

H3C CAS 3.0 REST API 接口使用操作指导书

目 录

1 简介.....	1
2 配置前提.....	1
3 配置环境.....	1
3.1 服务器.....	1
3.2 软件	2
4 组网拓扑.....	2
5 操作指导.....	3
5.1 搭建开发环境.....	3
5.1.1 创建虚拟机	3
5.1.2 安装虚拟机	3
5.1.3 安装 Java SE 开发工具包	5
5.1.4 配置 Java 环境变量	6
5.1.5 安装 Tomcat	9
5.1.6 安装 Eclipse 集成开发环境	10
5.1.7 集成 Eclipse 和 Tomcat	10
5.2 创建开发工程.....	13
5.2.1 下载 HTTP 客户端编程工具包	13
5.2.2 创建 Java 工程	13
5.2.3 创建 Java 类	16
5.3 调用 H3C CAS 3.0 REST API 接口	18
5.3.1 查询虚拟机性能监控数据	19
5.3.2 JSP 调用获取所有虚拟机列表	21
5.3.3 单点登录.....	29

1 简介

H3C CAS 3.0 云计算管理平台定位为一个 IaaS（Infrastructure as a Service，基础架构即服务）层面的基础管理平台，提供数据中心物理设施和虚拟资源的集中化统一管理功能，而云计算作为目前 IT 行业最火的一个概念，在各行各业都有着极大的应用前景，但是，不同的行业对云计算的需求不一样，不同的客户对云的理解也不一样，因此，很多客户希望在 H3C CAS 3.0 云计算管理平台基础之上开发自己的专享云平台，底层资源的管理和分配由 H3C CAS 3.0 云计算管理平台完成，而客户自己的云平台只需要关注与自己业务相关的管理，这类客户主要是教育行业、运营商和部分有研发能力的企业。

为了满足客户的需求，H3C CAS 3.0 云计算管理平台在设计之初就被定义为一个开放的 SOA（Service Oriented Architecture，面向服务的架构）平台。在 SOA 架构中，Web Services 是核心。目前，在三种主流的 Web Services 实现方案中，因为 REST（Representational State Transfer，表述性状态转移）模式的 Web Services 与复杂的 SOAP 和 XML-RPC 对比来讲更加简洁，所以，越来越多的 Web 服务开始采用 REST 风格设计和实现。例如，Amazon 提供接近 REST 风格的 Web 服务进行图书查找；Yahoo 提供的 Web 服务也是 REST 风格的；国内的淘宝使用的也是 REST 风格的远程调用。

通过利用 REST 自身的 HATEOAS（Hypermedia As The Engine Of Application State，超媒体驱动的应用状态变更）特性以及 Web Services 富有体系化的结构和易扩展的特点，让 H3C CAS 3.0 的业务支撑能力更加丰富多彩，同时也使 H3C CAS 3.0 具备更强的可扩展性和可定制开发能力。

本文的目的是通过 Java 开发环境构建和示例代码，讲解调用 H3C CAS 3.0 REST API 接口的方法。阅读对象为需要基于 H3C CAS 3.0 云计算管理平台进行二次开发的客户开发人员和测试人员。

2 配置前提

本文档中的配置均是在实验室环境下进行的配置和验证，配置前服务器和软件的所有参数均采用出厂时的缺省配置。如果您已经对被测试对象进行了配置，为了保证配置效果，请确认现有配置和以下举例中的配置不冲突。

3 配置环境

3.1 服务器

本文档不严格与具体硬件服务器型号对应，如果使用过程中与产品实际情况有差异，请参考相关产品手册，或以设备实际情况为准。本文档使用的服务器型号与配置如下表所示，该环境不作为实际部署时的强制环境或推荐环境，只需要服务器能够兼容 H3C CAS 3.0 云计算管理平台即可完成本配置。

配置项	说明
服务器 #1 (H3C CAS CVM 虚拟化管理平台)	HP ProLiant BL460c Gen8 <ul style="list-style-type: none">CPU: 2 路 8 核, Intel Xeon E5-2670 @ 2.60GHz内存: 32 GB

服务器 #2 (H3C CAS CVK虚拟化内核系统)	HP ProLiant BL460c Gen8 <ul style="list-style-type: none"> • CPU: 2 路 8 核, Intel Xeon E5-2670 @ 2.60GHz • 内存: 32 GB
--------------------------------	---

3.2 软件

软件	版本
服务器虚拟化管理软件	H3C CAS-E0301 (KVM Kernel: 4.1.0)
虚拟机操作系统	Windows Server 2008 R2数据中心版64位 <ul style="list-style-type: none"> • 也可以使用 Windows 7、Windows XP 或者 Linux 操作系统, 本文档以 Windows Server 2008 R2 数据中心版 64 位为例。
Java SE开发工具包	Java SE Development Kit 8u92 64位 <ul style="list-style-type: none"> • 下载地址: http://www.oracle.com/
Web服务器	Apache Tomcat 8.0.35 64位Windows版 <ul style="list-style-type: none"> • 下载地址: http://tomcat.apache.org/download-80.cgi
Java集成开发环境	Eclipse IDE for Java EE Developers Mars.2 (4.5.2) 64位Windows版 <ul style="list-style-type: none"> • 下载地址: http://www.eclipse.org/downloads/
HTTP客户端编程工具包	HttpClient 4.1 (Binary) <ul style="list-style-type: none"> • 下载地址: http://hc.apache.org/downloads.cgi

4 组网拓扑

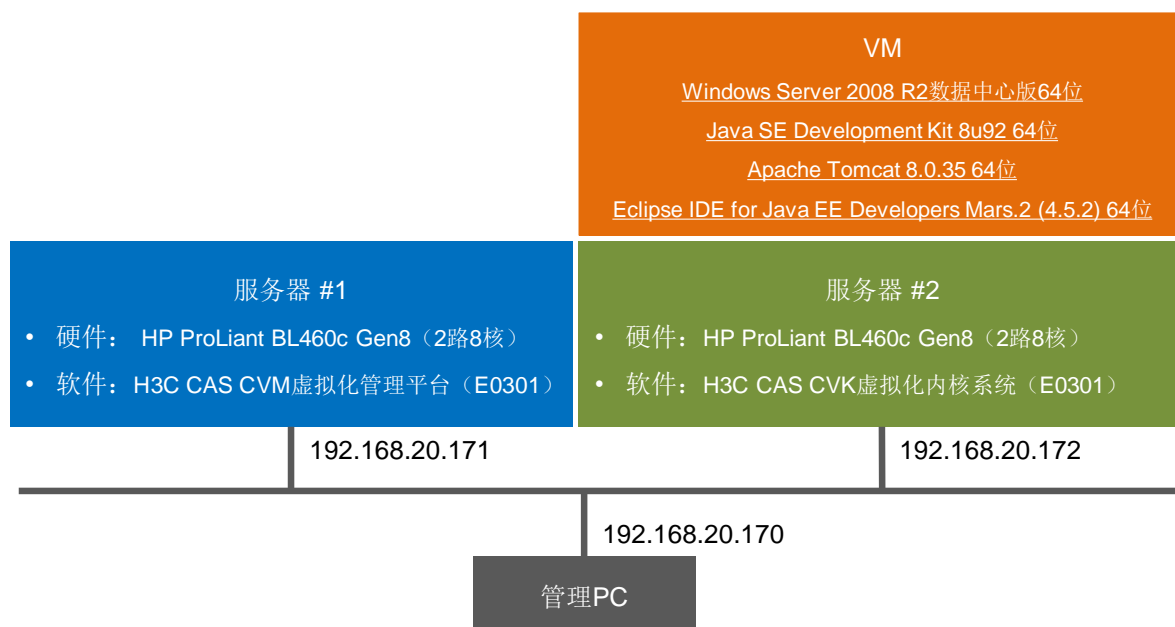


图1 H3C CAS 3.0 REST API 接口调用测试逻辑拓扑图

- 在服务器 #1 上安装 H3C CAS CVM 虚拟化管理平台，在服务器 #2 上安装 H3C CAS CVK 虚拟化系统。
- 在服务器 #2 上创建一个新的虚拟机，安装 Windows Server 2008 R2 数据中心版 64 位操作系统，在操作系统内安装 JDK、Tomcat 和 Eclipse 等开发软件。
- 在虚拟机操作系统内的 Eclipse 集成开发环境中使用 Java 调用服务器 #1 上的 H3C CAS CVM 虚拟化管理平台对外提供的 REST API 接口，实现自己想实现的功能效果。
- 本地管理 PC 机作为客户端，访问虚拟机操作系统内 Tomcat 提供的 Web 服务，验证开发效果。

5 操作指导

5.1 搭建开发环境

5.1.1 创建虚拟机

系统管理员登录 H3C CAS CVM 虚拟化管理平台（服务器 #1），在服务器 #2 上创建 1 个新的虚拟机（VM），虚拟机配置如下表所示。

资源	大小
虚拟CPU个数	1路2核
虚拟内存大小	4GB
虚拟磁盘大小	50GB（本地存储）
虚拟磁盘类型	默认（1 * Virtio，智能格式）
虚拟磁盘格式	默认（智能格式，QCOW2）
虚拟磁盘缓存模式	默认（directsync）
虚拟网卡类型	默认（1 * Virtio）
虚拟交换机	默认（vSwitch0）
虚拟机IP地址	DHCP自动分配



注意

上述虚拟机资源配置仅为测试环境下的配置，不作为生产环境中业务虚拟机的推荐配置。生产环境中的虚拟机配置应该根据业务系统本身对 CPU、内存、磁盘和网卡等资源的实际需求进行评估和测试后最终确定。

