

# 拓展资料

## 第1章 Filter拦截方式

### 1.1 include

当设置过滤器的拦截方式为include的时候，只有当使用include方式转发的请求才能被拦截

#### 1.1.1 include拦截方式代码演示

1.创建IncludeServlet,使用include转发的方式转发到index.jsp去

IncludeServlet代码如下：

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "IncludeServlet", urlPatterns = "/IncludeServlet")
public class IncludeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        System.out.println("include方式转发到index.jsp页面去");
        request.getRequestDispatcher("index.jsp").include(request, response);
    }
}
```

2.创建MethodFilter,配置MethodFilter的拦截方式为include，拦截路径为/\*

MethodFilter代码如下：

```
import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

@WebFilter(filterName = "MethodFilter", dispatcherTypes = DispatcherType.INCLUDE, urlPatterns = "/*")
public class MethodFilter implements Filter {
```

```

    public void destroy() {
    }

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws
    ServletException, IOException {
        System.out.println("+++++++MethodFilter过滤器执行了+++++++");
        chain.doFilter(req, resp);
    }

    public void init(FilterConfig config) throws ServletException {

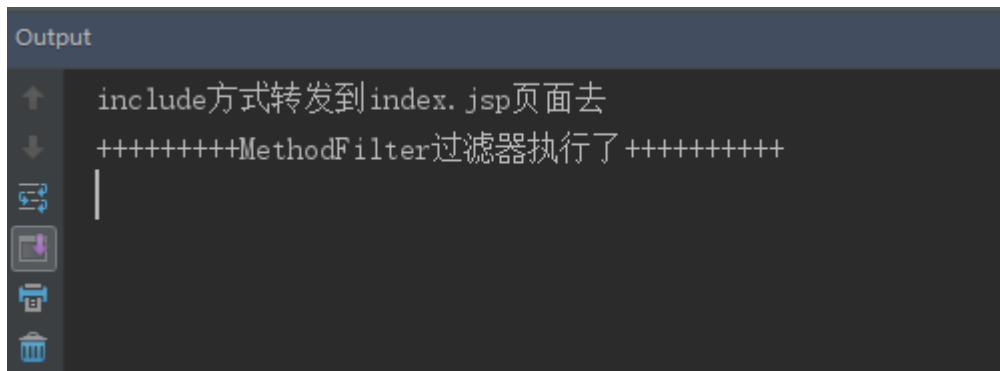
    }

}

```

3.浏览器地址栏输入<http://localhost:9090/IncludeServlet> ,

查看控制台，结果如下：



## 1.2 error

当设置过滤器的拦截方式为error的时候，只有当发生异常的时候请求才能被拦截

### 1.2.1 error拦截方式代码演示

过滤器：

```

package com.itheima.myfilter;

import javax.servlet.*;
import javax.servlet.annotation.WebFilter;
import java.io.IOException;

@WebFilter(filterName = "ErrorFilter",urlPatterns = "/*",dispatcherTypes = DispatcherType.ERROR)
public class ErrorFilter implements Filter {
    public void destroy() {
    }

    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain) throws

```

```

ServletException, IOException {
    System.out.println("拦截异常!!!");
    chain.doFilter(req, resp);
}

public void init(FilterConfig config) throws ServletException {

}

}

```

发生异常的jsp

```

<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<html>
<head>
    <title>Title</title>
</head>
<body>
<% int i = 1/0; %>出现异常!!!
</body>
</html>

```

xml配置：

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">
    <error-page>
        <!--错误响应码-->
        <error-code>500</error-code>
        <!--发生错误之后跳转的页面-->
        <location>/500.jsp</location>
    </error-page>
</web-app>

```

## 第2章 监听器和JavaMail

### 2.1 什么是监听器

在实际生活着，我们很多商场有摄像头，监视着客户的一举一动。如果客户有违法行为，商场可以采取相应的措施。同样，在我们的java程序中，有时也需要监视某些事情，一旦被监视的对象发生相应的变化，我们应该采取相应的操作。这就需要用到java中的监听器。

web监听器是一种Servlet中的特殊的类，它们能帮助开发者监听web中的特定事件，比如ServletContext,HttpSession,ServletRequest的创建和销毁；变量的创建、销毁和修改等。可以在某些动作前后增加处理，实现监控。

## 2.2 监听器的使用场景

### 2.2.1 系统启动时初始化信息

ServletContextListener用来监听ServletContext对象的创建和销毁的。当项目启动的时候，servletContext对象被创建，会调用ServletContextListener的contextInitialized方法。所以我们可以在此方法中初始化项目需要的信息。

### 2.2.2 统计在线人数

我们知道，每当一个用户访问项目的时候，都会创建一个session会话。所以当前session会话被创建，当前在线用户+1，每当session会话被销毁，当前在线用户-1。HttpSessionListener可以用来监听session对象的创建和销毁的。所以可以在HttpSessionListener中的监听session对象创建和销毁的方法中控制在线人数的加减。

