# 监听器和邮箱服务

**【学习目标】理解、了解、应用、记忆**

通过今天的学习，参训学员能够：（解释的时候说出二级目标的掌握程度）

1. **【理解】能够阐述监听器Listener的作用**
2. **【理解】了解监听三大域对象的创建与销毁的监听器**
3. **【理解】了解监听三大域对象的属性变化的监听器**
4. **【应用】了解与session中的绑定的对象相关的监听器**
5. **【理解】能够阐述邮箱服务器的基本概念**
6. **【理解】能够阐述邮件的发送过程**
7. **【应用】能够独立编写邮件发送代码**

# 监听器介绍

## 什么是监听器

生活中的例子：

班主任，根据学员上课的状态，做出相应的处理（睡觉了，扣分，玩手机，扣分没收， 发呆，扣分）

监听器：监听事件源，根据事件源上发生事件，做出相应的处理。

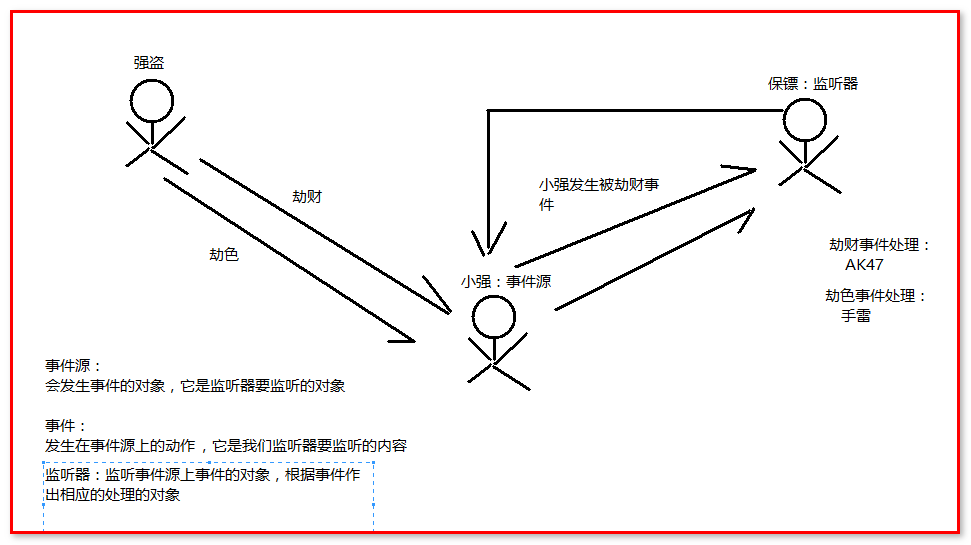


## 监听机制相关概念

事件源：发生事件的源头，监听器需要监听的对象。

事件：事件源上发生的动作，监听器监听的内容。

监听器：负责监听事件源的对象。



# web监听器介绍

## javaweb监听器介绍

JavaWEB中的监听器主要监听JavaWEB中的request、session、ServletContext对象的各种变化。

主要监听的任务：

监听request、ServletContext 、session对象的创建和销毁 （练习）

ServletRequestListener

ServletContextListener

HttpSessionListener

监听request、session、ServletContext 对象存放的数据变化情况（练习）

ServletContextAttributeListener

HttpSessionAttributeListener

ServletRequestAttributeListener

监听session中保存的JavaBean的状态

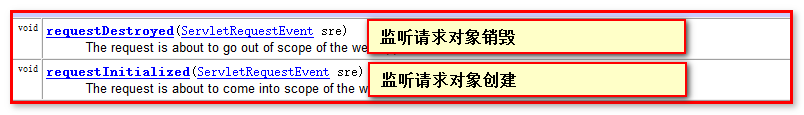
HttpSessionBindingListener

## javaweb监听器创建步骤（示例：ServletRequestListener）

### 需要定义一个类实现对应的监听器接口

ServletRequestListener定义（API截图）：





代码演示：

**package** cn.igeek.listener;

**import** javax.servlet.ServletRequestEvent;

**import** javax.servlet.ServletRequestListener;

**public** **class** MyServletRequestListener **implements** ServletRequestListener{

@Override

//监听request对象销毁的方法

**public** **void** requestDestroyed(ServletRequestEvent sre) {

System.*out*.println("MyServletRequestListener.....requestDestroyed....");

}

@Override

//监听request对象初始化的方法

**public** **void** requestInitialized(ServletRequestEvent sre) {

System.*out*.println("MyServletRequestListener.....requestInitialized....");

}

}



### 配置监听器对象



注意：当服务*器加载项*目的时候，会读取web.xml文件中listener标签，那么服务器会自动创建监听器对象，并且自动调用其方法

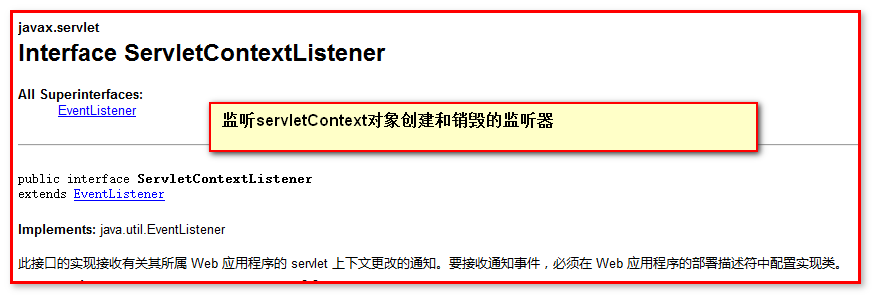
自动调用其方法，也是通过反射调用（因为他的方法名称和参数是固定的）

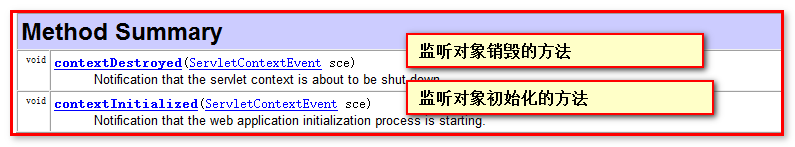
监听器的小结：

* + 1. 创建一个类，实现监听器接口
    2. 在监听器对象的方法中，书写相关的代码
    3. 在web.xml中配置当前监听器。省略

## ServletContext创建销毁监听

ServletContextListener定义（API截图）：





代码演示：

**package** cn.igeek.listener;

**import** javax.servlet.ServletContextEvent;

**import** javax.servlet.ServletContextListener;

**public** **class** MyServletContextListener **implements** ServletContextListener{

@Override

**public** **void** contextInitialized(ServletContextEvent sce) {

System.*out*.println("MyServletContextListener.....contextInitialized....");

}

@Override

**public** **void** contextDestroyed(ServletContextEvent sce) {

System.*out*.println("MyServletContextListener.....contextDestroyed....");