5.1.2 安装虚拟机

步骤1 通过控制台（VNC）为虚拟机挂载 Windows Server 2008 R2 64 位操作系统光盘镜像。



图2 为虚拟机挂载 Windows Server 2008 R2 数据中心版 64 位操作系统 ISO 镜像

步骤2 在控制台（VNC）窗口中，依次点击“操作”/“启动”菜单，启动虚拟机，按照提示逐步完成虚拟机操作系统的安装。

步骤3 【可选】在 H3C CAS CVM 虚拟化管理平台中，为虚拟机挂载 CAStools 工具软件。

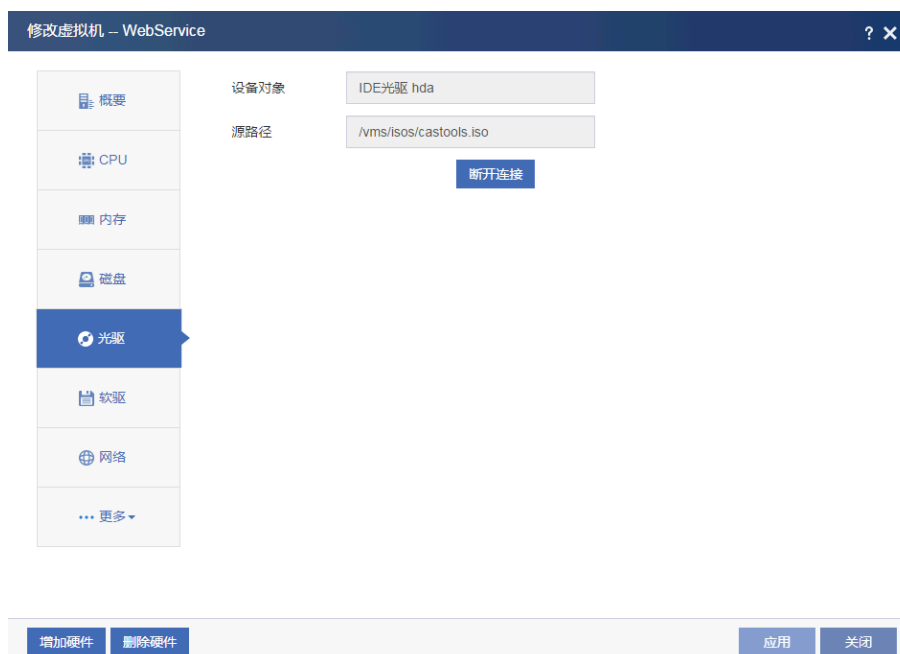


图3 为虚拟机挂载 CAStools 工具软件



说明

安装 CASTools 工具的目的主要是：

- 实时获取虚拟机的 CPU 利用率、内存利用率、磁盘 I/O 吞吐量、网络吞吐量、CPU 工作主频、磁盘请求 IOPS、磁盘 I/O 延迟、磁盘利用率、分区占用率等监控信息，以便系统管理员在 H3C CAS CVM 虚拟化管理平台中对虚拟机运行状况进行集中监管。
- 通过安装在操作系统内的轻代理工具，便于系统管理员通过 H3C CAS CVM 虚拟化管理平台对虚拟机操作系统级别的网络参数进行设置，例如，IP 地址、掩码、DNS 地址、网关等。
- 通过 CASTools 工具提供的虚拟串口通道，保持与 H3C CAS CVK 虚拟化内核系统的实时通信，判定虚拟机的存活状态，以实现操作系统蓝屏（Windows BSoD）和崩溃（Linux Panic）等异常情况的高可靠性。
- 通过 CASTools 工具提供的虚拟串口通道，保持与 H3C CAS CVM 虚拟化管理平台的实时通信，判定业务的存活状态，以实现操作系统内业务系统故障时的高可靠性。
- 通过 CASTools 工具提供的 NTP 时钟同步功能，自动定时与所在的物理主机同步时钟，虚拟机发生迁移后，自动定时与目标物理主机时钟同步。

5.1.3 安装 Java SE 开发工具包

步骤1 从 Oracle 官方网站上下载 Java SE Development Kit(下载地址：<http://www.oracle.com/>)，本文档以 Java SE Development Kit 8u92（Windows 64 位）为例。

步骤2 将下载好的 Java 开发工具包上传到虚拟机操作系统内，并双击 exe 文件开始安装，在安装过程中，会提示安装 JRE。



图4 JRE 安装



说明

JDK 是 Java 的开发平台，在编写 Java 程序时，需要 JDK 进行编译处理；JRE 是 Java 程序的运行环境，包含了 JVM 的实现及 Java 核心类库，编译后的 Java 程序必须使用 JRE 执行。在下载 JDK 安装包中集成了 JDK 与 JRE，所以在安装 JDK 过程中会提示安装 JRE。

5.1.4 配置 Java 环境变量

步骤1 在虚拟机操作系统内，打开计算机属性对话框，点击“高级系统设置”链接，在弹出的“系统属性”对话框中，点击<环境变量>按钮。



图5 操作系统属性

步骤2 在“系统变量”属性框中，点击<新建>按钮，在弹出的“新建系统变量”对话框中，输入“变量名”为“JAVA_HOME”，“变量值”为 JDK 安装路径，例如：“C:\Program Files\Java\jdk1.8.0_92\”，点击<确定>按钮。

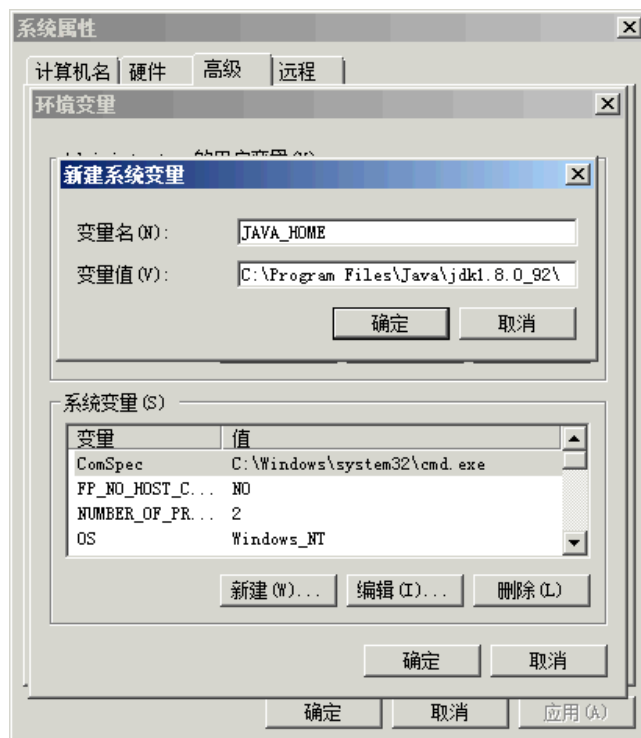


图6 设置 Java 环境变量：Java 安装路径

步骤3 编辑系统变量“Path”，在“变量值”的最前面加上：“%JAVA_HOME%\bin;”，点击<确定>按钮。

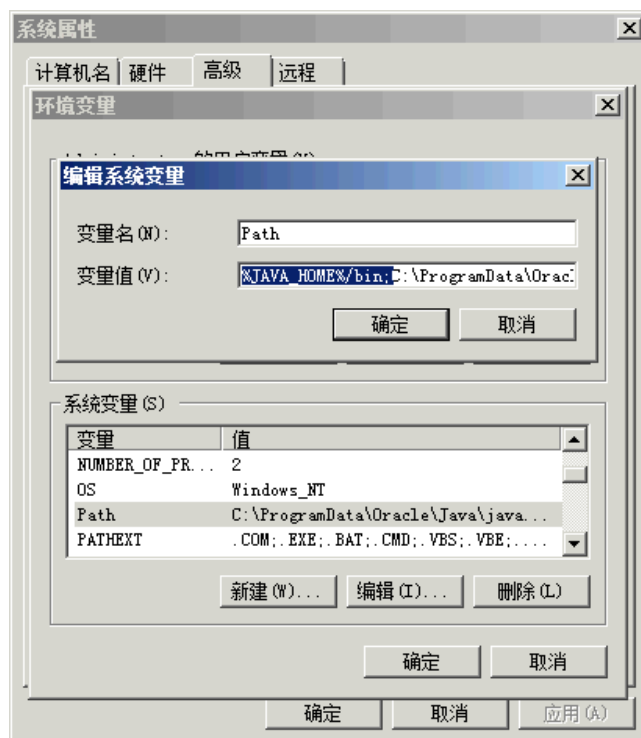


图7 编辑系统变量

步骤4 新建系统变量“CLASSPATH”，变量值为“%JAVA_HOME%\lib”，使 Java 自动从目录中发现 jar 文件类。



图8 设置 Java 环境变量：设置 jar 文件类路径

步骤5 验证安装是否成功。打开命令行窗口，运行 JDK 命令，如“javac”，效果如下图所示。

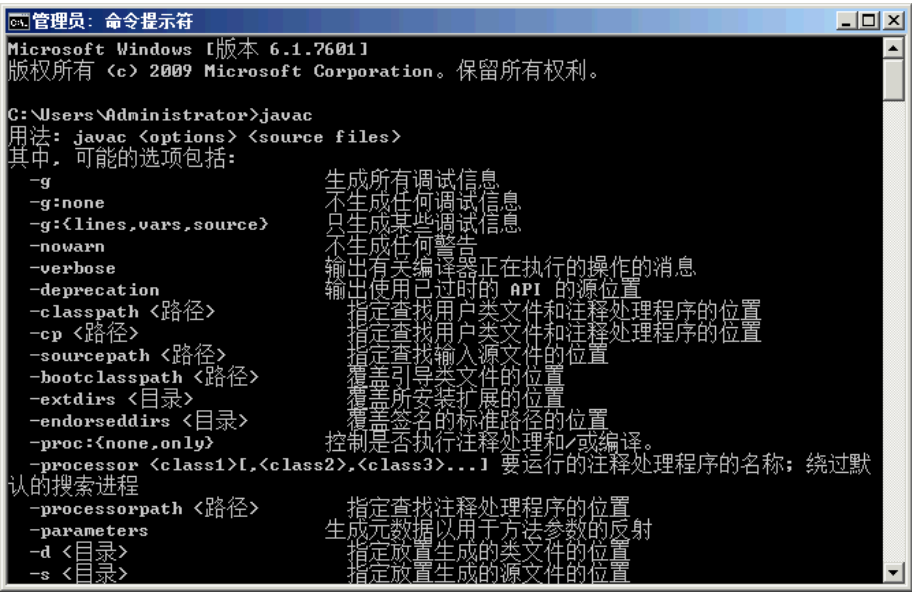


图9 验证 Java SE 开发环境安装结果

5.1.5 安装 Tomcat

步骤1 从 Tomcat 官方网站上下载 Apache Tomcat（下载地址：<http://tomcat.apache.org/>），本文档以 Apache Tomcat 8.0.35（Windows 64 位）为例。

