 性能不好

(2). 即时编译器 just intime compiler

每一次 每一个class类 解释成二进制格式 就缓存起来 下一次再用 直接从缓存中获取

**编译器要依赖于更多内存缓存解释的结果**

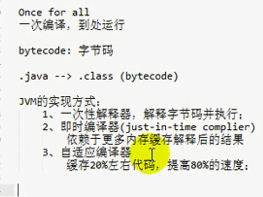
如果程序非常大 解释的都缓存 ----- 这样 非常耗费内存

所以 都缓存在内存里面 会导致占据大量内存

(3). 自适应编译器

可以监控所有的类中 哪些代码执行频率高 哪些执行低 把执行频率高的 缓存起来 ---- 依据28法则 80%的活是20%的人完成

自适应编译器 只去缓存20%的代码 --- 提高80%的速度



这是jvm的实现方式

Java有四个独立的部分组成

程序设计语言是程序的作者研发的

Java api ----- 这个按道理也是sun定义 ---- sun定义的规范 可能收费

Java api 有两类 --- 一类是sun定义的 另一类开放组织定义的api

sun自己定义的api一年就完蛋了

------ 以上是java api

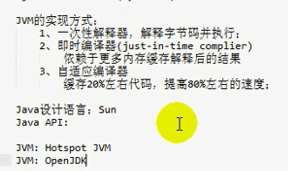
Java的类文件格式 ---- sun自己定义的 ----编译之后 遵循哪种规范

Jvm虚拟机谁提供的？----- jvm是实实在在的 jvm的实现有差异性

Jvm很多种

Sun自己的叫hotspot jvm

开源界有一个jvm 叫openjdk



Sun的Hotspot ------ 软件的名称叫jre

Jre ----- java running environment Java运行时环境

Jdk ----- java development toolkit Java开发套件

Jdk就是jre的超集

Jre仅仅能让你运行 但是 jdk既能让你开发 又能让你运行

如果是终端用户 直接安装jre就好了

早期jdk是收费的 jre是免费的 现在都是免费的

开发过程中 编译器最重要的

Openjdk就是包含了开发+运行的开源实现



到今天为止 java是全球最流行的语言 非常成功 现在为止 Java 自己api以外 许许多多第三方组织提供了类库

Jdk = 语言 + api +jvm

Jdk用于实现开发Java程序的最小环境

但是 为了后面描述方便 ----- jdk不仅仅包含这三个组件

Jre仅仅包含jvm和api ----还是api的子集



除了Java API jvm以外 还提供了调试工具 和 监控工具

编译器 打包工具（压缩格式）

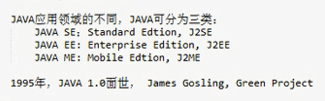
目前Java是跨平台的 所以 Java特别受欢迎 Java根据领域不同 Java技术分成3类

---- j2se j2ee j2me

三种技术中j2me还是没有怎么流行过 企业开发 可以用Android或者objectc

1995年 Java1.0面世 ----- 项目的领导者 james gosling 项目叫green project –> oak -> java

1999年hotspot sun才购买过来



1995年发布的时候 互联网方兴未艾 过几年 人们对互联网的需求 越来越多密切 ---- 有了applet ---- 这个研发的小程序 可以让html开发者 直接提供一个applet小程序 这个 时候 只要客户端在浏览器中安装了jre插件 ----- 这样 就可以在客户端运行程序 并把结果通过网页返回给用户 ====== 由此 动态网站成为了现实

 ---- applet开发的程序要先开发好 然后 把这个小程序嵌入到html中 ------ 然后客户端要把html和applet ----- 都是在客户端 ----- 但这个有安全性不好

因此 这种网站技术 给客户端带来了麻烦 --- 不一定每一个用户都会安装jre 还有 就是网络带宽不大的时候 还要多多下载applet 最后还有安全性问题

-------- 这样CGI就诞生了 ---- common gateway interface

CGI 通用网关接口 ---- 这种能让用户访问某种资源的时候 ---- 能够让web服务器根据这种这个请求 额外触发一段执行这段程序

比如 web站点

Web进程只能服务静态资源 用户请求的不仅仅是文本 还不仅仅是静态的怎么实现？依靠于mime来实现的 -

Mime这种机制 客户端浏览器中支持各种mime插件 有了mime插件 可以将各种http协议将各种mime类型都转换成文本格式 --- 传输过去 对方可以还原

CGI

只需要配置我们的服务器a.cgi ---- 发现请求的是a.cgi ----- 处理器能够以基于cgi这种协议 调用某一个程序 ---- 也就是额外启动一个进程 执行一段代码 把执行的结果返回给web服务器 ---- 再返回给客户端