}

}

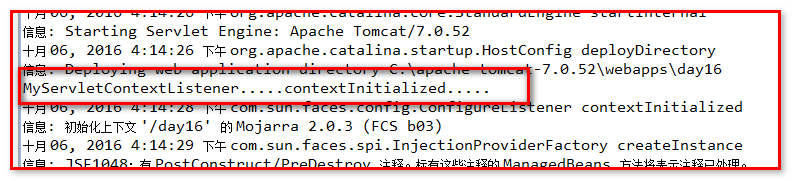
监听器配置：

<listener>

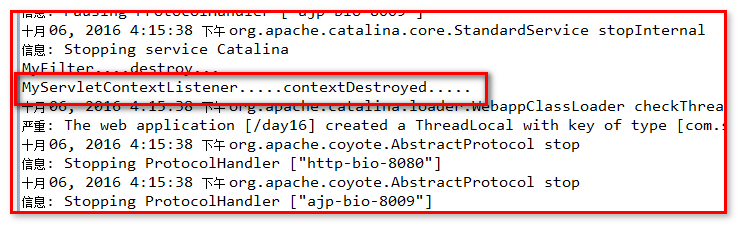
<listener-class>cn.igeek.listener.MyServletContextListener</listener-class>

</listener>

监听servletcontext对象初始化截图：



监听servletcontext对象销毁截图：



## ServletContextListener案例：定时任务

需求：项目启动时，获取服务器时间（new Date()），每一秒钟更新一次，打印在控制台

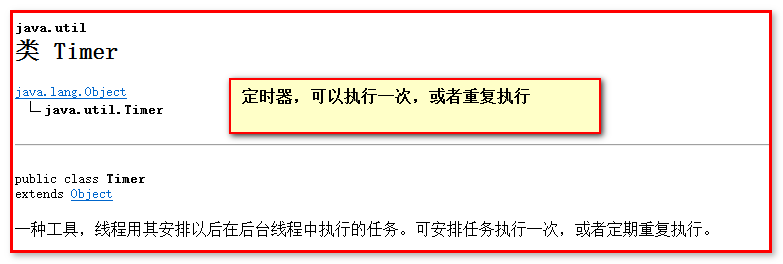
思路：

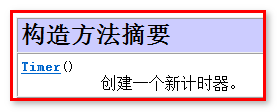
1）监控项目的启动（使用ServletContextListener来监听ServletContext对象的初始化）

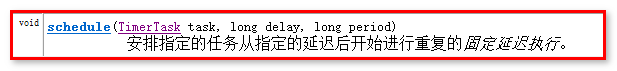
1. 获取服务器时间：new Date（）；
2. 每一秒更新一次：定时器Timer

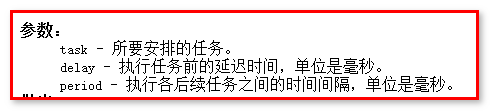
4）给定时器设置定时任务

Timer：定时器 任务调度的对象







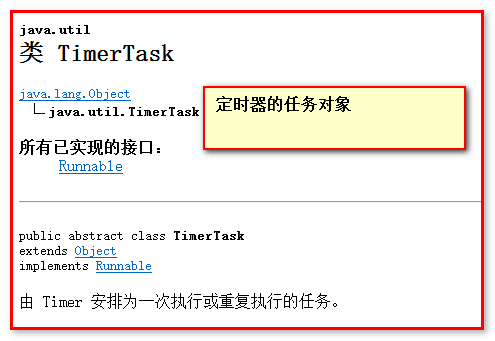


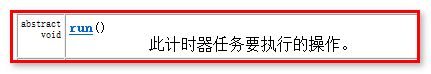
timeTask：定时器的任务（类）

delay：从什么时候开始执行，立即执行设置为：0

period ：间隔多少时间重复执行，毫秒值，1秒=1000毫秒

TimerTask：定时器的任务（类）





Run方法中应该写我们的定时任务：每一秒钟更新一次时间，打印在控制台上

代码实现：

**package** cn.igeek.listener;

**import** java.util.Date;

**import** java.util.Timer;

**import** java.util.TimerTask;

**import** javax.servlet.ServletContextEvent;

**import** javax.servlet.ServletContextListener;

/\*\*

\* **@author** wjn

\* 1） 创建一个类，实现监听器接口

2） 在监听器对象的方法中，书写相关的代码

3） 在web.xml中配置当前监听器。

\*/

**public** **class** MyServletContextListener **implements** ServletContextListener{

@Override

**public** **void** contextInitialized(ServletContextEvent sce) {

System.*out*.println("MyServletContextListener....contextInitialized...");

//监控项目的启动（使用ServletContextListener来监听ServletContext对象的初始化）

//2) 获取服务器时间：new Date（）；

//3） 每一秒更新一次：定时器Timer

//4） 给定时器设置定时任务

//获取定时器

Timer timer = **new** Timer();

//调用定时器的设置定时任务的方法

//firstTime 0:立即执行

//period：间隔多长时间执行一次，1000

timer.schedule(**new** TimerTask() {

@Override

**public** **void** run() {

//在run方法中，书写，要执行的任务

//过时的方法一般不推荐使用，但是，过时的方法，jdk不会删除它的效果。

//当前显示时间，可以使用服务器中的时间——java代码，new Date();

//当前显示时间——javascript代码，new Date();

//javascript代码，是在浏览器运行，客户端的时间，一般是不使用客户端的时间

//业务：整点秒杀

//获取的是服务器时间，用户，是没有办法控制

//获取客户端时间，时间有客户控制，时间是不对的

//一般尊循的原则，只要可以控制在服务器的，绝对不给客户端

System.*out*.println(**new** Date().~~toLocaleString~~());

}

}, 0, 1000);

}

@Override

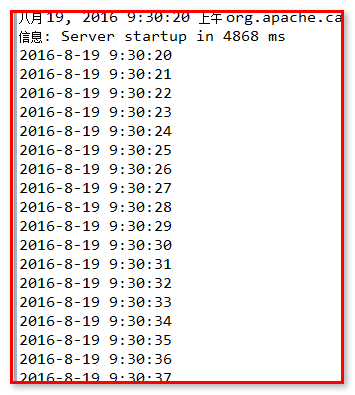
**public** **void** contextDestroyed(ServletContextEvent sce) {

System.*out*.println("MyServletContextListener....contextDestroyed...");

}

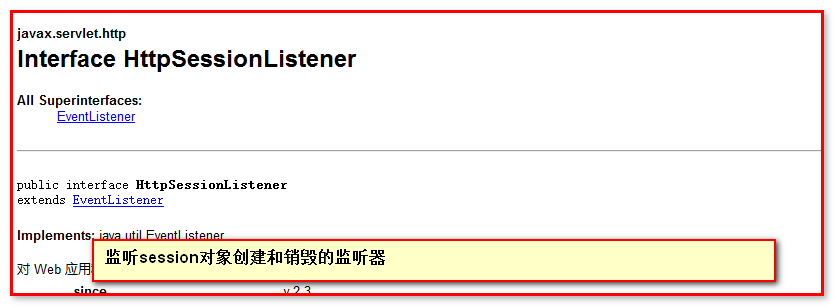
}

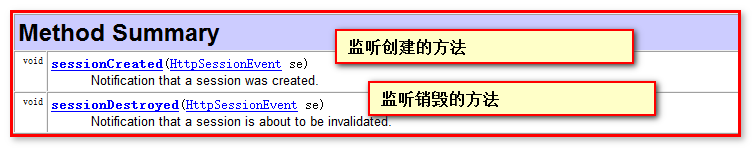
**效果：**



## HttpSessionListener对象监听session的创建与销毁监听

HttpSessionListener定义（API截图）：





代码演示：

**package** cn.igeek.listener;

**import** javax.servlet.http.HttpSessionEvent;

**import** javax.servlet.http.HttpSessionListener;

**public** **class** MyHttpSessionListener **implements** HttpSessionListener{

@Override

**public** **void** sessionCreated(HttpSessionEvent se) {

System.*out*.println("MyHttpSessionListener....sessionCreated....");