步骤2 将下载好的 Tomcat 压缩文件上传到虚拟机操作系统内，并解压缩到任意目录，如“C:\”，解压缩后目录结构如下图所示。

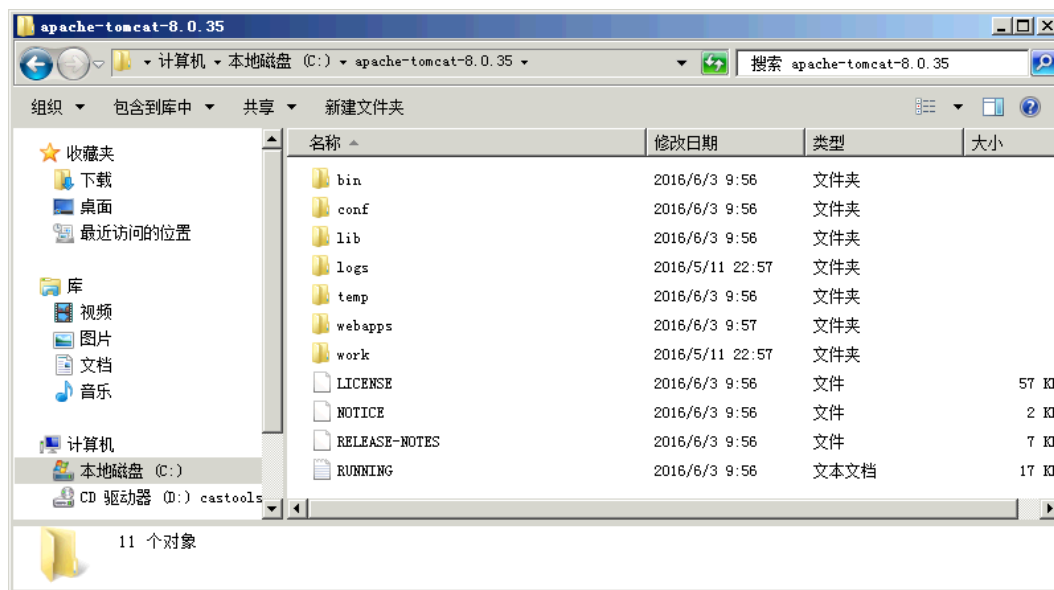


图10 Tomcat 目录结构

步骤3 进入 Tomcat 目录下的“bin”子目录，双击“startup.bat”批处理文件，运行 Tomcat 服务。

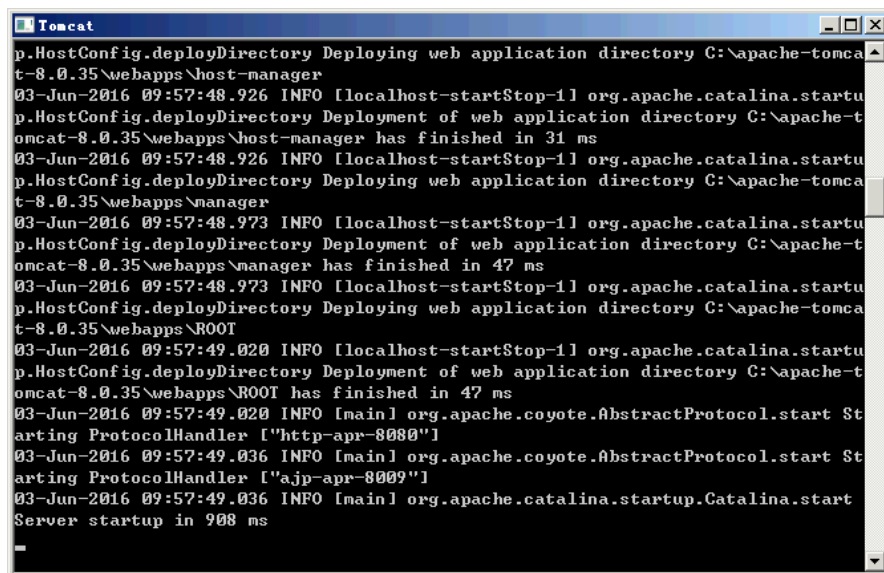


图11 Tomcat 服务运行后台

步骤4 验证 Tomcat 服务运行结果。打开浏览器，在地址栏中键入“<http://localhost:8080>”后回车，如果出现如下界面，则表示 Tomcat 服务运行正常。

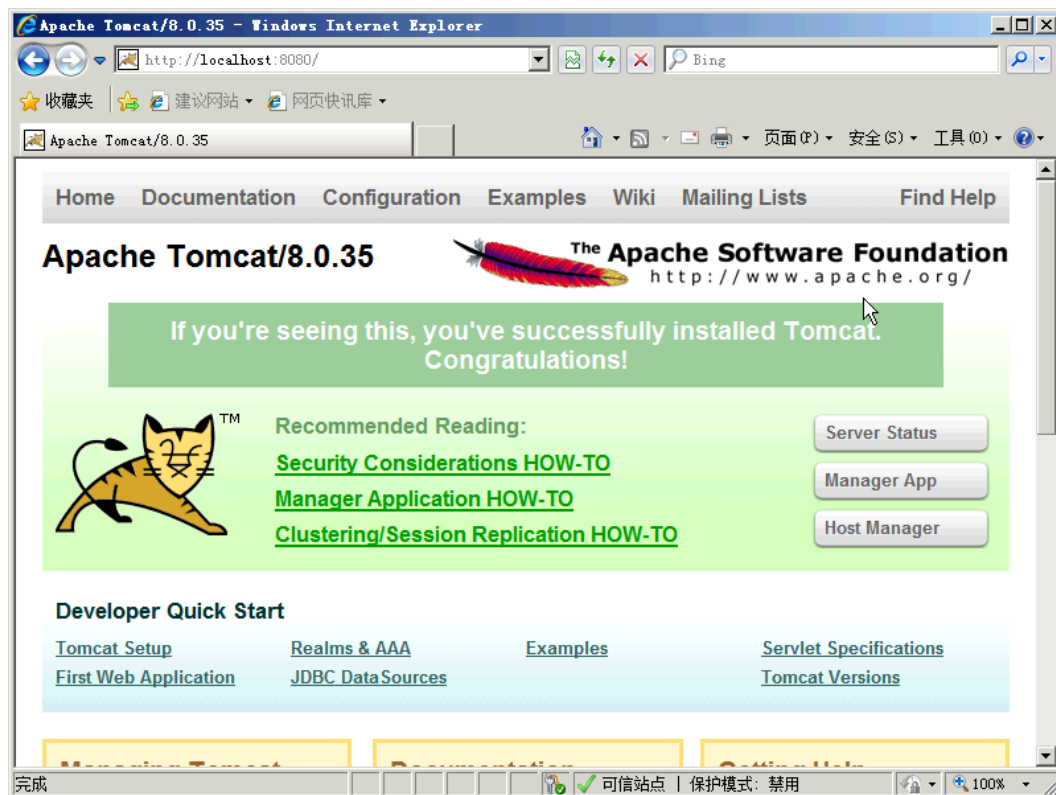


图12 Tomcat 服务运行结果

5.1.6 安装 Eclipse 集成开发环境

步骤1 从 Eclipse 官方网站上下载 Eclipse IDE for Java EE Developers（下载地址：<http://www.eclipse.org/downloads/>），本文档以 Eclipse IDE for Java EE Developers Mars.2（4.5.2）Windows 64 位为例。

步骤2 将下载好的 Eclipse IDE 上传到虚拟机操作系统内，并解压缩到任意目录，即完成了 Eclipse 集成开发环境的安装。

5.1.7 集成 Eclipse 和 Tomcat

在开发过程中，手工部署和运行 Web 项目，过程繁琐，效率低下，因此，需要将 Tomcat 服务配置到 Eclipse 集成开发环境中，为 Web 项目指定一个 Web 应用服务器，这样就可以在 Eclipse 中控制 Tomcat，并自动部署和运行 Web 项目。

步骤1 添加 Tomcat 运行时环境。

双击“eclipse.exe”可执行文件，在打开的 Eclipse 集成开发环境窗口中，依次点击“Window” / “Preferences” / “Server” / “Runtime Environments”，点击<Add...>按钮。