CGI技术框架下 每一个内容都是由cgi生成的 包括你的静态内容



如果用户请求的是静态内容 我们的web服务器自己响应就行 如果请求的是动态内容 交给cgi协议交给另一段程序执行 ------ 但是cgi对于数据 和 html格式 没有办法区分 所以 无论是静态内容还是动态内容 都要有cgi来生成

由此 html文档都要通过任何一种cgi程序语言echo出来 或者print出来

如果开发一个站点 有很多html代码 ---- 发现html写错了 要改源文件 源文件就要重新编译一次 按道理静态内容 不用编译 但是 还是源代码 所以 任何一点点改动就会导致整个网站的编译 ----- 缓存下来的opcode都会失效

CGI -> MVC

因此 有了基于MVC框架的开发

MVC就要求所有的静态资源不用编译

CGI出现了之后 Java不仅仅支持applet ----- 还支持类似cgi 来开发动态网站----因此 Java又提出了一种规范 Java Servlet

**Servlet说白了就是以Java语言实现的cgi技术**

开发的时候 遵循servlet规范 理解http请求

使用Java代码开发网站 必须遵循Servlet规范 能接受Http请求 理解http协议 能够执行结果以http报文封装起来 返回给客户端

------ 加入了http协议的处理能力 ----- 能够理解http协议 也能理解html规范了

------ 但是 Servlet也是CGI技术

Java所有都要编译成字节码 对应的字节码源文件改变了 就要重新编译 ------

更重要的是 Servlet仍然逃脱不了CGI技术的限定 ---- 也就是任何静态内容还要动态生成 ---- 这样要求Java程序员还要懂得html 这就大大限定了Java的流行 ----- 如果能做到mvc 各司其职 精细分工 ------- php就出现smarty php框架 做到数据和展示的分离

Java该怎么做呢？让Java代码嵌入到html页面中 ----- 另外一种 基于Servlet的Jsp诞生了

Jsp是Servlet上面的特殊类

Java Server Page

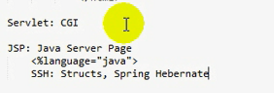
<% %>

可以在html中直接嵌入Java代码 开发Servlet

Jsp仅仅是在Servlet层面上改造的升级的

有了这种技术之后 为实现mvc打下了基础

Java中有这样的框架 struts spring hibernate ---- 简称ssh



Struts就是mvc的框架

学习Java开发 这个ssh是必须会的框架

但是注意 jsp运行的时候 要转换成Servlet 才能运行这个转化不需要程序员的参与 ----- jsp环境就提供了jsp的编译器 叫 Jsper ----- Jsper是jsp的编译器

Jsp尽管摆脱了Servlet束缚 但是 最后还是要回到Servlet

这样 展示和数据就分开了

Java早期 靠 EJB ---- 向前端展示的技术 presentation layer

所以 jsp servlet applet

servlet和applet没有关系

php和jsp功能近视

为什么有php还要jsp ---

----- jsp的运行性能比php好很多

----- java比php规范很多

所以 大型站点 都用的是Jsp

Facebook 直接把php 代码转换成C++ ---- 每一次升级 都重新部署

JSP这种框架 可以让程序员在html中嵌入Java代码 ---- 基于动态网站的开发

基于jsp这种的开发 网站的名称就叫xxxx.jsp

真正的jsp最后都是转换成Servlet

Java类中 applet servlet都是特殊类 都是专门用于开发特定场景中的特定程序的

Jsp就是将Servlet开发简化了 实际上 都是Java的类 ---- 用于让用户加快开发

.jsp -> .java ->



所以 要运行.jsp程序 谁来编译？

需要Java编译器 ----- jsper负责将.java 转成.class

Jdk提供了Java编译器 能够将.java-> .class

但是 javac无法实现.jsp->.java

并且jdk本身没有这样的工具 可以按照需求 把jsp转换成.java ------ 这个按需就是要有一个东西

就算使用基于servlet的cgi技术 开发.java 但是页面 改变了 就要重新编译 ---- 要做到按需 那就是 谁来监控源程序的改变？发现源程序改变之后 谁负责把.java->.class --- 谁来负责和前端的web服务器进行交互？【因为web是web 你的编译什么的 就是本地了】

----- 基于CGI的话 要和web交互

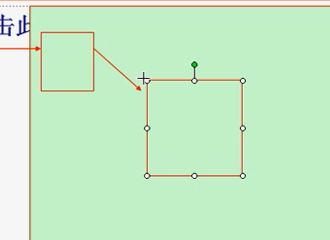
**需要特殊的机制 ----- 专门的应用程序 就是 Servlet-Container ----- 这个Servlet容器就诞生了**