}

@Override

**public** **void** sessionDestroyed(HttpSessionEvent se) {

System.*out*.println("MyHttpSessionListener....sessionDestroyed....");

}

}

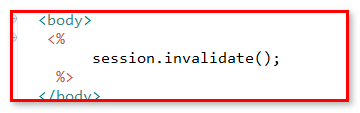
配置文件：

<listener>

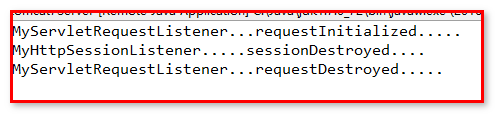
<listener-class>cn.igeek.listener.MyHttpSessionListener</listener-class>

</listener>

Invalidate.jsp页面代码：



效果截图：



## 案例——统计在线人数

用户积累：优惠，折扣，广告，扫码关注，想所有QQ用推送一条消息，给所有支付宝用户发送消息。

第三方登录，QQ账号，微博账号，微信账号，优酷账号，支付宝账号，银行账户，百度账号

1. 用户体验非常好
2. 双赢，创业公司，除了积累用户以外，还获取了用户的QQ或者支付宝，或者微信，可以使用现成推广渠道，再次推广自己应用

对于QQ，它收取费用。

需求：统计当前访问网站的人数有多少人？

问题：什么时候我们可以知道用户访问了网站？

只要用户访问了我们的网站，session一定会创建。只要用户离开，点退出（注销），session就销毁。

思路：

只要判断session创建，在线人数就加一

只要判断session销毁，在线人数就减一

在线人数的数据，要存在哪里？

答：ServletContext对象中，所有应用程序范围都可以获取，所有访问当前网站的用户，都应该可以看到在线人数

总思路：

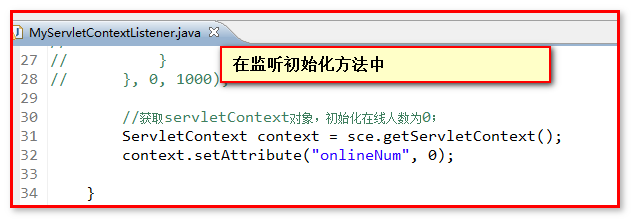
１　现在项目启动的时候，初始化在线人数0，保存在ServletContext

2 监听到session创建数据加一

3 监听到session销毁数据减一

4 在页面展示数据

代码实现：



监听器代码：

**package** cn.igeek.listener;

**import** javax.servlet.ServletContext;

**import** javax.servlet.http.HttpSessionEvent;

**import** javax.servlet.http.HttpSessionListener;

**public** **class** MyHttpSessionListener **implements** HttpSessionListener {

@Override

**public** **void** sessionCreated(HttpSessionEvent se) {

System.*out*.println("MyHttpSessionListener....sessionCreated....");

// 在监听器中只要判断session创建，在线人数就加一

ServletContext context = se.getSession().getServletContext();

// 获取里面的在线人数

Integer onlineNum = (Integer) context.getAttribute("onlineNum");

onlineNum = onlineNum + 1;

context.setAttribute("onlineNum", onlineNum);

}

@Override

**public** **void** sessionDestroyed(HttpSessionEvent se) {

System.*out*.println("MyHttpSessionListener....sessionDestroyed....");

// 在监听器中只要判断session销毁，在线人数就减去一

ServletContext context = se.getSession().getServletContext();

// 获取里面的在线人数

Integer onlineNum = (Integer) context.getAttribute("onlineNum");

onlineNum = onlineNum - 1;

context.setAttribute("onlineNum", onlineNum);

}

}

index.jsp显示在线人数，显示退出链接:

<%@ page language=*"java"* import=*"java.util.\*"* pageEncoding=*"UTF-8"*%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>My JSP 'index.jsp' starting page</title>

<meta http-equiv=*"pragma"* content=*"no-cache"*>

<meta http-equiv=*"cache-control"* content=*"no-cache"*>

<meta http-equiv=*"expires"* content=*"0"*>

<meta http-equiv=*"keywords"* content=*"keyword1,keyword2,keyword3"*>

<meta http-equiv=*"description"* content=*"This is my page"*>

<!--

<link rel="stylesheet" type="text/css" href="styles.css">

-->

</head>

<body>

igeek-filter2项目主页<br>

当前在线人数：${onlineNum }

<a href=*"*${pageContext.request.contextPath }*/validate.jsp"*>退出</a>

</body>

</html>

页面效果：



# 属性变化的监听

## 属性监听器介绍

主要是监听使用setAttribute、removeAttribute方法。

ServletContextAttributeListener 专门用于监听ServletContext对象中的属性的变化情况

HttpSessionAttributeListener 专门用于监听session对象中的属性的变化情况

ServletRequestAttributeListener 专门用于监听request对象中的属性的变化情况

它们中的的监听 添加 、删除 、 修改的方法名称全部一致：



代码演示：

Jsp：

<%

session.setAttribute("name", "毛爷爷");

session.setAttribute("name", "邓爷爷");

session.removeAttribute("name");

%>

监听器：

**package** cn.igeek.listener;

**import** javax.servlet.http.HttpSessionAttributeListener;

**import** javax.servlet.http.HttpSessionBindingEvent;

**public** **class** MyHttpSessionAttributeListener **implements** HttpSessionAttributeListener {

@Override

**public** **void** attributeAdded(HttpSessionBindingEvent se) {

System.*out*.println("MyHttpSessionAttributeListener....attributeAdded...");

}

@Override

**public** **void** attributeRemoved(HttpSessionBindingEvent se) {

System.*out*.println("MyHttpSessionAttributeListener....attributeRemoved...");

}

@Override

**public** **void** attributeReplaced(HttpSessionBindingEvent se) {

System.*out*.println("MyHttpSessionAttributeListener....attributeReplaced...");

}

}

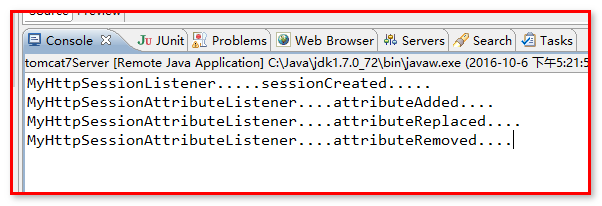
配置文件：

<listener>

<listener-class>cn.igeek.listener.MyHttpSessionAttributeListener</listener-class>

</listener>

效果：



# Session中的bean(JavaBean)监听

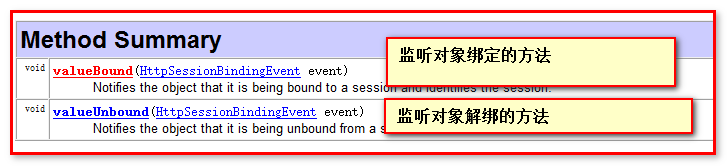
当我们给Session中保存一个Java对象的时候，或者把Java对象从Session中移除的时候会触发专门用来监听Session中对象变化的监听器中的方法。拥有这个方法的对象——HttpSessionBindingListener接口