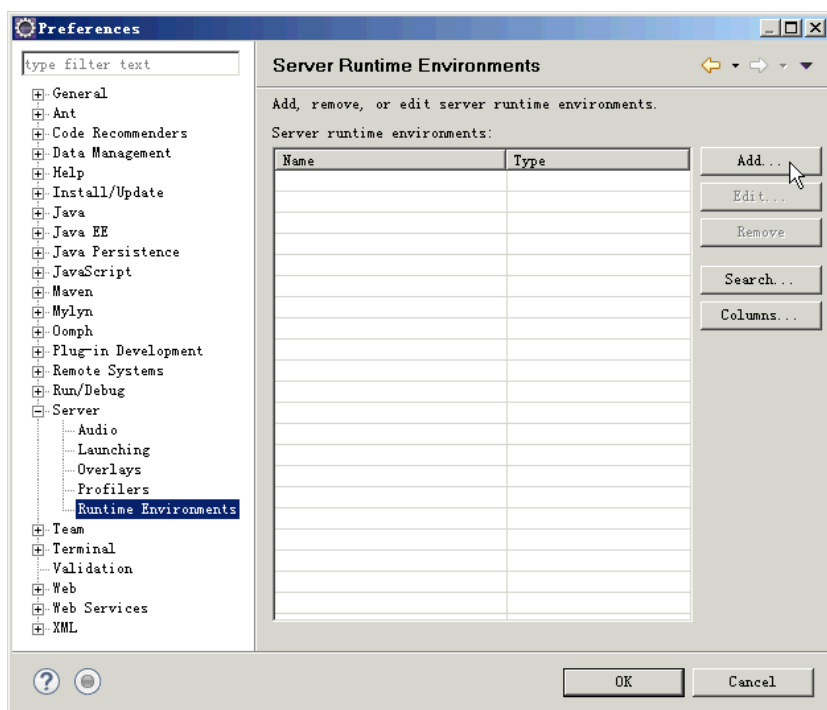


图13 准备添加 Tomcat 运行时环境

在弹出的“New Server Runtime Environment”配置向导中，选中“Apache Tomcat v8.0”，点击<Next>按钮。

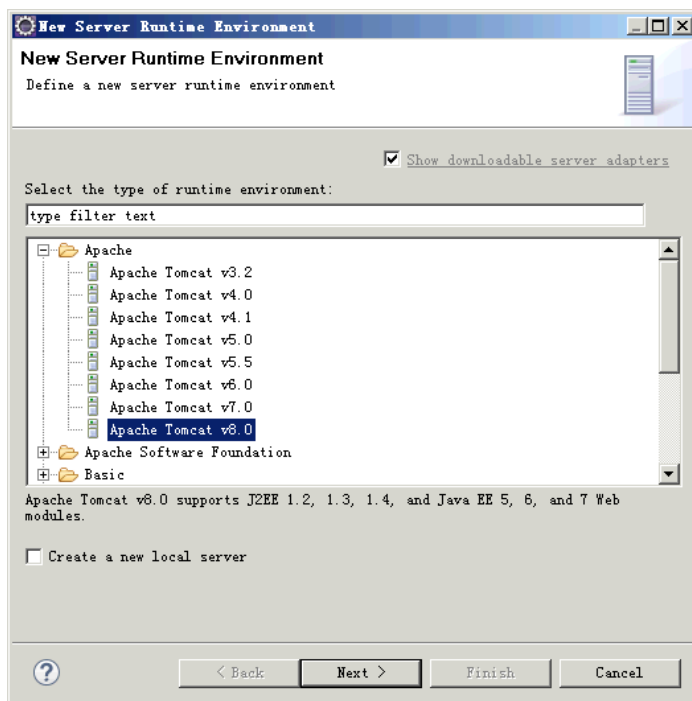


图14 创建新的运行时环境配置向导第 1 步 - 选择运行时环境类型

点击<Browse...>按钮，选择 Tomcat 安装路径，在“JRE”下拉列表框中，选择上述安装的 JRE 环境，点击<Finish>按钮。

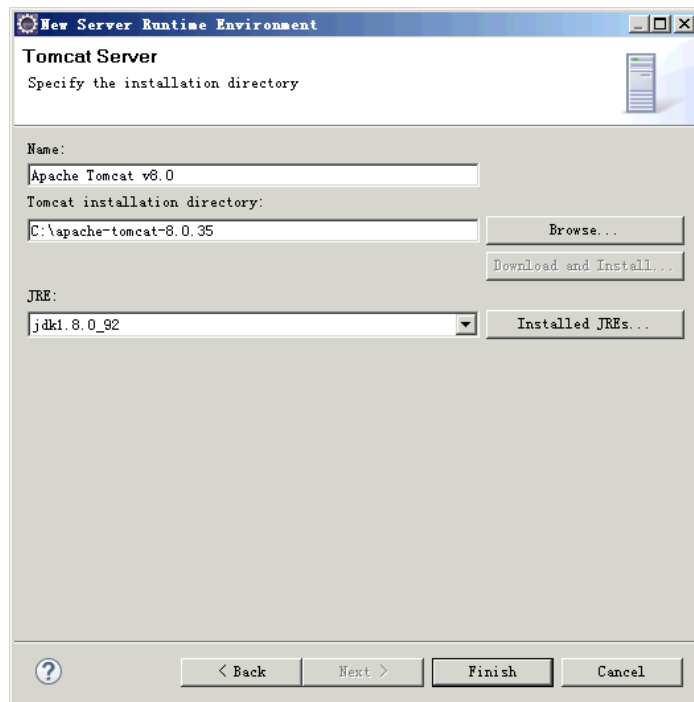


图15 创建新的运行时环境配置向导第 2 步 - 指定安装路径

步骤2 指定支持中文的编码格式。

在 Eclipse 集成开发环境窗口中，依次点击“Window”/“Preferences”/“Web”/“JSP Files”，在“Encoding”下拉列表框中选中“ISO 10646/Unicode(UTF-8)”，点击<OK>按钮。

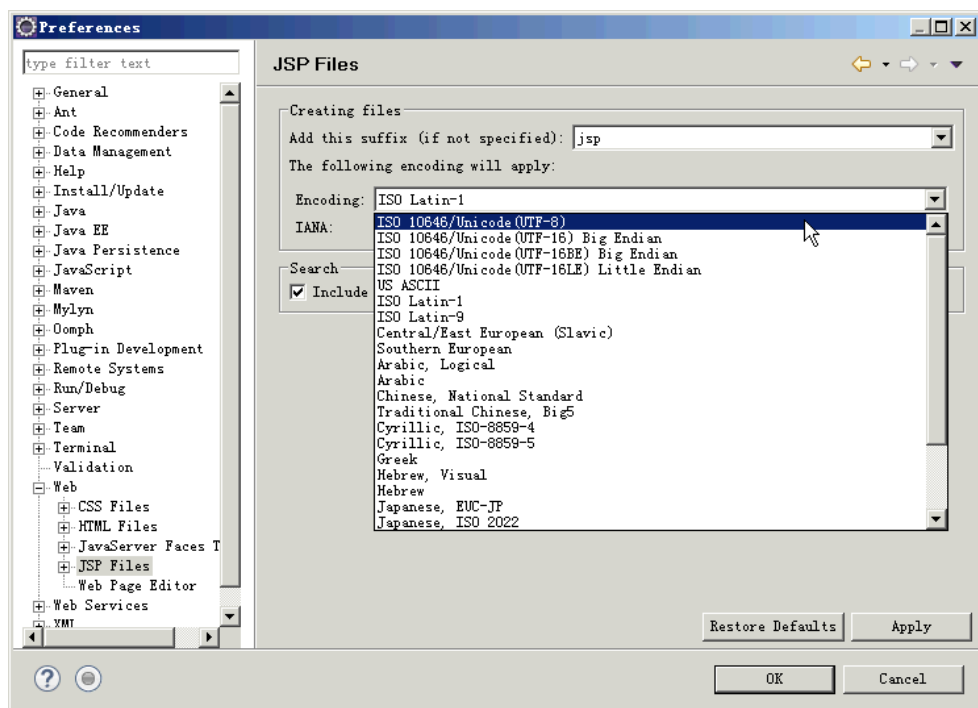


图16 选择支持中文的编码格式

步骤3 为 Eclipse 指定 Web 浏览器。

在 Eclipse 集成开发环境窗口中, 依次点击“Window”/“Preferences”/“General”/“Web Browser”, 选中 “Use external web browser”, 勾选 “Internet Explorer”, 点击<OK>按钮。

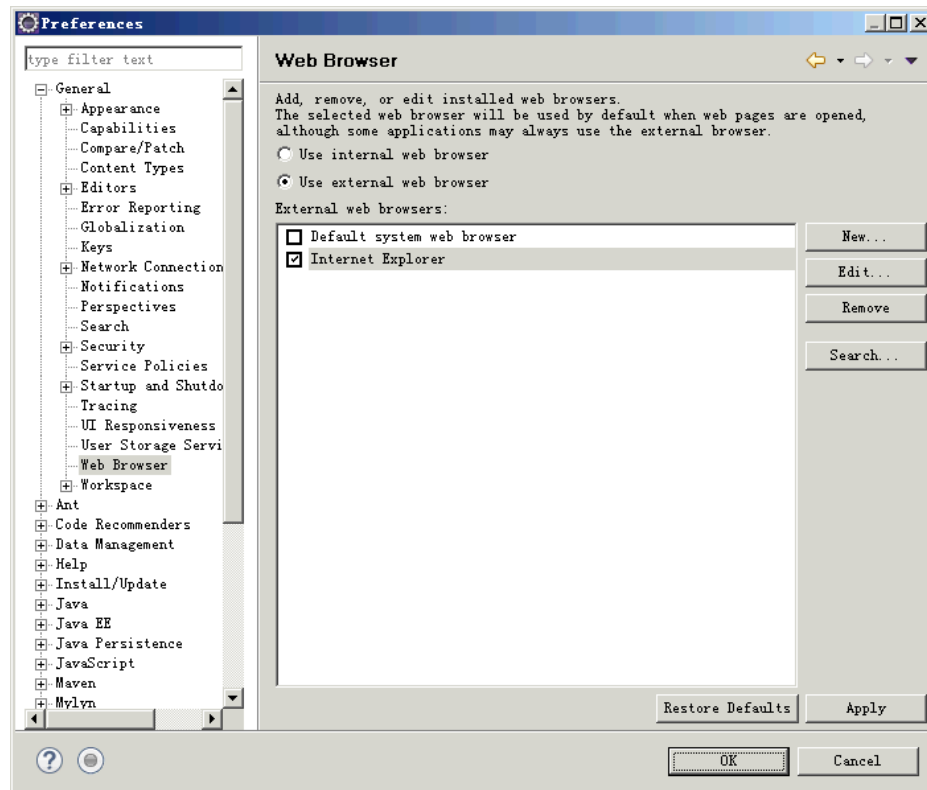


图17 为 Eclipse 指定 Web 浏览器

5.2 创建开发工程

5.2.1 下载 HTTP 客户端编程工具包

步骤1 从 HttpComponents 官方网站上下载 HTTP 客户端编程工具包（下载地址：<http://hc.apache.org/downloads.cgi>），本文档以 HttpClient 4.1（Binary）为例。