### 2.2.3 开启定时任务调度

在实际开发中，我们很多时候需要定时器。那么web项目中的定时器应该随着项目的启动而开始任务调度。ServletContextListener用来监听ServletContext对象的创建和销毁的。当项目启动的时候，servletContext对象被创建，会调用ServletContextListener的contextInitialized方法。所以我们可以在此方法中进行定时器任务的调度。这样，定时任务就随着项目的启动开始调度了。

## 2.3 我的第一个监听器开发步骤

### 2.3.1 ServletContextListener监听器

#### 2.3.1.1 API介绍

ServletContextListener接口

1. `void contextDestroyed(ServletContextEvent sce)` 监听servletcontext销毁
2. `void contextInitialized(ServletContextEvent sce)` 监听servletcontext创建

#### 2.3.1.2 使用步骤

1. 创建一个类实现ServletContextListener接口
2. 给这个类添加注解@WebListener

3.实现ServletContextListener的contextInitialized和contextDestroyed方法。

### 2.3.1.3 案例代码

```
package com.itheima.listener;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class MyServletContextListener implements ServletContextListener {

    @Override
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        //ServletContextEvent 被监听对象的事件源
        //获取被监听的对象
        ServletContext servletContext = servletContextEvent.getServletContext();
        System.out.println("服务器启动, servletContext被创建了");
    }

    @Override
    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        System.out.println("服务器停止, servletContext被销毁了");
    }
}
```

## 2.3.2 ServletContextAttributeListener监听器

### 2.3.2.1 API介绍

ServletContextAttributeListener接口

1. `void attributeAdded(ServletContextAttributeEvent scab)` 监听属性添加到servletcontext中
2. `void attributeRemoved(ServletContextAttributeEvent scab)` 监听属性从servletcontext中移除
3. `void attributeReplaced(ServletContextAttributeEvent scab)` 监听属性从servletcontext中被替换

### 2.3.2.2 使用步骤

- 1.创建一个类实现ServletContextAttributeListener接口

2.给这个类添加注解@WebListener

3.实现ServletContextAttributeListener接口的方法

4.创建一个servlet，doGet方法中分别向servletContext对象中添加、替换、删除属性

### 2.3.2.3 案例代码

MyServletContextAttributeListenerner代码：

```
package com.itheima.listener;

import javax.servlet.ServletContext;
import javax.servlet.ServletContextAttributeEvent;
import javax.servlet.ServletContextAttributeListener;
import javax.servlet.annotation.WebListener;

@WebListener
public class MyServletContextAttributeListenerner implements ServletContextAttributeListener {
    @Override
    public void attributeAdded(ServletContextAttributeEvent servletContextAttributeEvent) {
        //获取被监听的对象
        ServletContext servletContext = servletContextAttributeEvent.getServletContext();
        //获取被添加到servletContext对象中的属性名
        String name = servletContextAttributeEvent.getName();
        //获取被添加到servletContext对象中的属性值
        String value = (String)servletContextAttributeEvent.getValue();
        System.out.println("被添加到servletContext对象中的属性是："+name+"="+value);
    }

    @Override
    public void attributeRemoved(ServletContextAttributeEvent servletContextAttributeEvent) {
        //获取被监听的对象
        ServletContext servletContext = servletContextAttributeEvent.getServletContext();
        //获取被移出servletContext对象中的属性名
        String name = servletContextAttributeEvent.getName();
        //获取被移出servletContext对象中的属性值
        String value = (String)servletContextAttributeEvent.getValue();
        System.out.println("被移出servletContext对象中的属性是："+name+"="+value);
    }

    @Override
    public void attributeReplaced(ServletContextAttributeEvent servletContextAttributeEvent) {
        //获取被监听的对象
        ServletContext servletContext = servletContextAttributeEvent.getServletContext();
        //获取servletContext对象中被替换的属性名
        String name = servletContextAttributeEvent.getName();
        //获取servletContext对象中被替换的属性值
        String value = (String)servletContextAttributeEvent.getValue();
        System.out.println("servletContext对象中的被替换前属性是："+name+"="+value);
    }
}
```

ServletContextAttributeServlet代码：

```
package com.itheima.servlet;

import javax.servlet.ServletContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

@WebServlet(name = "ServletContextAttributeServlet", urlPatterns =
"/ServletContextAttributeServlet")
public class ServletContextAttributeServlet extends HttpServlet {
    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        doGet(request, response);
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {
        ServletContext servletContext = getServletContext();
        //向servletContext对象中添加属性username:zhangsan
        servletContext.setAttribute("username", "zhangsan");
        //替换servletContext对象中的属性username:lisi
        servletContext.setAttribute("username", "lisi");
        //移除servletContext对象中的属性username
        servletContext.removeAttribute("username");
    }
}
```

## 2.4 拓展资料

### 2.4.1 JavaMail

#### 2.4.1.1 什么是JavaMail

在我们开发中，我们很多时候都需要涉及发送邮件的操作，比如：用户注册成功邮件激活账号，会员生日邮件提醒等等。这样的邮件操作就需要学习JavaMail来完成我们的java发送邮件的操作。