属性监听和bean监听的区别：

属性监听：是对三个容器中的任何属性（包括对象和不是对象的数据，基本类型数据）的变化，进行监听

Bean监听：它只监听javabean对象往session中保存和session中移出的过程。





由于HttpSessionBindingListener是用来监听某个JavaBean对象的绑定（存入session容器）和解绑（从session容器中移除）的，所以这个监听器的实现类必须是被操作的JavaBean（HttpSessionBindingListener不需要再web.xml中配置）

javaBean：

**package** cn.igeek.domain;

**import** javax.servlet.http.HttpSessionBindingEvent;

**import** javax.servlet.http.HttpSessionBindingListener;

**public** **class** User **implements** HttpSessionBindingListener{

**private** **int** age;

**private** String name;

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

@Override

**public** String toString() {

**return** "User [age=" + age + ", name=" + name + "]";

}

@Override

**public** **void** valueBound(HttpSessionBindingEvent event) {

System.*out*.println("User....valueBound...");

}

@Override

**public** **void** valueUnbound(HttpSessionBindingEvent event) {

System.*out*.println("User....valueUnbound...");

}

}

JSP：

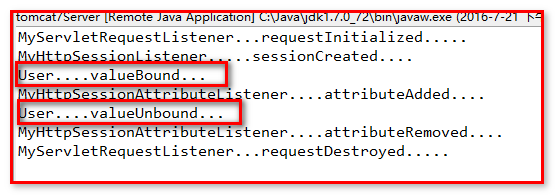
<%

session.setAttribute("user", **new** User());

session.removeAttribute("user");

%>

效果：



**Bean监听需求：**

**在线人数，根据session创建和销毁，来做人数的增减。**

**在线会员统计：**

1. **User类实现bean监听接口**
2. **每次监听到loginUser对象被绑定到session中的时候，会员人数加一**
3. **每次监听到loginUser对象被解绑的时候，会员人数减一**