步骤2 将下载好的 HttpClient 4.1 压缩文件上传到虚拟机操作系统内，并解压缩到任意目录，如“C:\”。

5.2.2 创建 Java 工程

步骤1 在 Eclipse 集成开发环境窗口中, 依次点击 “File” / “New” / “Project”。

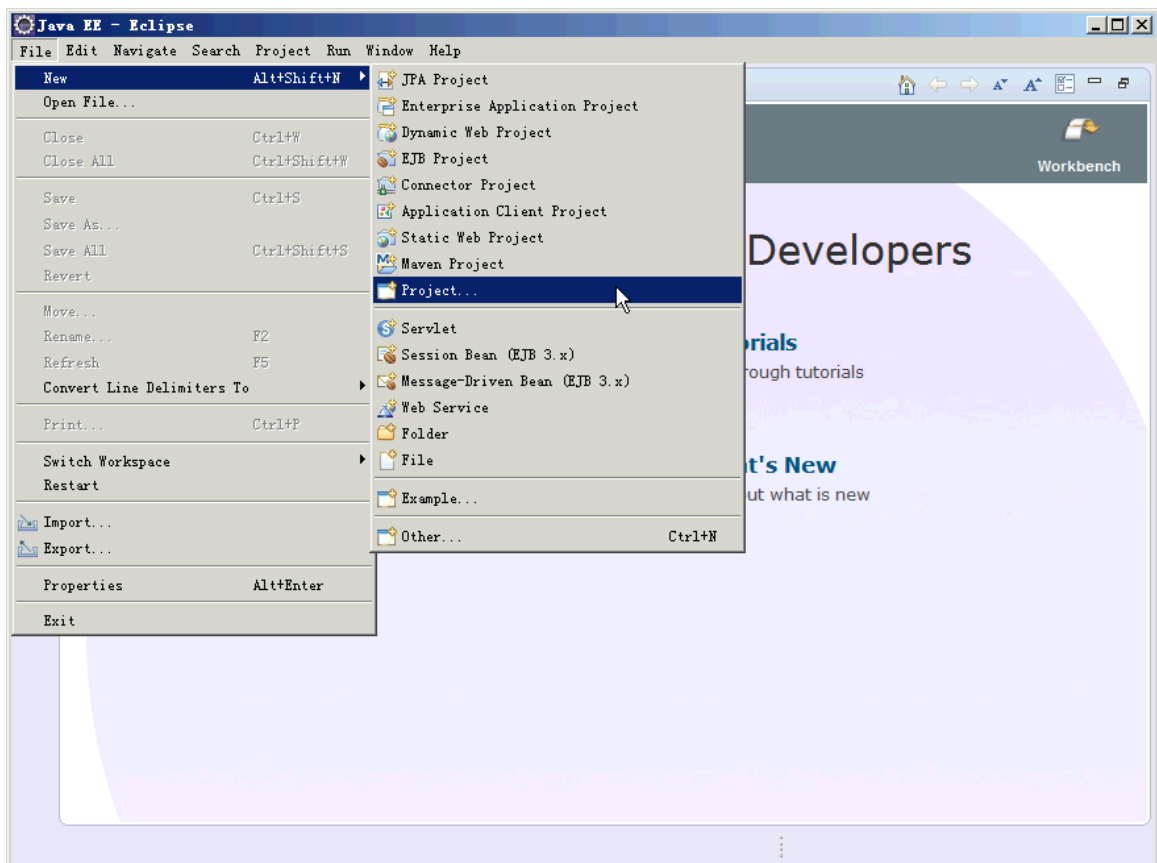


图18 新建工程

步骤2 在弹出的“New Project”配置向导中，选择“Java Project”，点击<Next>按钮。

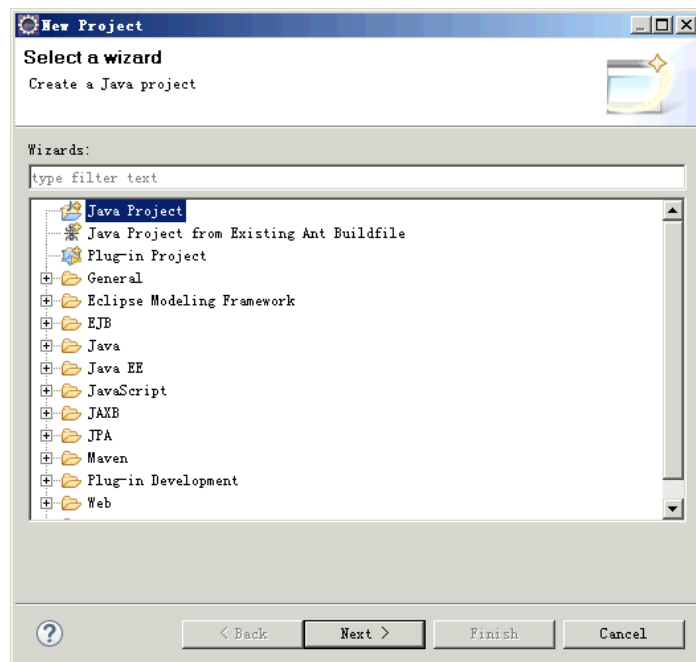


图19 创建工程配置向导 1 – 选择工程类型

步骤3 在“Project name”中键入工程名称，如“MyCloud”，使用指定的 JRE，点击<Next>按钮。



图20 创建工程配置向导 2 – 指定工程名称和 JRE

步骤4 选中“Libraries”选项卡，点击<Add External JARs...>按钮，选择 HTTP 客户端编程工具包“lib”子目录下的“httpcore-4.1.jar”、“httpclient-4.1.jar”和“commons-logging-1.1.1.jar”等文件，将其作为外部库依赖包添加到编译路径，点击<Finish>按钮，完成 Java 工程的创建。