JavaMail是Sun公司提供的处理电子邮件的一套编程接口。使用JavaMail我们可以使用java代码完成邮件的接收和发送。

#### 2.4.1.2 JavaMail应用场景

##### 2.4.1.2.1 会员生日时发送邮件

例如qq每当到qq用户的生日的时候，qq会发送一封生日祝福邮件给qq用户。

2.4.1.2.2 会员注册成功，发送邮件，会员激活后，会员才能登陆。

我们现在上网很多时候需要注册一个用户，往往在我们注册后，网站会给我们发送一封激活邮件，需要我们去激活才能进行登录。

2.4.1.3 邮件使用的协议

网易有网易的邮件服务，qq有qq的邮件服务，传智播客有传智播客的邮件服务...那么我们我们播客的邮箱账号可以给网易的邮箱发送邮件，也可以给qq邮箱发送邮件。也就是说网页和qq邮箱服务能够解析我们传智发送的邮件，所以网络中发送邮件应该有与http协议一样概念的邮件协议。

2.4.1.3.1 发邮件协议

发邮件协议使用的是SMTP,协议的端口号是25。

**SMTP**的全称是“Simple Mail Transfer Protocol”，即简单邮件传输协议。它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。SMTP 协议属于 TCP/IP 协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。

2.4.1.3.2 收邮件协议

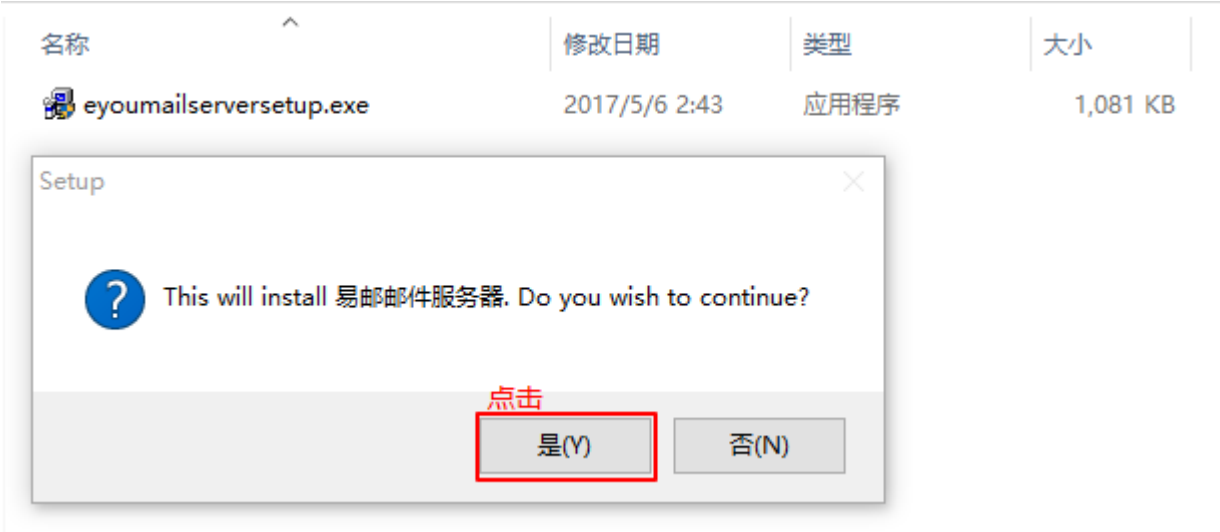
收邮件协议有2中，一种是POP3，还有一种是IMAP。协议的端口号为110。

**POP3**是Post Office Protocol 3的简称，即邮局协议的第3个版本,它规定怎样将个人计算机连接到Internet的邮件服务器和下载电子邮件的电子协议。它是因特网电子邮件的第一个离线协议标准,POP3允许用户从服务器上把邮件存储到本地主机（即自己的计算机）上,同时删除保存在邮件服务器上的邮件。

**IMAP**全称是Internet Mail Access Protocol，即交互式邮件存取协议，它是跟POP3类似邮件访问标准协议之一。不同的是，开启了IMAP后，您在电子邮件客户端收取的邮件仍然保留在服务器上，同时在客户端上的操作都会反馈到服务器上，如：删除邮件，标记已读等，服务器上的邮件也会做相应的动作。所以无论从浏览器登录邮箱或者客户端软件登录邮箱，看到的邮件以及状态都是一致的。