总结：

1 熟悉监听器相关的概念（事件源 事件 监听器）

2 掌握监听器的创建步骤

3 完成笔记中两个案例（定时任务 在线人数）

服务器的启动时间动态获取如何做？

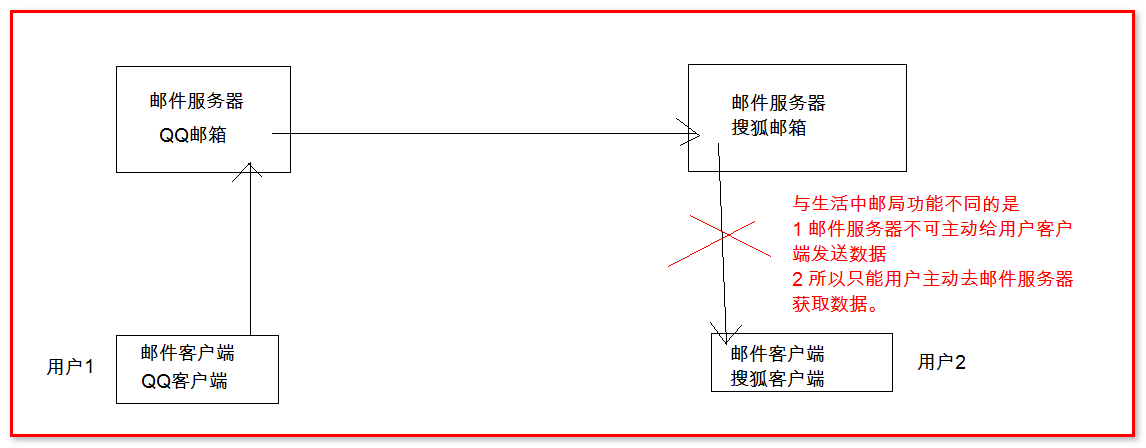
# 电子邮件学习

## 一、电子邮件运行机制？

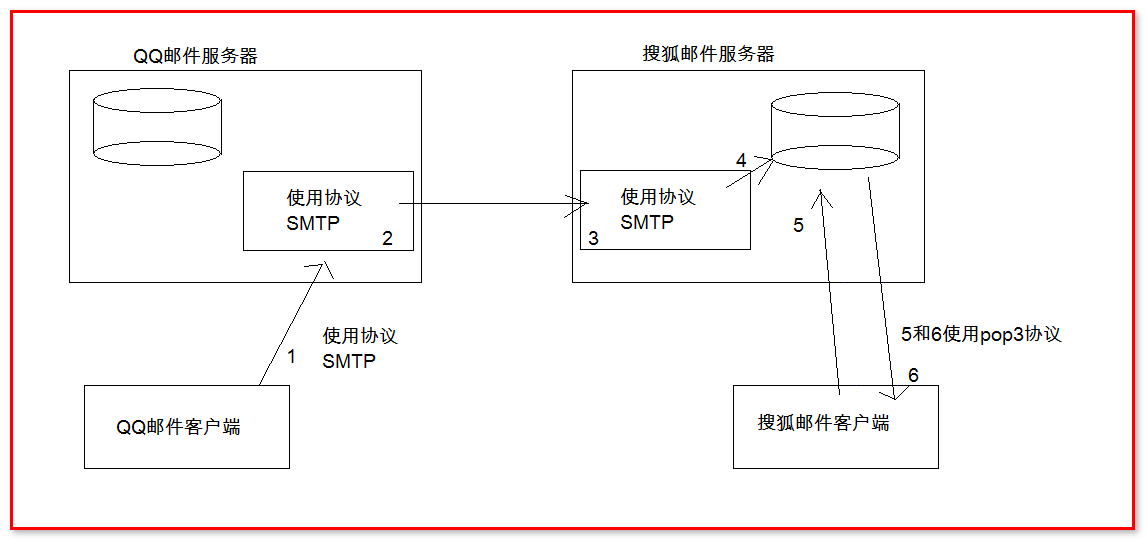
平时生活中写信：

用户1——————》邮局——————————》邮局——————》用户2

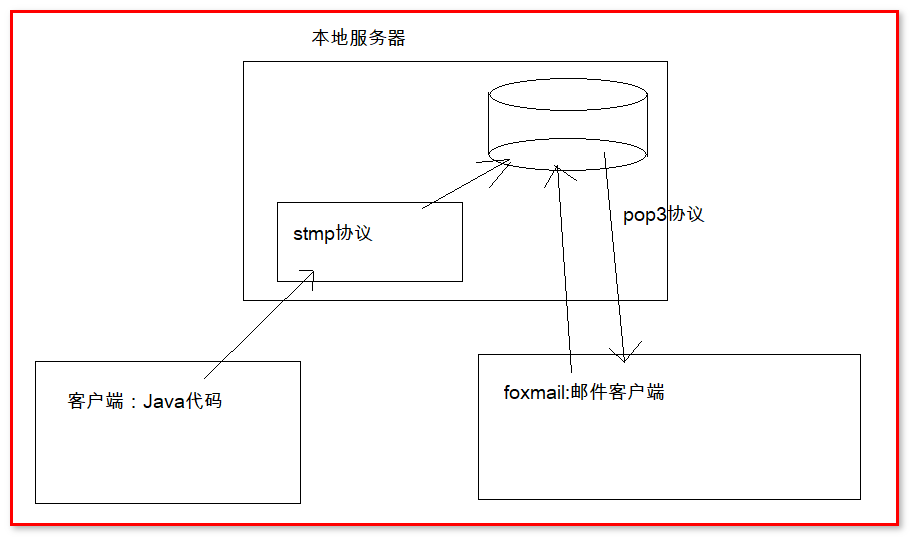
### 错误的常识：



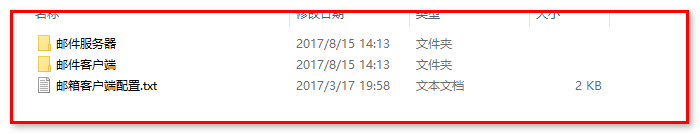
### 正确的原理：



今天的邮件服务器逻辑：



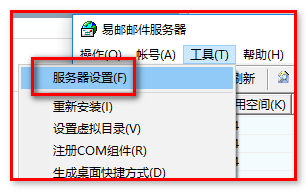
## 二、邮件开发的环境搭建

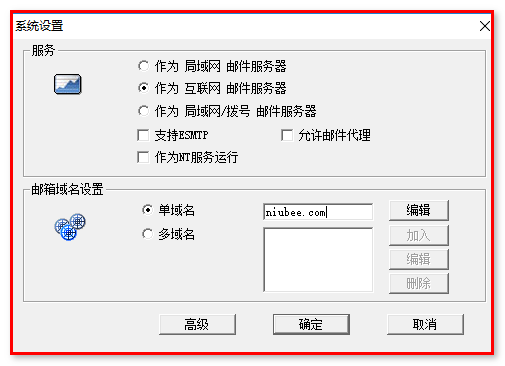


服务器安装：

全部下一步即可（可能出现提示无法创建虚拟目录，用户无法使用web浏览器访问——没关系，我们使用客户端访问）

设置服务器：



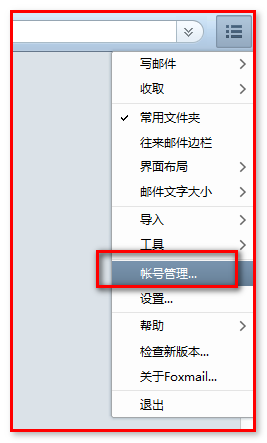


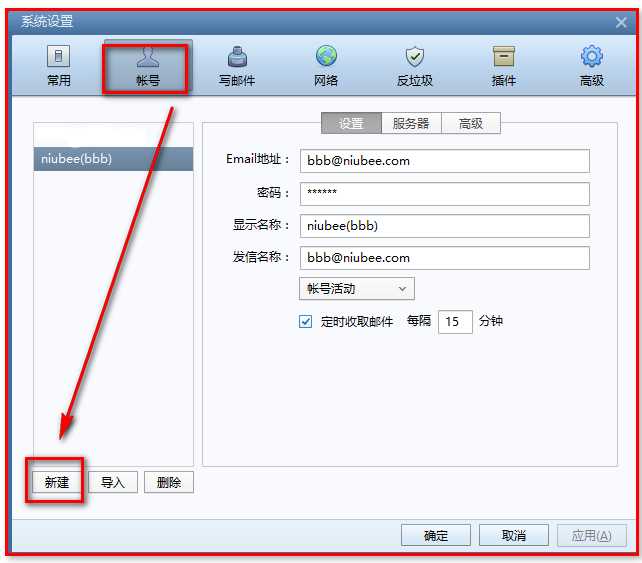