\*\*\*\*\*说白了 就是包含了jdk又包含了jdk不具备但是能让Servlet运行起来的组件 ---Servlet容器【play容器也是这个意思】

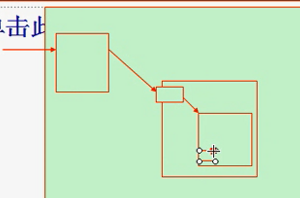
在Servlet容器中 如何工作？

基于前端CGI技术 要能够接受前端web服务器的交互 前端有一个web程序 请求

一旦发现是一个cgi请求 ------ 就执行一个Servlet程序 ---- 为了让Servlet长久的运行 servlet临时启动 就太慢了 ---- 就让这个servlet程序直接运行在一个进程中 -----通过cgi协议和web进程进行交互 ---- 把用户请求内容接受进来 之后 在本地JVM完成应用程序的执行 ---- 最后再和前端的web进行交互 把执行的结果返回给用户



Servlet-Container负责接受用户的cgi请求 负责监视本地的.java程序是否进行了修改 如果修改 ---- 修改之后 负责转换成.class 然后在jvm上面运行

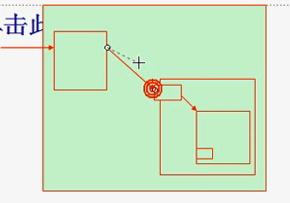


这个只是Servlet程序的执行

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

如果是jsp的执行呢

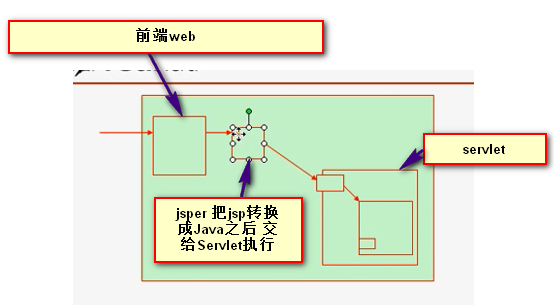
JSP需要一个额外的组件

左上角是web应用 右下角是一个独立的Servlet

首先.jsp->.java 由此 需要一个组件 把jsp文件转换成java

这个组件就叫jsper ----- jsper负责监控jsp源程序的改变 并且负责按照需求对转换之后的Java文件编译成class文件

.java再次交给Servlet执行



总之：Jsp技术就是在Servlet之外 添加了能够以嵌入式写动态的Java页面 并且能够监控页面何时改变 从而进行按需转换的一系列的技术

包含jsper+Servlet的框架叫做web container



虽然有时候 Servlet容器 和Web容器不区分 但是要明白真正的含义

真正http请求 还要提供一个web服务器和用户交互

所以 如果把这个web也纳入进来

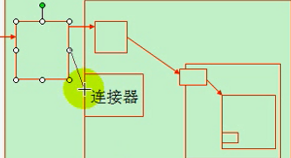
使得这个容器的功能更强大

【所以 注意 什么是web服务器 什么是web容器 web服务器仅仅能提供静态资源】

现在很多web Container都能直接和用户交互 ------ 前端就算没有web服务器来解析http请求 没有web服务器封装http响应 --- 我们的web container自带的组件 也可以完成和用户直接交互

这个时候 站在tomcat的角度讲 ---- 这个叫连接器 Connector

有Connector可以直接和用户交互 也可以和前端web应用程序或者说是进程进行交互



让连接器直接面对客户端和直接和代理相连 有什么区别？

LAMP ---- 承受 压力太小 所以 前端要有一个nginx

维持用户会话都在nginx

当请求的内容在nginx上没有 就由nginx反向代理给后面的Connector

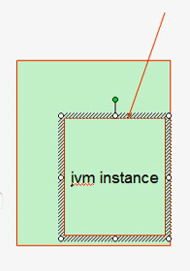
Connector可以直接面对用户 但是 如果Connector的高并发能力不强 这个web container的压力就非常大

所以 前面放一个前置web服务器 用来进行用户会话的创建 释放等压力 ---- 如果前端的web服务器提供了缓存能力 ---- 再和后端的Connector进行联系 ----web container的压力就小了很多 人家不交LAMP 就叫 Java web container 或者有时候叫应用程序服务器

无论怎么样 就是在jvm的外层 增加了jvm的组件 扩展了jvm的功能

无论是web container/servlet container/jdk都是运行在jvm虚拟机内部的环境

无论是servlet container或者web container 真正运行起来之后 在OS上面 都表现为一个JVM进程 --- 意味着在jvm进程的内部 如果是web container 就有web container的功能 如果是Servlet container 就有Servlet container的功能 ------ 这个我们就称为一个jvm instance ---- JVM实例