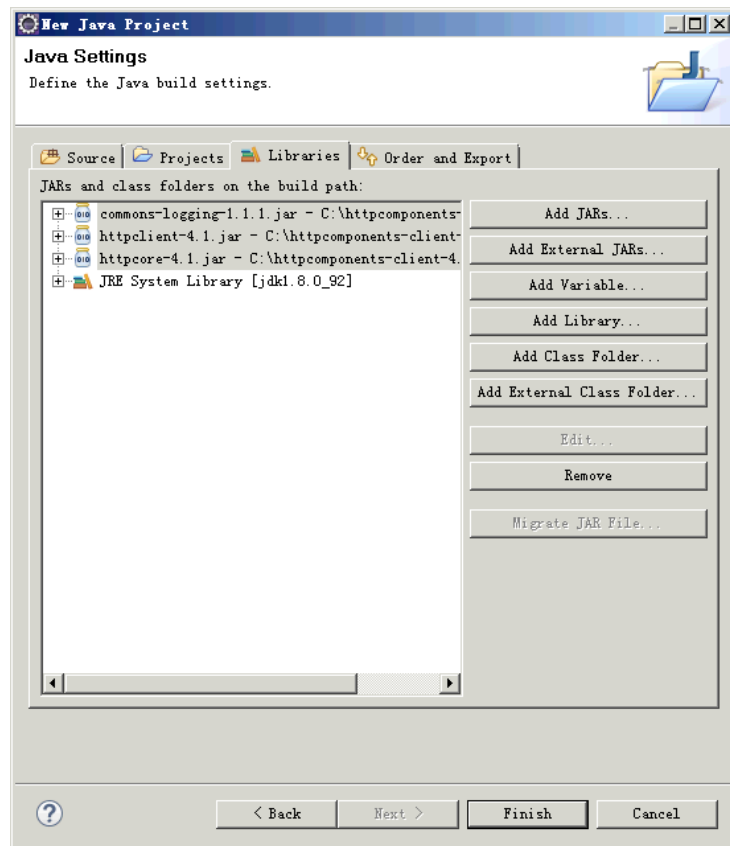


图21 创建工程配置向导 3 – 定义 Java 编译设置

5.2.3 创建 Java 类

步骤1 鼠标右键点击上述创建的工程名称，在弹出的上下文菜单中，依次选择“New”/“Class”。

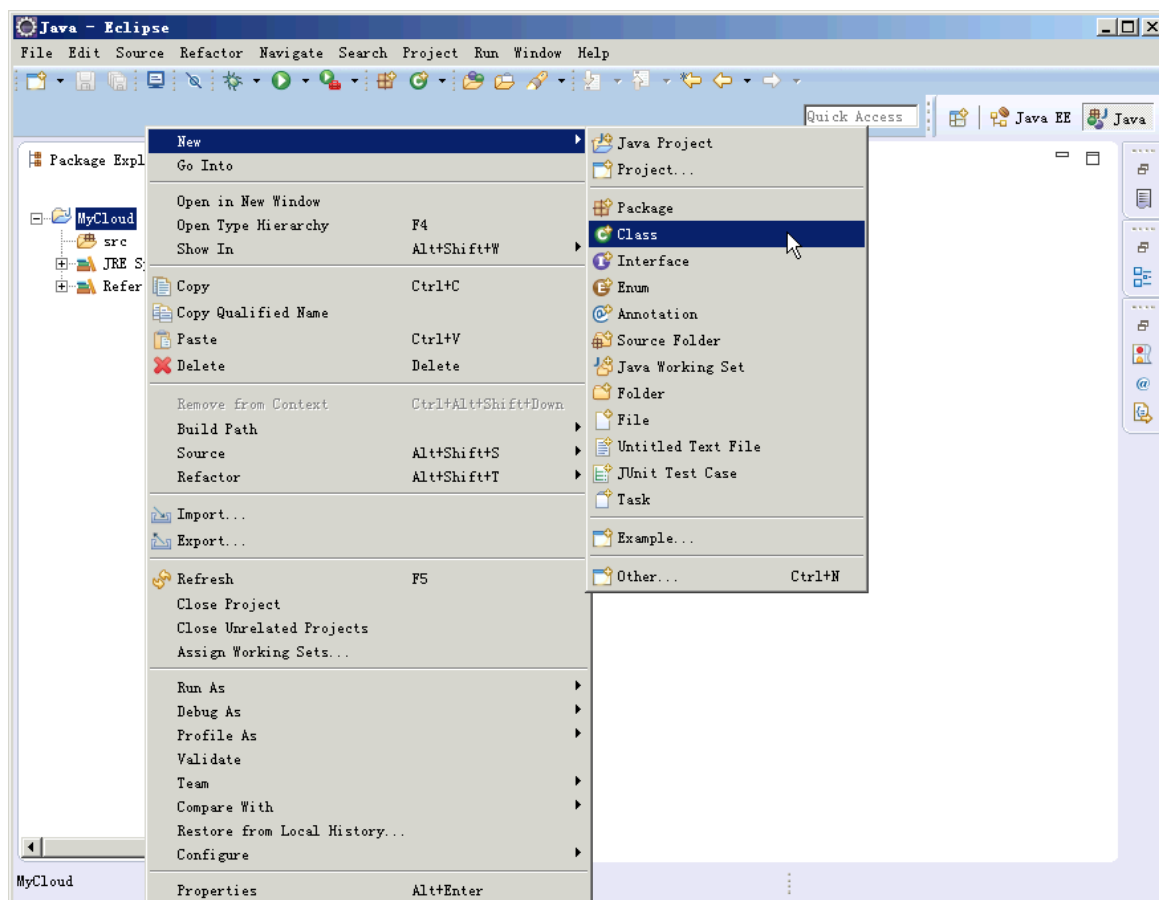


图22 为 Java 工程创建新的类

步骤2 在“Package”输入框中键入包名，如“com.mycompany.rest”，在“Name”输入框中键入类名，如“MyClient”，勾选“public static void main (String[] args)”，点击<Finish>按钮。

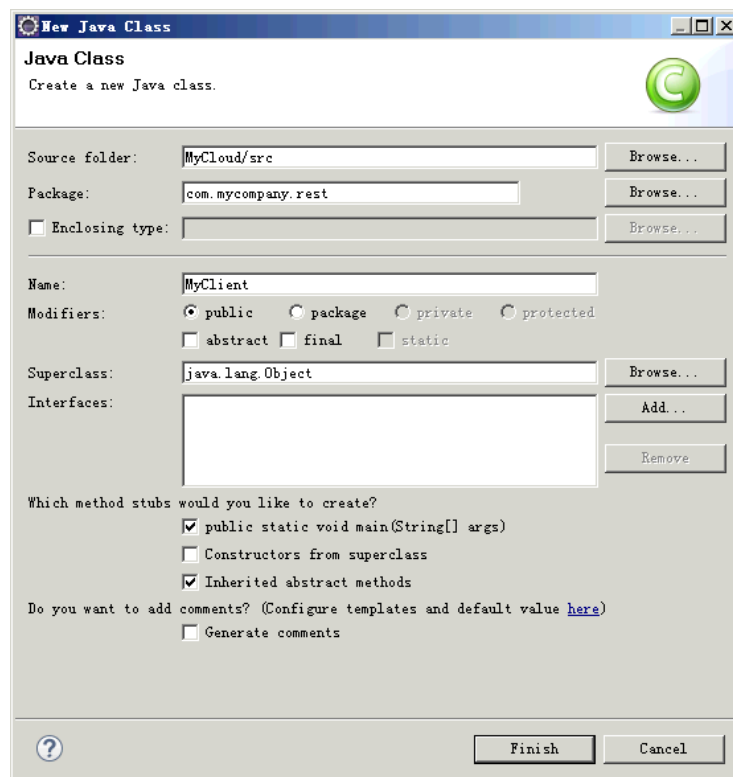


图23 创建新的 Java 类

5.3 调用H3C CAS 3.0 REST API接口

H3C CAS 3.0 云计算管理平台是一个合作开放的、聚焦于底层虚拟化资源集中统一管理的 IaaS 基础管理平台，对外公开的 REST API 接口参考可以从 H3C 公司官方网站下载（下载地址为：http://www.h3c.com.cn/pub/VCF_API/04/index.html）。

REST 是基于 HTTP 协议的，任何对资源的操作行为都通过 HTTP 协议来实现。以往的 Web 开发大多数用的都是 HTTP 协议中的 GET 和 POST 方法，对其它方法很少使用，这实际上是因为对 HTTP 协议认识片面的理解造成的。HTTP 不仅仅是一个简单的承载数据的协议，而且是一个具有丰富内涵的网络协议，它不仅仅能对互联网资源进行唯一定位，还能告诉我们如何对该资源进行操作。HTTP 把一个资源的操作限制为 4 个方法：GET、POST、PUT 和 DELETE，这正是对资源 CRUD 操作的实现。由于资源和 URI 是一一对应的，执行这些操作的时候 URI 是没有变化的，这和以往的 Web 开发有很大的区别。正由于这一点，极大的简化了 Web 开发，也使得 URI 可以被设计成更为直观的反映资源的结构，这种 URI 的设计被称作 RESTful 的 URI。

H3C CAS 3.0 云计算管理平台对外公开的 API 接口都是基于 REST 风格的，客户无论是使用 Java、PHP、JSP，或者是 .NET 等编程语言，都可以采用相同的步骤和方法调用这些 API 接口，完成底层复杂的虚拟化操作功能。

下面以几个典型接口调用为例，描述 H3C CAS 3.0 云计算管理平台提供的 REST API 接口在 Java 编程环境中的使用方法。



注意

以下代码仅是调用 H3C CAS 3.0 REST API 接口的示例代码,并不一定是实现功能的最优代码实现。

5.3.1 查询虚拟机性能监控数据

步骤1 在新创建的类中,编写如下代码。

```
package com.mycompany.rest;

import org.apache.http.HttpResponse;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.apache.http.util.EntityUtils;

public class MyClient {
    public static DefaultHttpClient client;

    public static void main(String[] args) throws Exception {
        /* 获取所有主机列表 */
        getHost("http://192.168.20.171:8080/cas/casrs/host/");

        /* 获取某个主机上的所有虚拟机列表 */
        getVM("http://192.168.20.171:8080/cas/casrs/vm/vmList?hostId=2");

        /* 获取某个虚拟机的性能监控数据 */
        getVMperf("http://192.168.20.171:8080/cas/casrs/vm/id/9/monitor");

        /* 查询指定集群统计信息 */
        getClusterInfo("http://192.168.20.171:8080/cas/casrs/cluster/summary/2");
    }

    /* 获取所有主机列表 */
    public static void getHost (String url) throws Exception {
        DefaultHttpClient client = newInstance();
        HttpGet get = new HttpGet(url);
        get.addHeader("accept", "application/xml");
        HttpResponse response = client.execute(get);
        System.out.println(response.getStatusLine());
        System.out.println(EntityUtils.toString(response.getEntity()));
    }

    /* 获取某个主机上的所有虚拟机列表 */
    public static void getVM (String url) throws Exception {
        DefaultHttpClient client = newInstance();
        HttpGet get = new HttpGet(url);
        get.addHeader("accept", "application/xml");
        HttpResponse response = client.execute(get);
        System.out.println(response.getStatusLine());
        System.out.println(EntityUtils.toString(response.getEntity()));
    }

    /* 获取某个虚拟机的性能监控数据 */
}
```

```

public static void getVMperf(String url) throws Exception {
    DefaultHttpClient client = newInstance();
    HttpGet get = new HttpGet(url);
    get.addHeader("accept", "application/xml");
    HttpResponse response = client.execute(get);
    System.out.println(response.getStatusLine());
    System.out.println(EntityUtils.toString(response.getEntity(), "UTF-8"));
}


/* 查询指定集群统计信息 */
public static void getClusterInfo(String url) throws Exception {
    DefaultHttpClient client = newInstance();
    HttpGet get = new HttpGet(url);
    get.addHeader("accept", "application/xml");
    HttpResponse response = client.execute(get);
    System.out.println(response.getStatusLine());
    System.out.println(EntityUtils.toString(response.getEntity(), "UTF-8"));
}

/* 以实例方式实现 H3C CAS 3.0 云计算管理平台认证 */
public static DefaultHttpClient newInstance() {
    if (client == null) {
        client = new DefaultHttpClient();
        client.getCredentialsProvider().setCredentials(
            new AuthScope("192.168.20.171", 8080, "VMC RESTful Web Services"),
            new UsernamePasswordCredentials("admin", "admin"));
    }
    return client;
}
}

```

其中：

- **192.168.20.171**：H3C CAS CVM 虚拟化管理平台的 IP 地址，请根据实际环境修改；
- **8080**：访问 H3C CAS CVM 虚拟化管理平台的端口号；
- **VMC RESTful Web Services**：H3C CAS 3.0 REST API 接口认证域；
- **http://192.168.20.171:8080/cas/casrs/host/**：H3C CAS 3.0 提供的获取所有主机信息 API 接口，IP 地址请根据实际环境中修改。
- **admin**：访问 H3C CAS CVM 虚拟化管理平台的帐号和密码。

步骤2 在 Eclipse 集成开发环境窗口工具栏中，点击 “” 图标，或使用快捷键 “Ctrl+F11” 运行上述代码，在 Eclipse 控制台（Console）窗口将获取到相应的 XML 输出结果，以查询某个虚拟机性能监控数据为例。

```

<domain>
<id>9</id>                                # 虚拟机 ID
<cpuRate>0.0</cpuRate>                    # CPU 利用率
<memRate>35.78</memRate>                  # 内存利用率
<status>2</status>                        # 运行状态
<uuid>9e0ec565-cb40-4fbb-858e-fbe423668ffb</uuid> # 虚拟机 UUID
<disk>
<device>vda</device>                      # 磁盘镜像物理名称
<time>1464948210</time>                  # 自 1970 年 1 月 1 日到当前时刻的秒数