设置用户：按照下图设置aaa和bbb两个账户

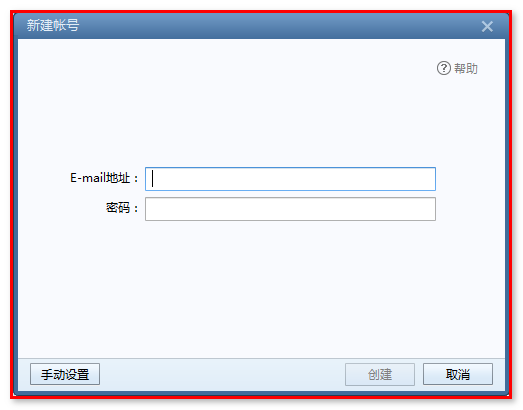


客户端安装（foxmail）：默认快速安装

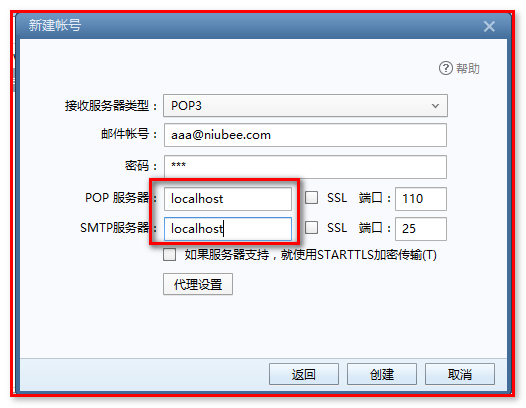
用户设置：







会验证一会儿。。。



收到默认的邮件：



搭建完成！！！！

开发需要的软件——开发环境

## 三、学习使用Javamail发送电子邮件

学习的套路：

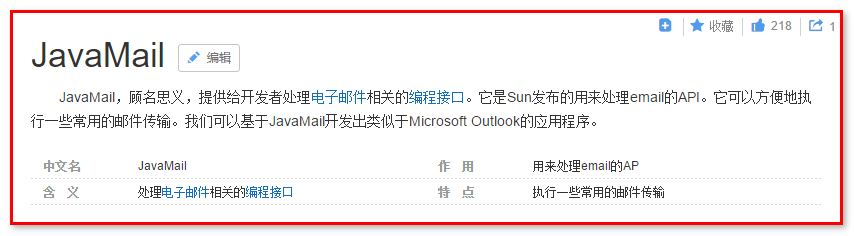
1 百度（官方、论坛、 百度知道、 csdn。。。。）

2 下载（jar包和API文档）

3 查看API，根据API文档测试技术

4 创建工程测试

5 记录笔记

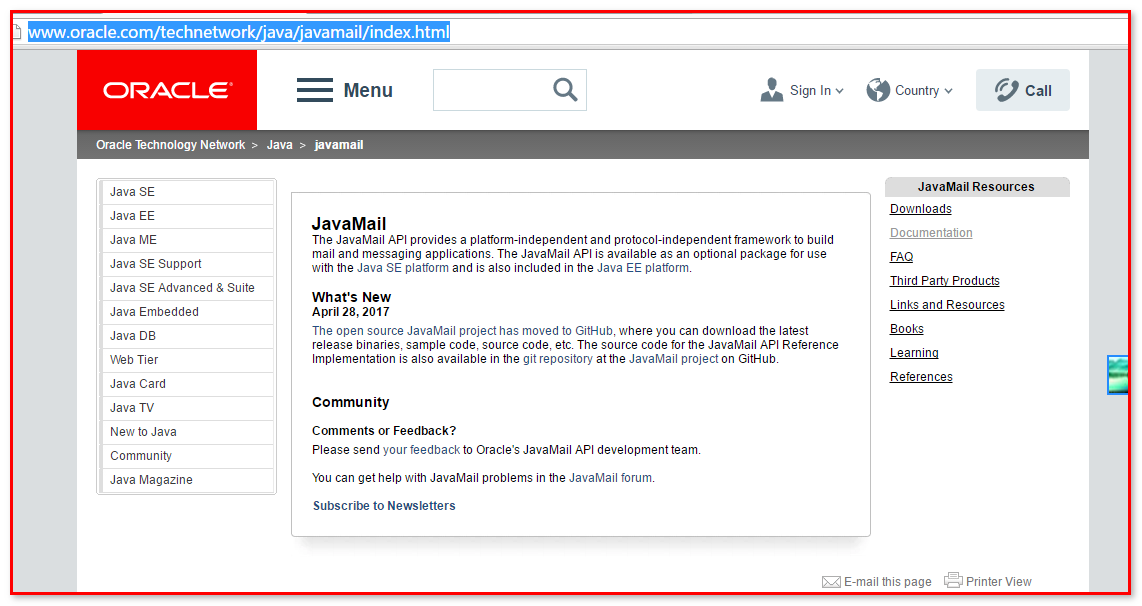


## 四、快速入门

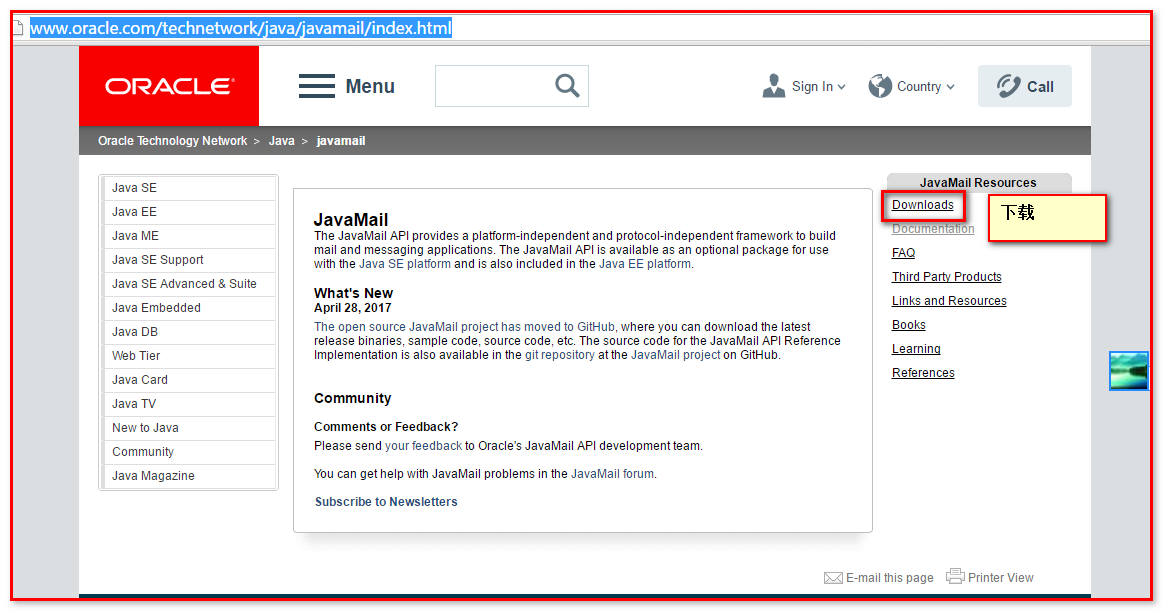


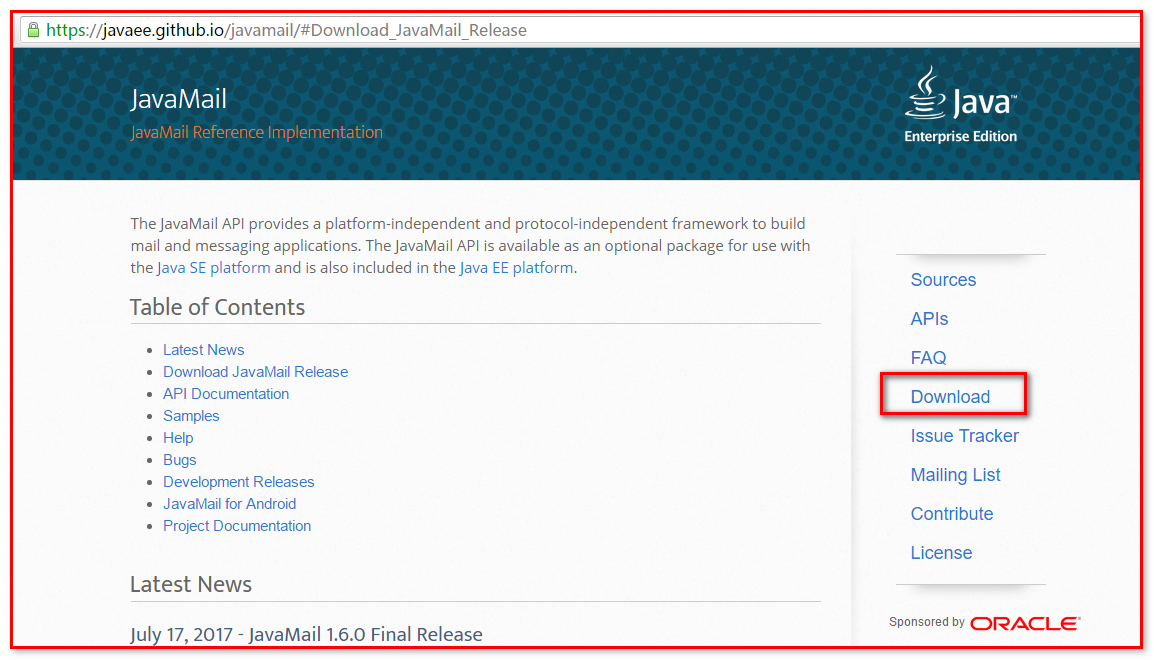
官网：

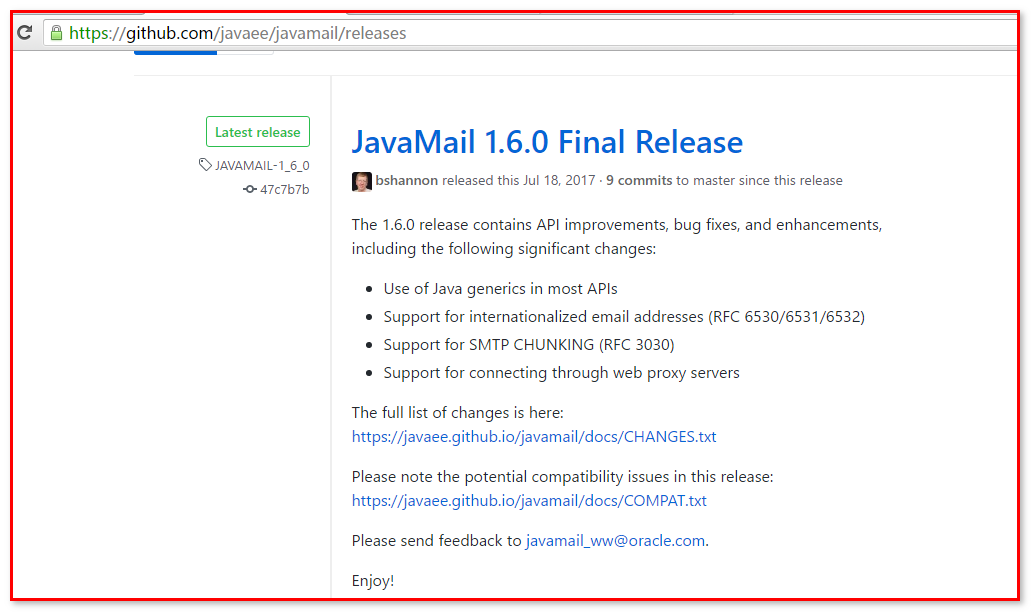
<http://www.oracle.com/technetwork/java/javamail/index.html>