 jvm instance

如果很多用户同时发起请求 而且前端服务器 转发进来大概100个请求 但是 jvm实例只有一个 怎么并行执行这100个请求呢？

Php如何并行执行？在fastcgi模式下 php每一个请求启动一个进程 完成并发

难道我们Java也要创建多个jvm实力么？---- jvm实例的创建 比 php的代价要大得多 ----- 所以 一般来讲 不会那么做

---- 这就意味着在这个jvm实例内部启动多线程 ----- Java本身就是支持多线程编程的

---- 每一个请求进来 就是一个线程

----- 这么多的线程 都在同一个jvm进程 这使得jvm运行时候的内存变得非常复杂

---------- 现在说一下 jvm运行时的数据区域划分

这个内存空间非常复杂 jvm划分成好几个区域

第一个就是**程序计数器区** ---- 因为java自己就把自己模拟成一个虚拟机 ----- 大家还记得在CPU内部 就有一个程序计数器指针 这个指针 每一个进程 就是程序+数据来执行 执行完这个进程中的一条指令 就载入下一条 ------ 如果进程出现了切换 就需要一个程序计数器 把当前执行到哪一条给记录下来 ---- 程序计数器就是做这件事的 Java本身是在JVM上面运行的 ---- 所以 Java要模拟出来CPU的效果

**每一个线程都需要一个程序计数器 ----- 所以 程序计数器是每线程独有的**

而且每一个jvm在某一时刻 会调度线程运行在某个CPU上 并且这个线程独立保持一个程序计数器 -----

这是程序计数器

第二个是**Java虚拟机栈**

通常 栈用来保存本地变量的 对Java虚拟机的栈 栈是线程私有的 通常用来存储局部变量 方法出口 等等 都是线程私有

每一个线程都由自己独立的空间

下面是线程共享内存区

第一个是**方法区**

面向过程中 只需要理解一个事情的解决过程就可以了

但是 OOP中 将整个处理的程序中的 把每一个产生的任何机制 就像设计关系型数据库表一样 抽象出来E-R关系图

所以每一个e有很多字段组成 每一个e都是一张表 表之间有relation

oop也一样 看看程序中 有多少个可以操作的对象 再把这些对象抽象成类 每一类对象能够执行的动作 叫操作 通常定义为方法来完成

把类实例化的过程 就是对象化

每一种对象 实例化之后 功能不一样 操作不一样 这个操作就叫方法

方法区就是某一类对象的可以执行的方法的集合 ---- 方法是共享的

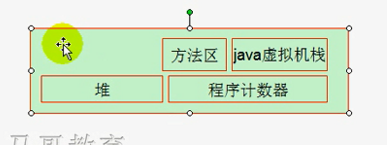
同一类的对象的方法是一样的 但是 方法操作的数据是不一样的

方法对同一类对象 属于同一类方法

第二个就是 堆 ---也是共享内存区

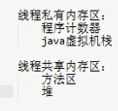
对于jvm中 堆通常是最大的

堆中通常放**对象**



Oop中 所有的文件都是类文件 一执行起来 就会按照需要创建一堆对象 这堆对象就要执行方法 执行方法的过程 就是执行程序的过程 ---程序的执行 就要计数器 ----每一个实例化的程序 都要保存自己的私有变量

所以 堆中的对象非常多 也会非常大



Jvm启动的时候 堆就直接被启动起来

几乎所有的对象实例都在这里面存储

当某一个对象执行结束了 该怎办？

如果对象存储不下了 怎么办？

因为所需要的空间太大 ------ 存不下 就会溢出 ---程序就会崩溃

避免溢出 就需要有人来回收这个空间

Java的高级 重要特性的表现在 ----- 可以实现自动内存回收 由GC完成 Garbage Collector来完成 垃圾回收器

GC就是始终监控这个堆里面的对象

发现老的对象 就进行回收

GC把堆分成了新生代和老年代

对象都是占据内存空间的 未必连续 但是 会产生碎片

所以 GC要基于一定的算法进行回收

Java的程序会运行的好好的 突然间 就会很慢 一忙起来 就会比较慢

这就是Java的坏处

如果碰巧访问量最大的时候 进行了GC 就完蛋了

Jvm运行起来的时候 有很多属性 还有就是GC有很多算法

Jvm启动的时候 要传递许许多多的参数

还有一个就是本地方法栈

Java方法+本地方法

Java真的要和以硬件打交道 必须和os的内核进行打交道 有些特定的功能 要通过系统调用来完成 system call ----- 方法去中的方法 要转化为本地方法 ----- 这样可能会慢一点