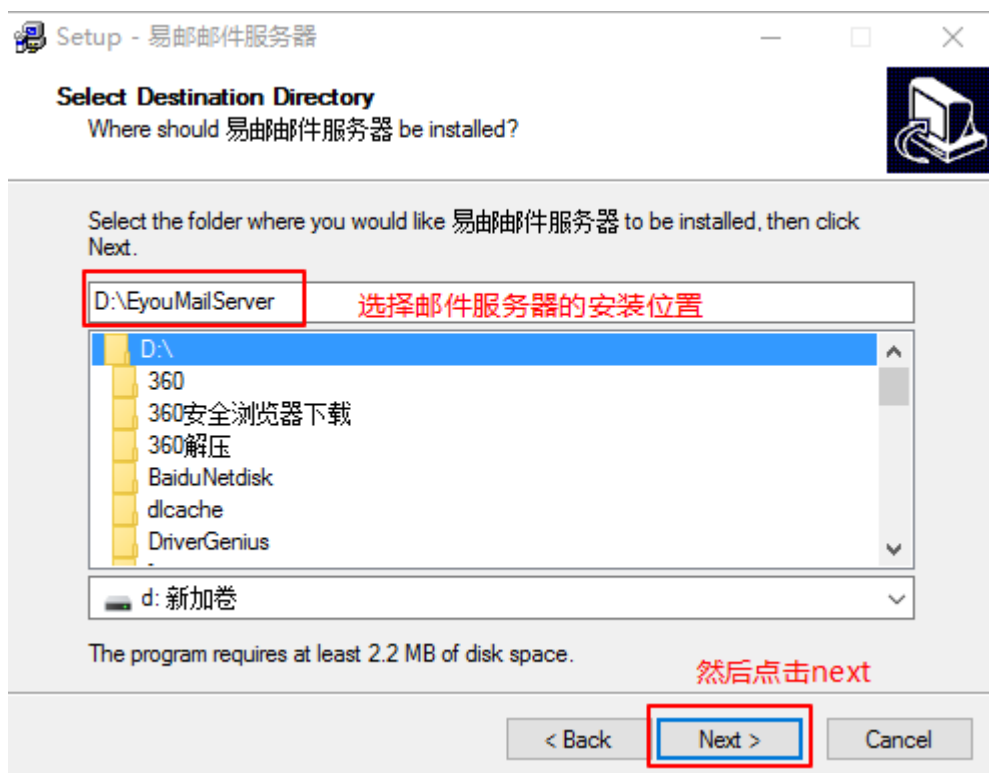
2.4.1.4 邮件服务器安装

1.找到易邮邮件服务器安装包，双击安装

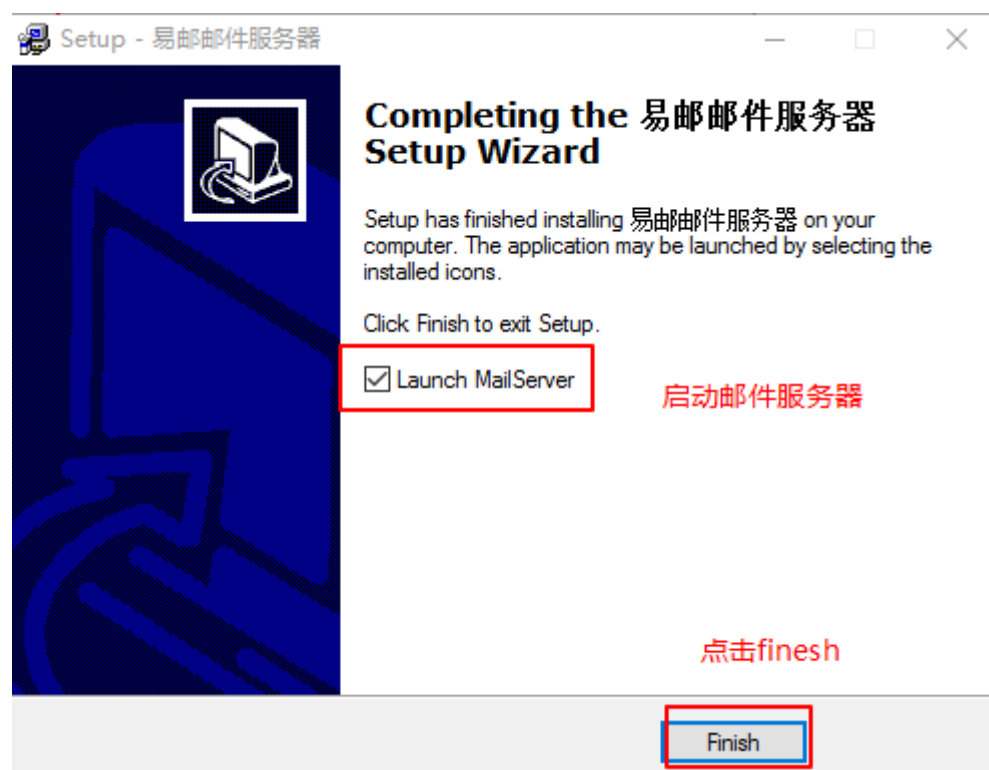


2.选择邮件服务器的安装位置

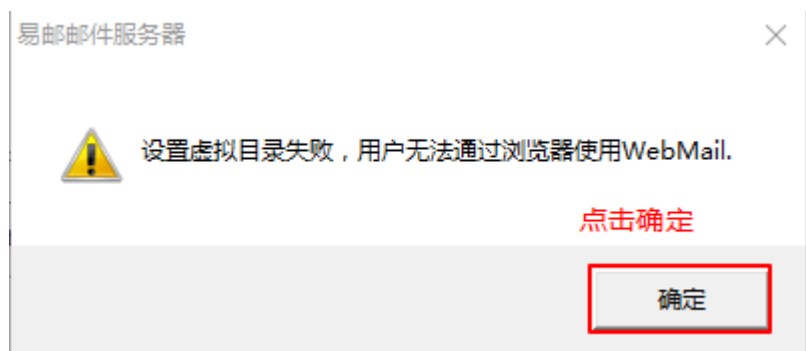




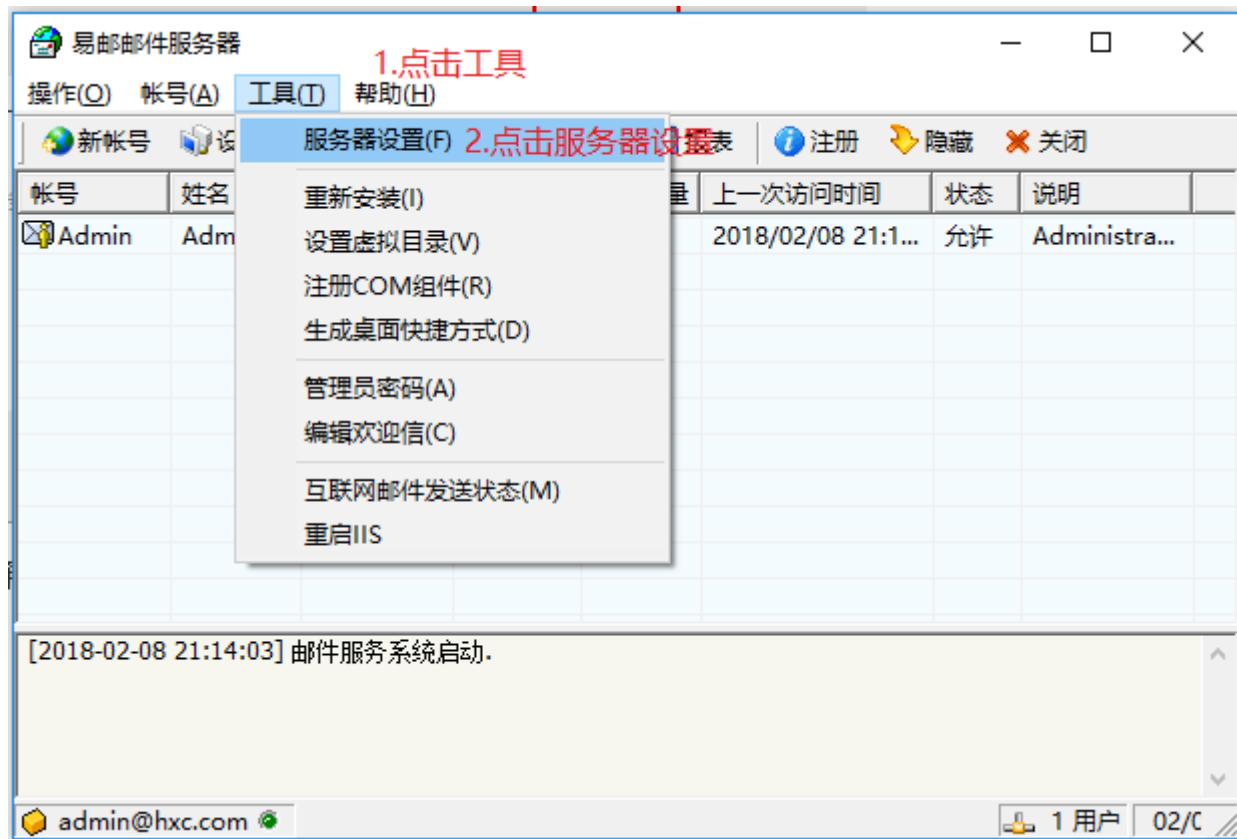
### 3.启动邮件服务器



### 4.出现如下页面，继续点击确定



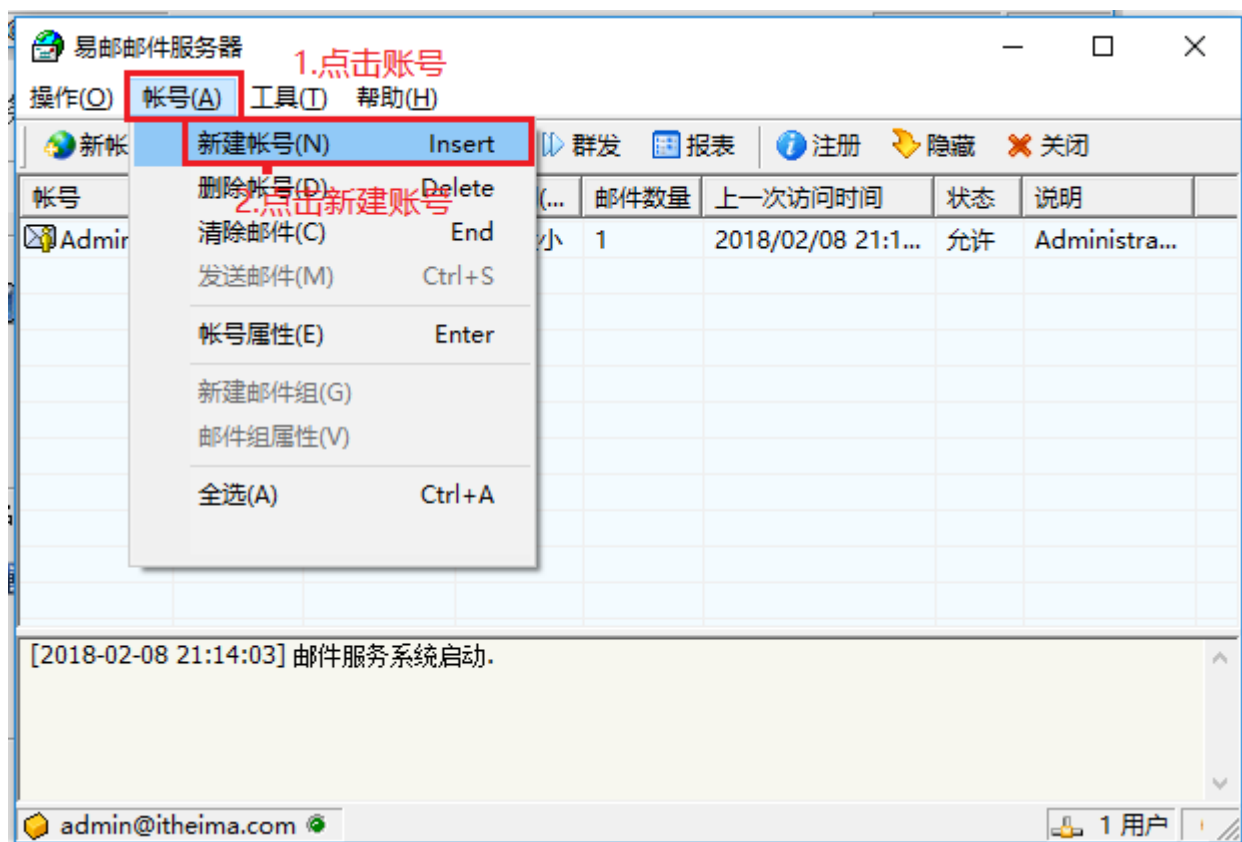
##### 5.设置邮箱服务器



##### 6.设置服务器的域名



7.创建2个账号



8.创建账号zhangsan , 密码为123 和账号lisi,密码123

帐号

基本信息 远程POP3 邮件转发 邮件过滤 自动回复

1.选择基本信息

☐ 禁用

帐号 帐户名 zhangsan

密码 密码 \*\*\*

姓名

说明

邮箱大小 20 (M) ☐ 不限大小

联系邮件地址

☒ 帐号 ☐ 域管理员 ☐ 系统管理员

确定 取消

#### 2.4.1.5 Foxmail安装

1.双击foxmail邮件客户端安装软件

安装 - Foxmail

欢迎使用 Foxmail 6.5 正式版 安装向导

这个向导将指引您完成 Foxmail 6.5 正式版的安装进程。

在开始安装之前，建议先关闭其他所有应用程序。这将允许“安装程序”更新指定的系统文件，而不需要重新启动您的计算机。