下载：

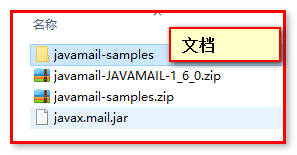


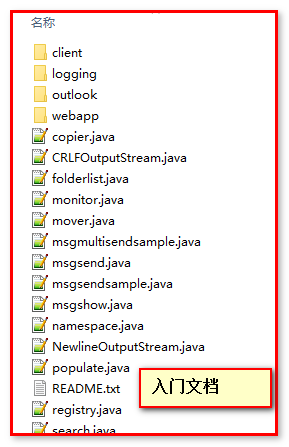




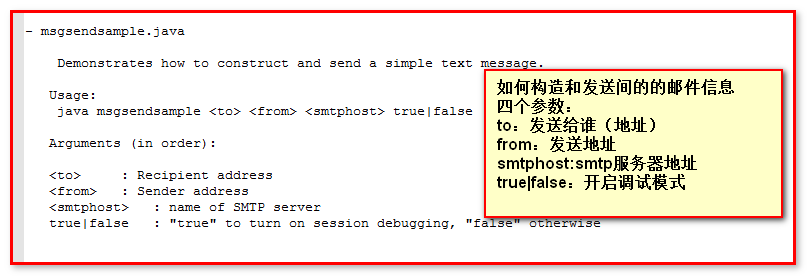


用户指南：





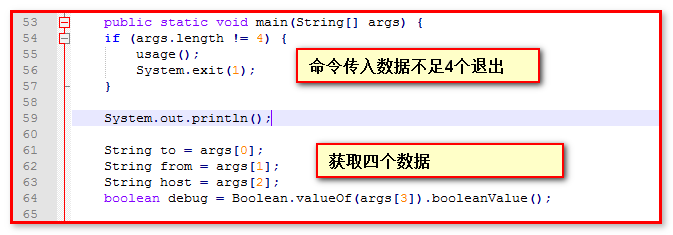
README.txt

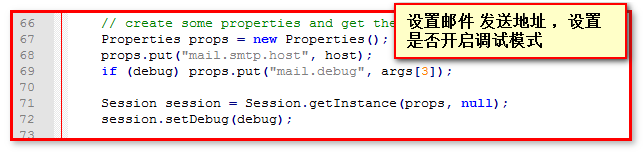


官方源码：

|  |
| --- |
| /\*  \* Copyright (c) 1996-2011 Oracle and/or its affiliates. All rights reserved.  \*  \* Redistribution and use in source and binary forms, with or without  \* modification, are permitted provided that the following conditions  \* are met:  \*  \* - Redistributions of source code must retain the above copyright  \* notice, this list of conditions and the following disclaimer.  \*  \* - Redistributions in binary form must reproduce the above copyright  \* notice, this list of conditions and the following disclaimer in the  \* documentation and/or other materials provided with the distribution.  \*  \* - Neither the name of Oracle nor the names of its  \* contributors may be used to endorse or promote products derived  \* from this software without specific prior written permission.  \*  \* THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS  \* IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,  \* THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR  \* PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR  \* CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,  \* EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,  \* PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR  \* PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF  \* LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING  \* NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS  \* SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.  \*/  import java.util.\*;  import java.io.\*;  import javax.mail.\*;  import javax.mail.internet.\*;  import javax.activation.\*;  /\*\*  \* msgsendsample creates a very simple text/plain message and sends it.  \* <p>  \* usage: <code>java msgsendsample <i>to from smtphost true|false</i></code>  \* where <i>to</i> and <i>from</i> are the destination and  \* origin email addresses, respectively, and <i>smtphost</i>  \* is the hostname of the machine that has the smtp server  \* running. The last parameter either turns on or turns off  \* debugging during sending.  \*  \* @author Max Spivak  \*/  public class msgsendsample {  static String msgText = "This is a message body.\nHere's the second line.";  public static void main(String[] args) {  if (args.length != 4) {  usage();  System.exit(1);  }  System.out.println();    String to = args[0];  String from = args[1];  String host = args[2];  boolean debug = Boolean.valueOf(args[3]).booleanValue();  // create some properties and get the default Session  Properties props = new Properties();  props.put("mail.smtp.host", host);  if (debug) props.put("mail.debug", args[3]);  Session session = Session.getInstance(props, null);  session.setDebug(debug);    try {  // create a message  MimeMessage msg = new MimeMessage(session);  msg.setFrom(new InternetAddress(from));  InternetAddress[] address = {new InternetAddress(to)};  msg.setRecipients(Message.RecipientType.TO, address);  msg.setSubject("JavaMail APIs Test");  msg.setSentDate(new Date());  // If the desired charset is known, you can use  // setText(text, charset)  msg.setText(msgText);    Transport.send(msg);  } catch (MessagingException mex) {  System.out.println("\n--Exception handling in msgsendsample.java");  mex.printStackTrace();  System.out.println();  Exception ex = mex;  do {  if (ex instanceof SendFailedException) {  SendFailedException sfex = (SendFailedException)ex;  Address[] invalid = sfex.getInvalidAddresses();  if (invalid != null) {  System.out.println(" \*\* Invalid Addresses");  for (int i = 0; i < invalid.length; i++)  System.out.println(" " + invalid[i]);  }  Address[] validUnsent = sfex.getValidUnsentAddresses();  if (validUnsent != null) {  System.out.println(" \*\* ValidUnsent Addresses");  for (int i = 0; i < validUnsent.length; i++)  System.out.println(" "+validUnsent[i]);  }  Address[] validSent = sfex.getValidSentAddresses();  if (validSent != null) {  System.out.println(" \*\* ValidSent Addresses");  for (int i = 0; i < validSent.length; i++)  System.out.println(" "+validSent[i]);  }  }  System.out.println();  if (ex instanceof MessagingException)  ex = ((MessagingException)ex).getNextException();  else  ex = null;  } while (ex != null);  }  }  private static void usage() {  System.out.println("usage: java msgsendsample <to> <from> <smtp> true|false");  }  } |

源码阅读和测试：









测试代码：

|  |
| --- |
| @Test  **public** **void** test1() {  String to = "bbb@niubee.com";  String from = "aaa@niubee.com";  String host = "localhost";  **boolean** debug = **false**;  Properties props = **new** Properties();  props.put("mail.smtp.host", host);  props.put("mail.debug", debug);  Session session = Session.*getInstance*(props, **null**);  session.setDebug(debug);  // 创建邮件对象  MimeMessage msg = **new** MimeMessage(session);  **try** {  //设置邮件是谁发送的  msg.setFrom(**new** InternetAddress(from));  //设置邮件发送给谁  InternetAddress[] address = { **new** InternetAddress(to) };  msg.setRecipients(Message.RecipientType.***TO***, address);  //邮件主题  msg.setSubject("JavaMail APIs Test");  //邮件发送日期  msg.setSentDate(**new** Date());  //邮件内容  msg.setText("测试邮件发送");  //发送  Transport.*send*(msg);  } **catch** (Exception e) {  e.printStackTrace();  }  } |