所以 干脆 就是 java编程+本地编程 直接和操作系统打交道

---- Java某些方法 可以自身实现 也可以调用外部的方法 ----- 调用外部的方法就叫做本地方法

本地方法栈 ----- 不是每个线程独有的



在某些地方 比如方法区的子组件：比如常量区 常量池 --- 运行时的数据区域 还有MMap ---- 直接内存

就是JDK运行到1.4之后 引入了NIO ---- new input output buffer的机制

利用NIO中的Buffer 可以和OS共享内存 避免数据的复制

这样可以提高Java成性能的机制

上面的这些区域都有可能溢出 做需要关注的就是堆区域

所以 垃圾回收 用的是 GC

什么样的对象认为是垃圾？每一个对象内部维护一个引用计数器

当引用计数器为0 表示可以回收 ---- 不是立刻就回收的 ---- 而是 引用计数器为0 标记为可以回收

隔一段时间 会被GC批量回收

有很多种算法

简单了解一下垃圾回收算法

最常用的一种就是标记清除算法

标记清除的结果就是产生大量的碎片

对标记清除的改进 ---- 复制算法

---- 这样每一个对象都是通过复制的机制创建 回收以后 不会产生碎片 但是 浪费空间

根据大公司的研究 类所创建的对象 98% 大多数对象都是浮游对象 很快就没有生命周期了 ---- 所以复制这种算法就过于浪费

%2用复制算法 剩下都是标记清除

现在所有商用的垃圾回收 都是基于复制算法来实现的

对于那种新生代朝生夕死的对象 使用标记清除 多长久使用的使用复制算法

下一种是标记清理算法

但是对于2%朝生夕死的 上面的呢复制算法 效率就极低了

创建完之后 标记 不用的时候 移动内存另外一端 这个是标记整理 --- 尽可能避免内存碎片

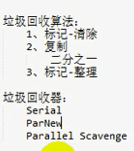
无论哪一种算法 都是把堆中的对象进行分类

------------------------ 上面都是算法 ------ 市面上常见的实现出来的垃圾回收器 有

Serial ---- 历史悠久 完成新生代搜集 ---- 对朝生夕死的回收 ---- 基于标记清除算法 ---- 效率低 --- 一次只能回收一个

ParallelNew --- 实现多线程回收 实现算法细节的调优 --- 要求多核CPU 单核无法发挥优势

Parallel Scavenge ----也是针对新生代搜集的 跟Parallel的类似 但是 可以降低系统停顿时间的 ------ 这样就提高了系统的吞吐量



比如虚拟机运行了5s 垃圾回收1s ---- 垃圾回收占5%

着眼点就是降低垃圾回收的时间比

下面是针对老年代的回收算法

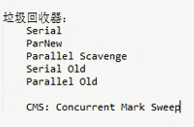


Serial是单线程的 Parallel是多线程的

多线程的模型就好很多

后面把Parallel Scavenge 和Parallel Old柔和起来 就是CMS

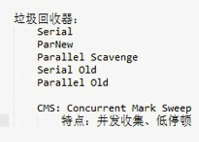
并行标记清除 --- -使用的是标记清除算法



CMS适合于互联网 并行回收+ 降低垃圾回收占用的jvm运行的整体时间

---- 主要目的是吞吐量有限

主要实现并发搜集 + 低停顿



CMS有一个缺点 就是无法清除浮动垃圾

浮动垃圾：某个线程创建过程中 产生的垃圾不能被回收

100个Java线程 只有4个核 所以 某一个时刻 96个线程的垃圾可以被搜集 但是这4个线程的垃圾不能回收

Cms每一次搜集 – 这种机制 还是有可能堆溢出的

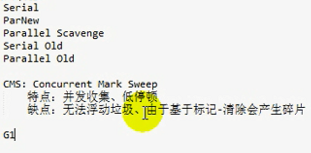
由于基于标记清除算法 所以可能会产生碎片



当前 有一个G1 没有大规模应用

G1 1.7中已经使用了

G1不会产生碎片 并且非常精确控制停顿 使得GC可以更多被参与



----- jvm虚拟机运行的过程 都由可能堆移除 要随时监控jvm堆的状况

这些都在jvm中

Sun公司 jre和jdk提供了jvm

除了sun 还有一个openjdk

除了这些 很多其他公司 像ibm 惠普 也有自己的jdk

很多商业公司都由自己的jdk

---- 在一定程度是兼容的

Sun的jdk也是开源的

如果商用的话 要自己看 哪些是开源的 哪些是闭源

Openjdk bug很多 所以 openjdk很少应用在商用中 但是 taobao用的就是openjdk 以为淘宝做了很多个性化定制

所以经过淘宝改进了openjdk 改进了很多bug 提升了很多性能 ---- 可以到淘宝上去看看

为了使用更成熟的技术 建议使用sun的jdk ---- 到oracle的站点上

目前维护两个分支 1.6 和 1.7[过时了]