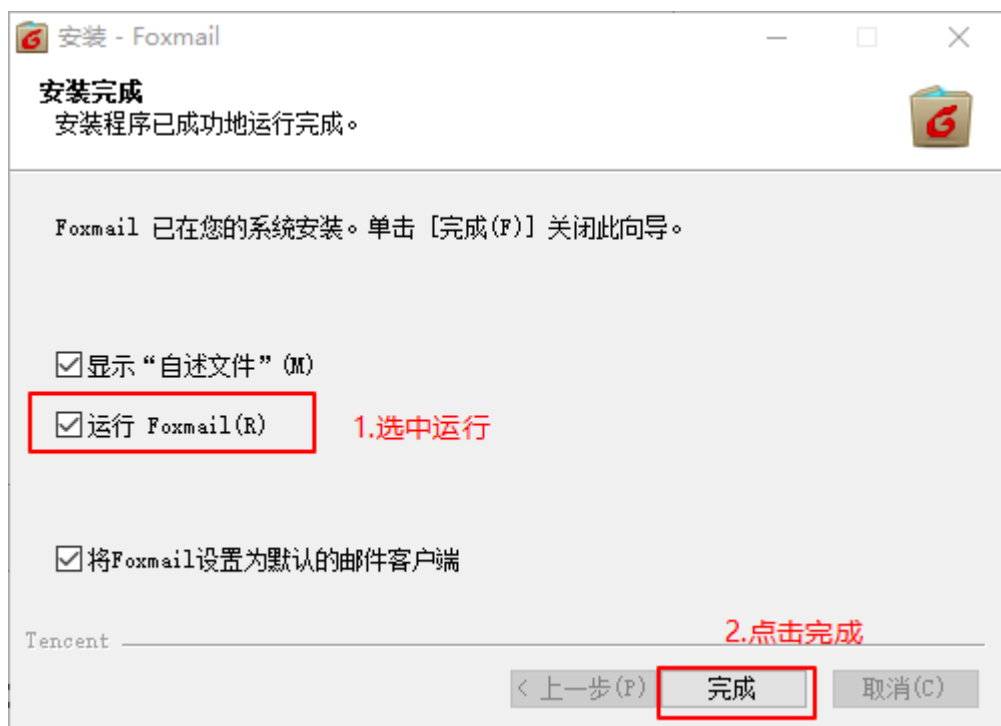
单击 [下一步(N)] 继续。

点击下一步

下一步(N) > 取消(C)

2.然后一直点击下一步，直到安装成功

3.运行foxmail邮箱客户端



#### 4. 设置登录foxmail的账号名和密码

向导

邮箱账号就是邮箱服务器中创建的用户名和邮箱服务器域名的结合

建立新的用户帐户

红色项是您需要填写的。其它选项，如“密码”可在收发邮件时再输入。

[必填] 电子邮件地址(A):  1.设置邮箱的账号

密码(W):  2.填写密码

"帐户名称"是在Foxmail中显示的名称，以区分不同的邮件帐户。"邮件中采用的名称"可填您的姓名或昵称，将包含在发出的邮件中。

[必填] 帐户显示名称(U):

邮件中采用的名称(S):


"邮箱路径"按默认即可。您也可以自行指定邮件的保存路径。

邮箱路径(M):

选择(B)... 默认(D)

< 上一步(B) 下一步(X) > 取消(C) 帮助(H)

#### 5. 设置登录账号的邮箱服务器的位置



### 指定邮件服务器

POP3(PostOffice Protocol 3)服务器是用来接收邮件的服务器，您的邮件保存在其上。如public.guangzhou.gd.cn。

接收服务器类型(T): POP3 1.设置接受邮件服务器地址为localhost

接收邮件服务器(I): localhost

邮件帐户(A): zhangsan

SMTP(Simple Mail Transfer Protocol)服务器用来中转发送您发出的邮件。  