```

<ioStat>3.5333333333333333</ioStat>	# I/O 总吞吐量, 单位: KBps
<readStat>0.0</readStat>	# I/O 读吞吐量, 单位: KBps
<writeStat>3.5333333333333333</writeStat>	# I/O 写吞吐量, 单位: KBps
<readRequet>0.0</readRequet>	# 磁盘读 IOPS
<writeRequet>0.0</writeRequet>	# 磁盘写 IOPS
<readLatency>0.0</readLatency>	# 磁盘读 I/O 延迟
<writeLatency>11.003</writeLatency>	# 磁盘写 I/O 延迟
</disk>	
<net>	
<mac>0c:da:41:1d:6a:a2</mac>	# 虚拟网卡 MAC 地址
<time>1464948210</time>	# 自 1970 年 1 月 1 日到当前时刻的秒数
<readFlow>0.03008575439453125</readFlow>	# 网络读流量, 单位: Mbps
<readPackets>505.0</readPackets>	# 网络读报文数
<writeFlow>0.0</writeFlow>	# 网络写流量, 单位: Mbps
<writePackets>0.0</writePackets>	# 网络写报文数
</net>	
<partition>	
<device>C:</device>	# 逻辑磁盘名称
<usage>12.11</usage>	# 逻辑磁盘利用率
</partition>	
</domain>	

5.3.2 JSP 调用获取所有虚拟机列表

1. Java 类

新创建 Java 类, 编写代码如下:

```
package com.customer.demo;

import java.io.InputStream;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;

public class RESTAPI {

    public static DefaultHttpClient client;
    public static String strCVMIP = new String("192.168.20.25");
    public static int nCVMPort = 8080;
    public static String strCVMUrl = new String("http://" + strCVMIP + ":" +
Integer.toString(nCVMPort));
    public static String strHostPool = new String("/cas/casrs/hostpool/all");
    public static String strGetVMListByHostPool = new String("/cas/casrs/vm/vmList?hpId=");

    // 获取所有主机池下的所有虚拟机列表
```

```

public ArrayList<VM> getAllVMList () throws Exception {
    ArrayList<VM> listAllVMs = new ArrayList<VM>();
    ArrayList<Long> hostPoolList = new ArrayList<Long>();
    hostPoolList = getHostPoolList();
    for(int nIndex = 0; nIndex < hostPoolList.size(); nIndex++) {
        ArrayList<VM> listVMs = new ArrayList<VM>();
        listVMs = getVMListByHostPoolID(hostPoolList.get(nIndex));
        listAllVMs.addAll(listVMs);
    }
    return listAllVMs;
}

// 根据主机池 ID 获取该主机池下的所有虚拟机列表
public ArrayList<VM> getVMListByHostPoolID (long nHostPoolID) throws Exception {
    DefaultHttpClient client = newInstance();
    ArrayList<VM> VMList = new ArrayList<VM>();
    HttpGet httpGet = new HttpGet(strCVMUrl + strGetVMListByHostPool +
Long.toString(nHostPoolID));
    httpGet.addHeader("accept", "application/xml");
    HttpResponse response = client.execute(httpGet);
    HttpEntity entity = response.getEntity();
    if (null != entity) {
        InputStream in = entity.getContent();
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(in);
        document.getDocumentElement().normalize();

        NodeList list = document.getElementsByTagName("domain");
        for (int i = 0; i < list.getLength(); i++) {
            VM vm = new VM();

            vm.setID(Long.parseLong(document.getElementsByTagName("id").item(i).getFirstChild().getNodeValue()));

            vm.setHostName(document.getElementsByTagName("hostName").item(i).getFirstChild().getNodeValue());

            vm.setTitle(document.getElementsByTagName("title").item(i).getFirstChild().getNodeValue());

            NodeList nodeDesc = document.getElementsByTagName("description");
            if (nodeDesc != null && nodeDesc.getLength() > 0 && nodeDesc.item(i) != null
&& nodeDesc.item(i).getFirstChild() != null) {

                vm.setDesc(document.getElementsByTagName("description").item(i).getFirstChild().getNodeValue());
            } else {
                vm.setDesc("");
            }

            vm.setStatus(document.getElementsByTagName("vmStatus").item(i).getFirstChild().getNodeValue());

            vm.setCPU(Integer.parseInt(document.getElementsByTagName("cpu").item(i).getFirstChild().getNodeValue()));

```



```

        vm.setMem(Integer.parseInt(document.getElementsByTagName("memory").item(i).getFirstChild().getNodeValue()));
        double dbCPUUsage =
Double.valueOf(document.getElementsByTagName("cpuRate").item(i).getFirstChild().getNodeValue());
        vm.setCPUUsage((double)Math.round(dbCPUUsage*100)/100);
        double dbMemUsage =
Double.valueOf(document.getElementsByTagName("memRate").item(i).getFirstChild().getNodeValue());
        vm.setMemUsage((double)Math.round(dbMemUsage*100)/100);
        NodeList nodeOS = document.getElementsByTagName("osDesc");
        if (nodeOS != null && nodeOS.getLength() > 0 && nodeOS.item(i) != null && nodeOS.item(i).getFirstChild() != null) {

            vm.setOS(document.getElementsByTagName("osDesc").item(i).getFirstChild().getNodeValue());
        } else {
            vm.setOS("Unknown");
        }
        VMList.add(vm);
    }
}
return VMList;
}

// 获取主机池列表
public ArrayList<Long> getHostPoolList () throws Exception {
    DefaultHttpClient client = newInstance();
    ArrayList<Long> hostPoolList = new ArrayList<Long>();
    HttpGet httpGet = new HttpGet(strCVMUrl + strHostPool);
    httpGet.addHeader("accpet", "application/xml");
    HttpResponse response = client.execute(httpGet);
    HttpEntity entity = response.getEntity();
    if (null != entity) {
        InputStream in = entity.getContent();
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(in);
        document.getDocumentElement().normalize();

        NodeList hostPool = document.getElementsByTagName("hostPool");
        for (int i = 0; i < hostPool.getLength(); i++) {
            String strID =
document.getElementsByTagName("id").item(i).getFirstChild().getNodeValue();
            hostPoolList.add(new Long(Long.parseLong(strID)));
        }
    }
    return hostPoolList;
}

// 认证实例
public static DefaultHttpClient newInstance() {
    if (client == null) {
        client = new DefaultHttpClient();
        client.getCredentialsProvider().setCredentials(
            new AuthScope(strCVMIP, nCVMPort, "VMC RESTful Web Services"),

```

```

        new UsernamePasswordCredentials("admin", "admin"));
    }

    return client;
}

public class VM {
    public long nID;           // 虚拟机 ID
    public String strHostName; // 所在主机名称
    public String strTitle;    // 虚拟机显示名称
    public String strDesc;     // 虚拟机描述
    public String strStatus;   // 虚拟机状态
    public int nCPU;           // 虚拟机 CPU 个数
    public int nMemory;        // 虚拟机内存大小
    public double dbCPUUsage;   // 虚拟机 CPU 利用率
    public double dbMemUsage;   // 虚拟机内存利用率
    public String strOS;       // 虚拟机操作系统

    public long getID() {
        return nID;
    }

    public void setID(long nID) {
        this.nID = nID;
    }

    public String getHostName() {
        return strHostName;
    }

    public void setHostName(String strHostName) {
        this.strHostName = strHostName;
    }

    public String getTitle() {
        return strTitle;
    }

    public void setTitle(String strTitle) {
        this.strTitle = strTitle;
    }

    public String getDesc() {
        return strDesc;
    }

    public void setDesc(String strDesc) {
        this.strDesc = strDesc;
    }

    public String getStatus() {
        return strStatus;
    }

    public void setStatus(String strStatus) {
        this.strStatus = strStatus;
    }
}

```

```

    }

    public int getCPU() {
        return nCPU;
    }

    public void setCPU(int nCPU) {
        this.nCPU = nCPU;
    }

    public int getMem() {
        return nMemory;
    }

    public void setMem(int nMemory) {
        this.nMemory = nMemory;
    }

    public double getCPUUsage() {
        return dbCPUUsage;
    }

    public void setCPUUsage(double dbCPUUsage) {
        this.dbCPUUsage = dbCPUUsage;
    }

    public double getMemUsage() {
        return dbMemUsage;
    }

    public void setMemUsage(double dbMemUsage) {
        this.dbMemUsage = dbMemUsage;
    }

    public String getOS() {
        return strOS;
    }

    public void setOS(String strOS) {
        this.strOS = strOS;
    }
}
}

```

2. JSP 界面

新建“Dynamic Web Project”，编写 JSP 网页文件，调用上述 Java 类获取所有虚拟机列表及信息。

```

<%@ page import="com.customer.demo.RESTAPI" %>
<%@ page import="java.util.ArrayList" %>
<%@ page language="java" contentType="text/html; charset=GB2312"
    pageEncoding="GB2312"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>

```

```
<meta http-equiv="Content-Type" content="text/html; charset=GB2312">
<title>MyCloud</title>
<style>
.process-bar
{
width:100%;
display:inline-block;
*zoom:1;
}
.pb-wrapper
{
border:1px solid gray;
border-style:solid none;
position:relative;
background:#cfd0d2;
}
.pb-container
{
border:1px solid gray;
border-style:none solid;
height:12px;
position:relative;
left:-1px;
margin-right:-2px;
font:1px/0 arial;
padding:1px;
}
.pb-highlight
{
position:absolute;
left:0;
top:0;
_top:1px;
width:100%;
opacity:0.6;
filter:alpha(opacity=60);
height:6px;
background:white;
font-size:1px;
line-height:0;
z-index:1
}
.pb-text
{
width:100%;
position:absolute;
left:0;
top:0;
text-align:center;
font:10px/12px arial;
}
.pb-value
{
height:100%;
width:50%;
background:#19d73d;
```

```

}
.pb-text
{
color:black;
}
.grid
{
border-collapse : collapse;
font:12px/1.5 arial;
background:white;
border:1px solid #d9d9d9;
width:100%;
}
.grid td,
.grid th
{
padding:5px
}
.grid th
{
background:#eee;
color:black;
text-align:left;
font-weight:normal
}
</style>
</head>
<body>
<table cellpadding="0" cellspacing="0" border="1" class="grid" rules="rows" frame="void">
<tr>
<th>虚拟机名称</th>
<th>描述</th>
<th>运行状态</th>
<th>所在主机</th>
<th>CPU 个数</th>
<th>内存容量</th>
<th>CPU 利用率</th>
<th>内存利用率</th>
<th>操作系统</th>
</tr>
<%
ArrayList<RESTAPI.VM> list = new ArrayList<RESTAPI.VM>();
RESTAPI rest = new RESTAPI();
list = rest.getAllVMList();
for(int nIndex = 0; nIndex < list.size(); nIndex++) {
%>
<tr>
<td width="15%"><%= list.get(nIndex).getTitle()%></td>
<td width="10%"><%= list.get(nIndex).getDesc()%></td>
<td width="5%">
<%
if(list.get(nIndex).getStatus().equalsIgnoreCase("running")) {
%>

<%
} else if(list.get(nIndex).getStatus().equalsIgnoreCase("paused")) {