每一次更新 Updated1.6 32 ---- 表示 1.6的32次升级

因为我们的jdk可能加 1.6的第32次升级版

Hadoop和Java结合的时候 会指定相应的版本

下面安装一下jdk

有三种 rpm包 通用二进制格式bin格式 和 源码编译安装

 源码编译 使用的都是openjdk了

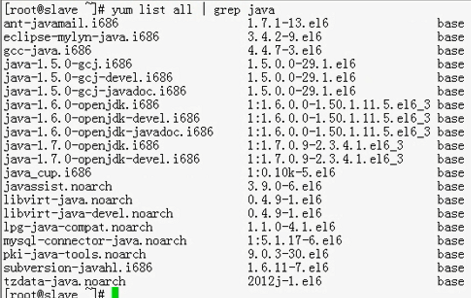
Java的源码编译安装是有难度的

Sun的jdk不提供源码编译

老师没有提供手动编译

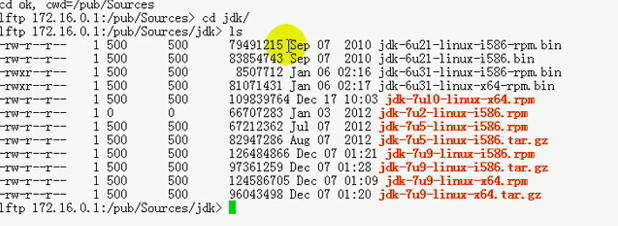
安装一个jdk看下里面有什么



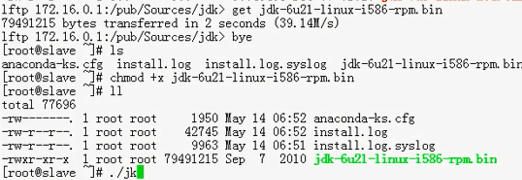


这个是操作系统自身提供的

如果下载sun的

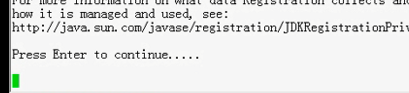












安装到哪里去了？

/usr







默认安装到/usr/java



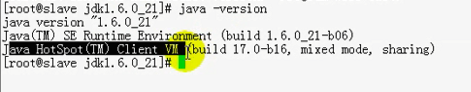


------ 配置环境变量 PATH



Source一下





这样Java配置完成

再看下 实现启动一个jvm虚拟机



虚拟机启动时候的命令 来配置虚拟机参数

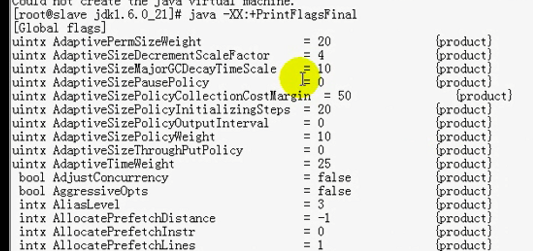
对虚拟机来讲

+表示开启此参数的功能

-表示关闭这个功能



-XX:+PrintFlagsFinal能够显示所有参数 及其默认值



只要是布尔型的 默认使用false 你就使用+来进行开启

如果是true的 你想关闭 就使用 –来进行关闭

如果非boolean类型的 那么就要使用进行赋值

这是配置参数

这个是设置系统属性值的 ----- 使用-D指定某个属性为什么

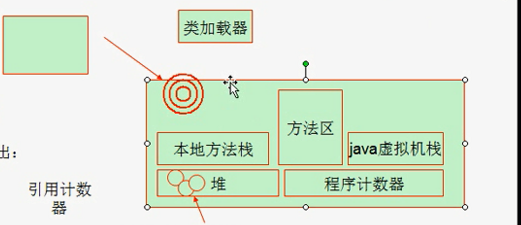


JVM启动起来之后 有那么多的数据区域

任何一个类都要加载到内存中 因为某一个程序的运行 要依赖别的类文件 ----- 所以 需要一个类加载器 帮用户加载类 ---- 这个机制是额外的

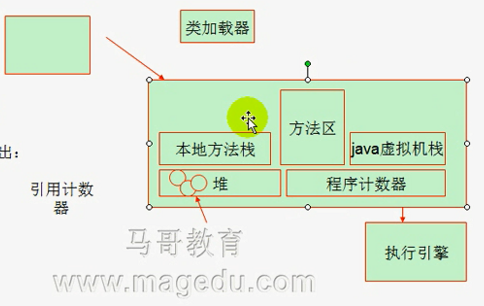
类加载器可以使用jvm自带的cl --- 也可以自定义类加载器

这恰恰这么受欢迎的原因



一旦装载成功 分配一个线程 ---- 轮到他执行 就可以运行

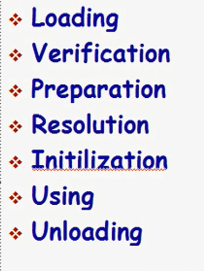
但是 要运行 ---- 要依靠jvm的执行引擎完成



类加载器的过程比较独特：类从被加载到虚拟机内存中 到被清理出去 生命周期包括：

类的生命周期

1. 加载 Loading
2. 验证 Verification 安全相关
3. 准备 Preparation
4. 解析 Resolution MySQL的Parse类似
5. 初始化
6. 使用
7. 卸载