SMTP服务器与POP3服务器可以不同。2.设置发送邮件服务器的地址为localhost

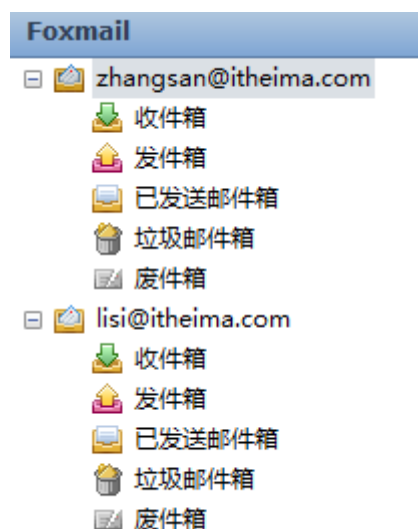
发送邮件服务器(O): localhost

高级(D)...

3.点击下一步

< 上一步(B) 下一步(X) > 取消(C) 帮助(H)

6.同样的设置使用lisi的账号登录foxmail



7.然后使用zhangsan的账号给lisi的账号发送邮件，如果lisi接受成功，安装完成

#### 2.4.1.4 JavaMail常用的API

1. 

```
Session.getInstance(Properties props, Authenticator authenticator);
```

  
创建java到邮件服务器间的会话对象Session  
  
Properties以key 和value的形式保存邮件服务器的设置  
  
例如：设置服务器的地址。  
  

```
props.put("mail.smtp.host", "127.0.0.1");
```

```
Properties props = new Properties();
```
2. 

```
Message message = new MimeMessage(session);
```

  
创建邮件对象。session是java和服务器的会话对象session
3. 

```
message.setFrom(new InternetAddress("admin@abc.com"));
```

  
这只这封邮件的发件人。InternetAddress发件人地址对象。
4. 

```
message.setRecipient(RecipientType.TO, new InternetAddress(toEmail));
```

  
参数一：设置邮件的接受方式。  
  
RecipientType.TO:为正常接受邮件  
  
RecipientType.CC:表示抄送  
  
RecipientType.BCC:表示秘密抄送  
  
参数二：设置邮件的收件人。
5. 

```
message.setSubject("通知");
```

 设置邮件的主题。
6. 

```
message.setContent(emailMsg, "text/html; charset=UTF-8");
```

  
参数一：设置邮件的内容  
  
参数二：设置解析邮件的方式为html，并且以utf-8的编码解析。
7. 

```
Transport.send(message);
```

  
发送邮件。参数就是需要发送的邮件对象Message

#### 2.4.1.5 JavaMail发送邮件步骤

## 1.登录SMTP服务器

1.1 创建Properties对象，以指定主机(mail.smtp.host=localhost)和是否验证 (mail.smtp.auth=true)。

1.2 创建Authenticator抽象类，重写方法getPasswordAuthentication()，返回它的子类：PasswordAuthentication()，指定用户名和密码进行加密。