```

```

%>
    
<%
} else {
%>
    
<%
}
%>
</td>
<td width="10%"><%= list.get(nIndex).getHostName() %></td>
<td width="5%"><%= list.get(nIndex).getCPU() %></td>
<td width="10%"><%= list.get(nIndex).getMem()/1024%> GB</td>
<td width="10%">
    <div class="process-bar">
        <div class="pb-wrapper">
            <div class="pb-highlight"></div>
            <div class="pb-container">
                <div class="pb-text"><%= list.get(nIndex).getCPUUsage() %>%</div>
                <div class="pb-value" style="width:<%=
list.get(nIndex).getCPUUsage() %>%"></div>
            </div>
        </div>
    </div>
</td>
<td width="10%">
    <div class="process-bar">
        <div class="pb-wrapper">
            <div class="pb-highlight"></div>
            <div class="pb-container">
                <div class="pb-text"><%= list.get(nIndex).getMemUsage() %>%</div>
                <div class="pb-value" style="width:<%=
list.get(nIndex).getMemUsage() %>%"></div>
            </div>
        </div>
    </div>
</td>
<td width="25%"><%= list.get(nIndex).getOS() %></td>
</tr>
<%
}
%>
</table>
</body>
</html>

```

页面执行结果如下图所示。

虚拟机名称	描述	运行状态	所在主机	CPU个数	内存容量	CPU利用率	内存利用率	操作系统
win2008r2-01		运行	tf-srm-server57	2	2 GB	0.17%	10.01%	Microsoft Windows Server 2008 R2 Datacenter Edition, 64-bit
win2012r2-64		运行	tf-srm-server57	2	2 GB	0.43%	10.09%	Windows Server 2012 R2 Datacenter, 64-bit
win2003		运行	tf-srm-server57	2	2 GB	0.25%	7.1%	Microsoft Windows Server 2003
mb-win2008r2_002		运行	tf-srm-server57	2	2 GB	0.19%	10.51%	Microsoft Windows Server 2008 R2 Datacenter Edition, 64-bit
mb-win2008r2_001		运行	tf-srm-server57	2	2 GB	0.16%	10.51%	Microsoft Windows Server 2008 R2 Datacenter Edition, 64-bit
mb-win2003_001		运行	tf-srm-server57	2	2 GB	0.23%	7.13%	Microsoft Windows Server 2003
mb-win2003_002		运行	tf-srm-server57	2	2 GB	0.29%	7.08%	Microsoft Windows Server 2003
mb-win2012r2_002		运行	tf-srm-server57	2	2 GB	0.2%	10.09%	Windows Server 2012 R2 Datacenter, 64-bit
mb-win2012r2_001		运行	tf-srm-server57	2	2 GB	0.19%	10.06%	Windows Server 2012 R2 Datacenter, 64-bit
360-center		停止	tf-srm-server24	2	4 GB	0.0%	0.0%	Other Linux(64-bit)
y2012_001		运行	tf-srm-server13	2	2 GB	0.3%	10.04%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_002		运行	tf-srm-server57	2	2 GB	0.29%	10.06%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_003		运行	tf-srm-server57	2	2 GB	0.35%	10.06%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_009		运行	tf-srm-server57	2	2 GB	0.21%	10.09%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_010		运行	tf-srm-server13	2	2 GB	0.28%	10.71%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_013		运行	tf-srm-server57	2	2 GB	0.34%	10.0%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_016		停止	tf-srm-server24	2	2 GB	0.0%	0.0%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_005		停止	tf-srm-server24	2	2 GB	0.0%	0.0%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_020		运行	tf-srm-server13	2	2 GB	0.29%	10.74%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_007		运行	tf-srm-server13	2	2 GB	0.12%	10.73%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_011		运行	tf-srm-server13	2	2 GB	0.16%	10.07%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_014		运行	tf-srm-server13	2	2 GB	0.17%	10.7%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_018		运行	tf-srm-server13	2	2 GB	0.2%	10.06%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_006		运行	tf-srm-server57	2	2 GB	0.4%	10.72%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_012		运行	tf-srm-server57	2	2 GB	0.28%	10.04%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_017		运行	tf-srm-server57	2	2 GB	0.31%	10.09%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_008		停止	tf-srm-server24	2	2 GB	0.0%	0.0%	Windows Server 2012 R2 Datacenter, 64-bit
y2012_019		运行	tf-srm-server57	2	2 GB	0.49%	10.7%	Windows Server 2012 R2 Datacenter, 64-bit

图24 使用 JSP 调用 REST API 获取所有虚拟机列表示例结果

5.3.3 单点登录

单点登录（Single Sign On，SSO）是目前比较流行的一种企业业务整合解决方案，用户只需要登录一次，即可访问所有相互信任的应用系统。在云计算环境中，运维管理人员可能同时需要对云管理平台、虚拟化管理平台、业务运维平台等多种软件进行管理，为了达到集中统一管理的目的，在已知这些分散的管理软件的登录帐号和密码情况下，运维管理人员期望能够一键免认证登录这些系统，以减少帐号和密码的输入过程。

H3C CAS 3.0 云计算管理平台的单点登录方法很简单，包括两个步骤：

步骤1 调用 H3C CAS CVM 虚拟化管理平台公开的如下 REST API 接口，获取单点登录 Token。

```
/cas/casrs/operator/getAuthUrl?clientId=XXX.XXX.XXX.XXX
```

其中，“?clientId=XXX.XXX.XXX.XXX”参数可选，表示允许使用 Token 的客户端 IP 地址，当前只允许一个客户端 IP 地址使用当前 Token。

上述 REST API 接口的 HTTP 响应为一个 XML 格式的文本，形如：

```
<ticket>
<uri>
spring_check?ticket=eU1sZ11DbVhRN01OVm12Nm1hdE1sc2NMRDVCmNESzNkWEFIcjrR05POGdwd0hJWjM4
SDhhV3cvbmVUWXJZTmVMUzcZajZPa0pZSS9uTlVKY3BuQkRUMDRCL1Q2VEcxUnFpNmhiS3JRclK9
</uri>
</ticket>
```

步骤2 将获取到的 Token 拼接访问 H3C CAS CVM 虚拟化管理平台的 URL 之后，组成一个完整的单点登录 URL，形如：

```
http://XXX.XXX.XXX.XXX:8080/cas/spring_check?ticket=eU1SZ1lDbVhRN01OVm12NmlhdE1sc2NMRDVC
VmNESzNkWEFIcjcrR05POGdwd0hJWjM4SDhhV3cvbmVUWXJZTmVMUzcZajZPa0pZSS9uTlVKY3BuQkRUMDRCL1Q2
VEcxUnFpNmhiS3JRclk9
```

下面是实现对 H3C CAS CVM 虚拟化管理平台单点登录的 Java 示例程序。

```
package com.mycompany.rest;

import java.io.InputStream;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.auth.AuthScope;
import org.apache.http.auth.UsernamePasswordCredentials;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.DefaultHttpClient;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;

public class MyClient {
    public static DefaultHttpClient client;

    public static void main(String[] args) {
        // H3C CAS CVM 虚拟化管理平台访问 URL
        String strCVMURL = "http://192.168.20.55:8080/cas";

        // 获取 Token 的 RESTful URL
        String strAuthURL = "/casrs/operator/getAuthUrl";

        // 获取单点登录 Token
        String strToken = GetSSOToken(strCVMURL + strAuthURL);

        // 拼接单点登录 URL
        String strSSOURL = strCVMURL + "/" + strToken;

        // 用浏览器打开 H3C CAS CVM 虚拟化管理平台
        OpenCVM(strSSOURL);
    }

    // 认证实例
    private static DefaultHttpClient newInstance() {
        if (client == null) {
            client = new DefaultHttpClient();
            client.getCredentialsProvider().setCredentials(
                new AuthScope("192.168.20.55", 8080, "VMC RESTful Web Services"),
                new UsernamePasswordCredentials("admin", "admin"));
        }

        return client;
    }

    // 获取单点登录 Token
    private static String GetSSOToken(String url) {
```



```

String strToken = "";
try {
    DefaultHttpClient client = newInstance();
    HttpGet httpGet = new HttpGet(url);
    HttpResponse response = client.execute(httpGet);
    HttpEntity entity = response.getEntity();
    if (null != entity) {
        InputStream in = entity.getContent();

        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();
        Document document = builder.parse(in);
        document.getDocumentElement().normalize();

        Element rootElement = document.getDocumentElement();
        NodeList uriNode = rootElement.getElementsByTagName("uri");
        Element element = (Element) uriNode.item(0);
        strToken = element.getTextContent();
    }
} catch (Exception ex) {
    ex.printStackTrace();
}

return strToken;
}

// 用单点登录 URL 打开 H3C CAS CVM 虚拟化管理平台
private static void OpenCVM(String url) {
    java.net.URI uri = java.net.URI.create(url);
    java.awt.Desktop desktop = java.awt.Desktop.getDesktop();
    if (desktop.isSupported(java.awt.Desktop.Action.BROWSE)) {
        try {
            desktop.browse(uri);
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
}

```