New就完成加载 加载由类加载器完成

为了确保class的字节流符合虚拟机的规范 不能什么都拿来执行 所以 Verification 要有验证这一步 --- 文件格式验证 元数据验证等等

Preparation ---- 准备 比如 初始化静态变量 准备创建线程等

Resolution ---- 直接引用 间接引用

**Initialization ---- 初始化 类加载的最后一步 加载完成 就可以使用这个类了**

所以 类加载器很强大

步骤上面看 大部分都是类加载完成的

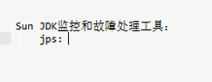
到底使用什么类加载器 可以在启动虚拟机的时候 使用-classpath来指定类路径 这里会有类加载器



这里面是jvm

Jvm运行的时候 有很多内存空间可用

Sun jdk 免费提供监控和故障处理工具

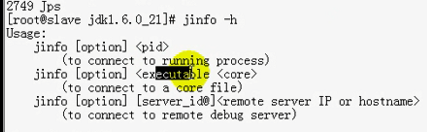
 jvm process status tool ---- jps 显示指定系统内的hotspot进程 这个命令在后面使用Hadoop的时候很多

Jstat java statistics ---- Java统计监控数据工具 搜集并显示hot spot各方面运行数据



Jinfo ---- 显示hotspot虚拟机的配置信息的

显示正在运行的虚拟机 jinfo后面通常跟一个进程号



还可以使用-flag来禁用或者启用某个标志