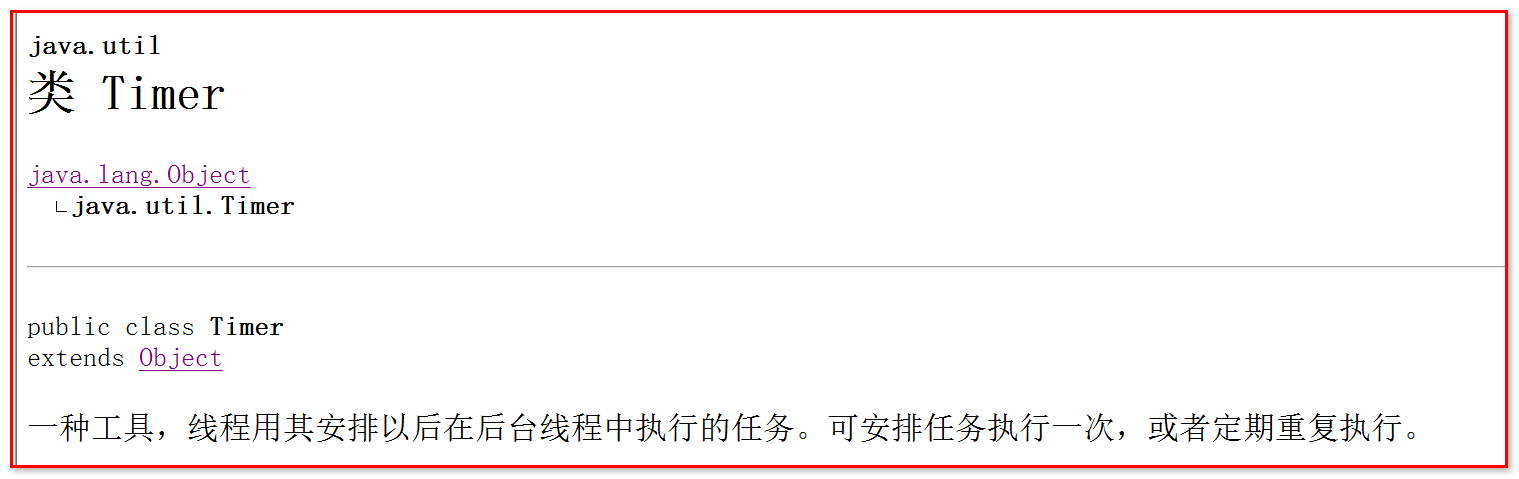
效果：

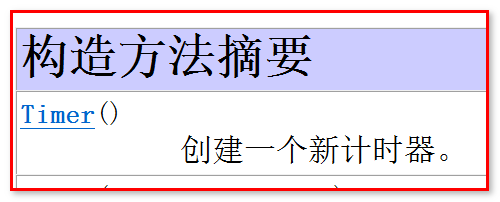


## 五、邮件发送工具类抽取

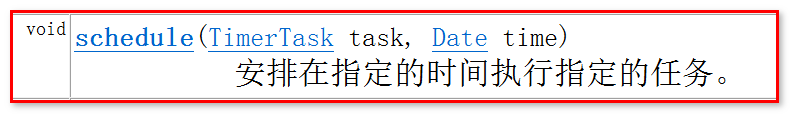
|  |
| --- |
| **package** cn.igeek.domain;  **import** java.util.Date;  **import** java.util.Properties;  **import** javax.mail.Message;  **import** javax.mail.MessagingException;  **import** javax.mail.Session;  **import** javax.mail.Transport;  **import** javax.mail.internet.InternetAddress;  **import** javax.mail.internet.MimeMessage;  **public** **class** MailUtils {  //收邮件的地址 to 主题 subject String text：邮件内容  **public** **static** **void** send(String to ,String subject,String text){    //发邮件的地址  String from = "Admin@niubee.com";  //邮件发送服务器地址  String host = "localhost";  //是否开启debug调试模式  **boolean** debug = **false**;  //设置发送邮件的配置信息  Properties props = **new** Properties();  props.put("mail.smtp.host", host);  **if** (debug) props.put("mail.debug", "false");  //邮件会话  Session session = Session.*getInstance*(props, **null**);  session.setDebug(debug);    **try** {  //创建邮件  MimeMessage msg = **new** MimeMessage(session);  msg.setFrom(**new** InternetAddress(from));  InternetAddress[] address = {**new** InternetAddress(to)};  msg.setRecipients(Message.RecipientType.***TO***, address);  //设置主题  msg.setSubject(subject);  //设置时间  msg.setSentDate(**new** Date());  // 设置邮件的内容  msg.setText(text);  //发送邮件  Transport.*send*(msg);  } **catch** (MessagingException mex) {  mex.printStackTrace();  }  }    **public** **static** **void** main(String[] args) {  //MailUtils.send();  }  } |

六、定时发送邮件(作业)



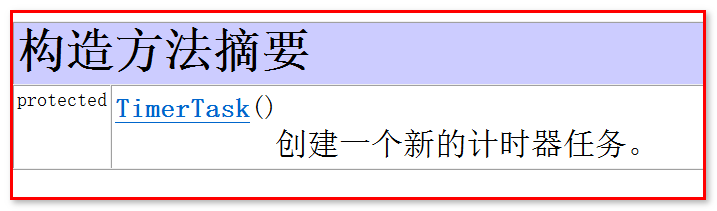


设置定时任务的方法：

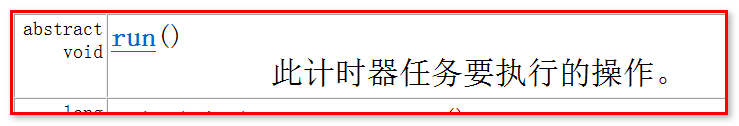


定时任务对象：





执行任务方法：需要执行的任务写在这里



发送邮件的代码：

|  |
| --- |
| **package** ci.igeek.listener;  **import** java.util.Date;  **import** java.util.Timer;  **import** java.util.TimerTask;  **import** javax.servlet.ServletContextEvent;  **import** javax.servlet.ServletContextListener;  **import** ci.igeek.utils.MailUtils;  **public** **class** MailListener **implements** ServletContextListener {  **public** **void** contextInitialized(ServletContextEvent sce) {  Timer timer = **new** Timer();  timer.schedule(**new** TimerTask() {    @Override  **public** **void** run() {  System.***out***.println("准备发送邮件。。。。。");  MailUtils.*sendMail*("aaa@niubee.com", "嘻嘻嘻", "嘎嘎嘎嘎嘎");    System.***out***.println("发送邮件完。。。。。");  }  }, **new** Date(**new** Date().getTime() + 1000\*10));    }    **public** **void** contextDestroyed(ServletContextEvent sce) {  // **TODO** Auto-generated method stub  }    } |