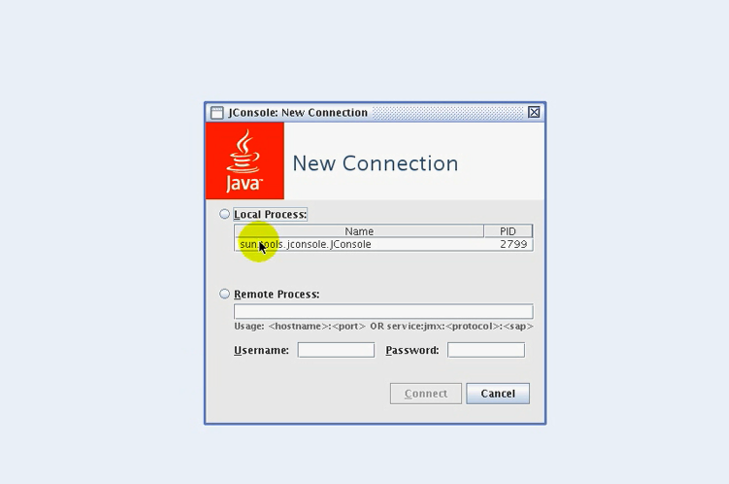
下一个是jmap

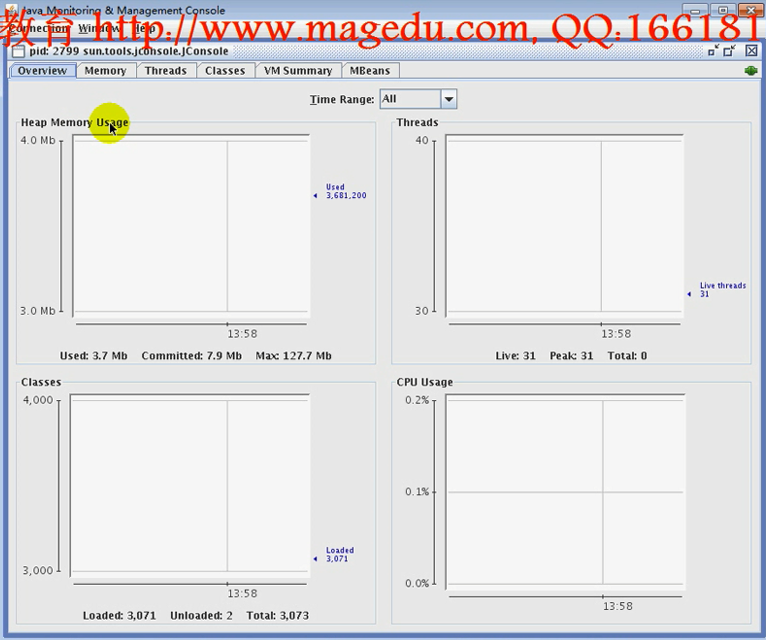
生成某个hotspot的内存转储快照 ---- 调试的时候 会用到

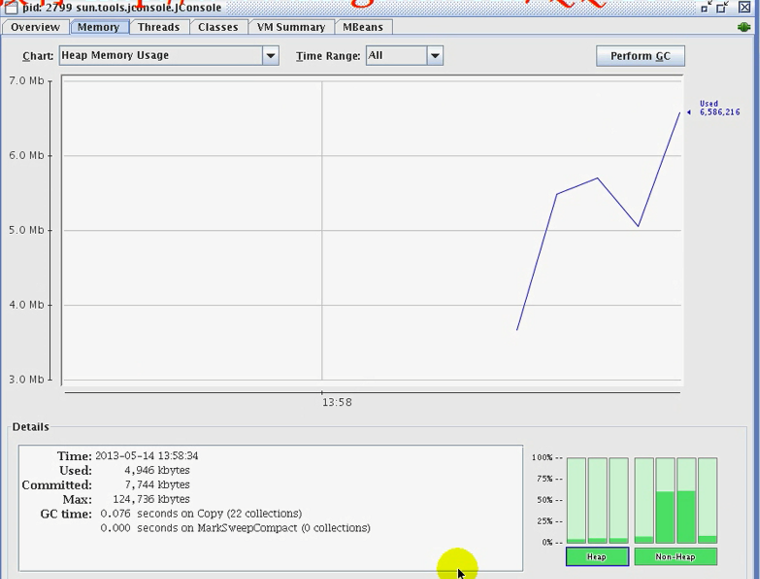


还有几个可视化工具 一个是 jconsle ----Java的监视与管理控制台

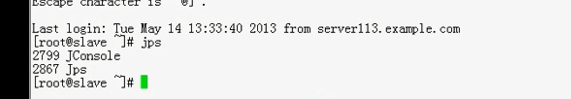
启动一下

可以链接到本地或者远程的虚拟机

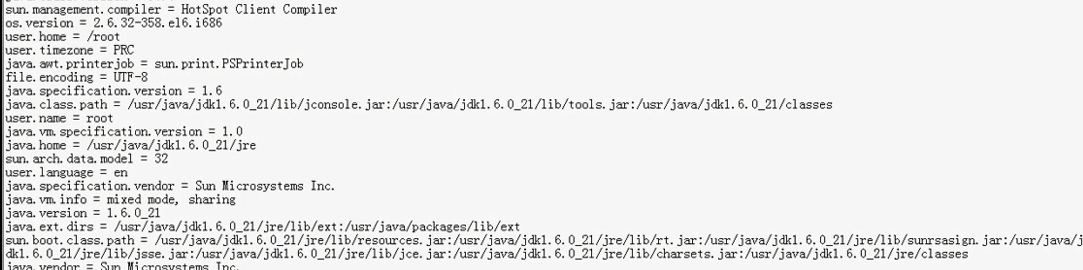


 ---- 非堆内存

这个时候 使用jps 就可以显示你的Java线程



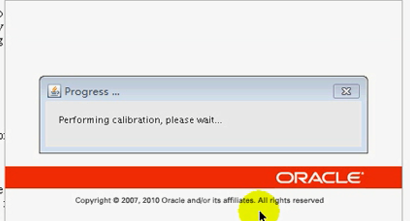
Jinfo 2799

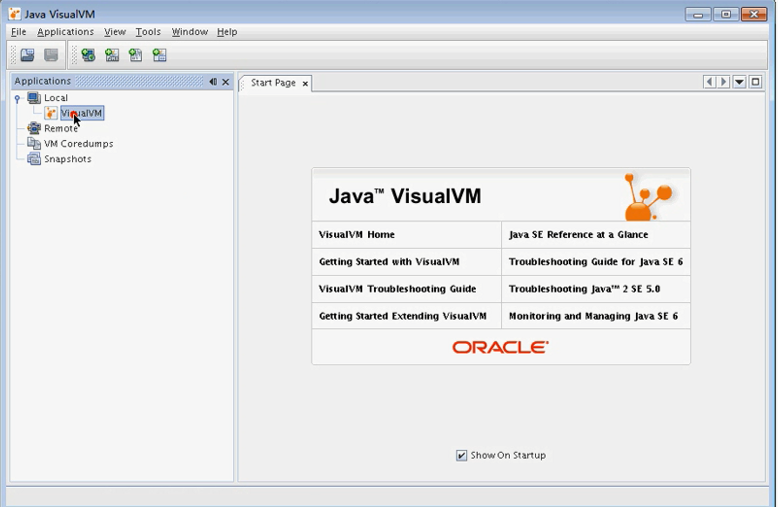


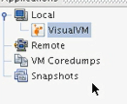
这个是jconsole

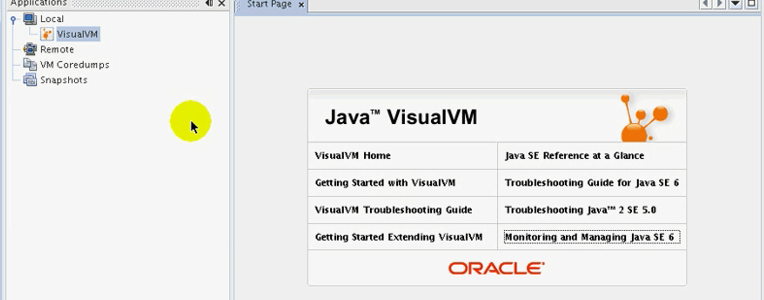
还有一个工具 叫virtual vm

也是可视化工具





各种管理都可以在这个页面上面进行



以上就是常用工具

还有很多商业公司 IBM eclipse 等等