1.3 创建Session对象，传入Properties和Authenticator的参数

## 2.创建邮件，提供（发件人，收件人，主题，内容）

2.1 创建MimeMessage对象

2.2 设置发件人，与登录的用户名一致，需要使用到InternetAddress类型

2.3 设置收件人，指定收件人的类型：RecipientType.TO

2.4 设置正文和主题，需要指定内容类型和编码

## 3.发送邮件，Transport.send()发送邮件

## 4.封装邮件发送工具类MailUtil

### 2.4.1.6 JavaMail发送邮件工具类代码

```
public class MailUtils {
    //发送邮件
    //mail:收件人  mailMsg:邮件内容  subject:邮件标题
    public static void sendMail(String mail,String mailMsg) throws AddressException,
    MessagingException{

        Properties pro = new Properties();

        pro.setProperty("mail.transport.protocol", "SMTP");//设置邮箱服务器协议
        pro.setProperty("mail.host", "localhost");//设置发邮件的邮箱服务器地址
        pro.setProperty("mail.smtp.auth", "true");// 验证发件箱的账户名和密码
        //创建验证器
        Authenticator auth = new Authenticator(){
            @Override
            protected PasswordAuthentication getPasswordAuthentication() {
                //第一个参数：账户名        第二个参数：密码
                return new PasswordAuthentication("zhangsan", "123");
            }
        };

        //创建一个程序和邮箱服务器的会话对象Session
        Session session = Session.getInstance(pro, auth);

        Message message = new MimeMessage(session);//创建一个message对象，用于设置邮件内容
        message.setFrom(new InternetAddress("zhangsan@itheima.com"));//设置发件人
        message.setRecipient(RecipientType.TO, new InternetAddress(mail));//设置发件方式和接受者
        message.setSubject("邮件主题");//设置标题
        message.setContent(mailMsg, "text/html;charset=utf-8");//设置邮件内容
        //执行发送
        Transport.send(message);
    }
}
```



```
}

public static void main(String[] args) throws AddressException, MessagingException {
    sendMail("lisi@itheima.cn", "你好");
}
}
```

## 2.4.2 定时任务

### 2.4.2.1 什么是定时任务

定时任务用于安排以后在后台线程中执行的任务。可安排任务执行一次，或者定期重复执行。

### 2.4.2.2 定时任务的应用场景

#### 2.4.2.2.1 每隔1小时检查用户在线人数

每隔1小时系统自动统计在线的用户人数。首先需要编写统计用户在线人数功能，然后我们需要1小时执行一次。需要java中的定时任务。

#### 2.4.2.2.2 每隔5分钟更新新闻

新闻都是实时的，而且新闻都是每隔一段时间更新最新的新闻。所以我们需要定时任务完成。

### 2.4.2.3 Timer常用的API

#### 2.4.2.3.1 API介绍

1. `Timer()` 创建一个新计时器。
2. `schedule(TimerTask task, Date firstTime, long period)` 执行定时任务的方法  
`TimerTask`：要执行的定时任务  
`firstTime`：开始执行任务的时间  
`period`：间隔多长时间重复执行
3. `TimerTask()` 创建定时器的任务
4. `abstract void run()` 要执行的定时任务在run方法中定义

#### ##### 2.4.2.3.2 使用步骤

1. 创建定时器Timer对象
2. 调用Timer对象的schedule方法执行定时任务

3.创建定时器任务对象TimerTask

4.实现TimerTask的run方法

##### 244.2.3.3 案例代码

```
```java
public static void main(String[] args) throws AddressException, MessagingException {
    Timer timer = new Timer();
    timer.schedule(new TimerTask() {
        @Override
        public void run() {
            System.out.println("该睡觉了");
        }
    },new Date(),1000*3);
}
```

## 2.4.2.4 案例：每隔30秒向易邮邮件服务器发送邮件

### 2.4.2.4.1 案例需求

每隔30秒向易邮邮件服务器发送邮件

### 2.4.2.4.2 案例效果

每隔30秒钟，lisi账户接受到zhangsan发送的邮件

### 2.4.2.4.3 案例分析

1. 编写ServletContextListener监听器监听servletContext对象的创建
2. 在监听servletContext对象创建的方法contextInitialized中创建Timer定时器对象
3. 设置定时器执行任务
4. 创建TimerTask定时器任务对象
5. 实现TimerTask的run方法，其中使用邮件工具类，给lisi发送邮件

### 2.4.2.4.4 案例代码

```
public class SendMailListener implements ServletContextListener {
    public SendMailListener() {
    }
    public void contextInitialized(ServletContextEvent sce) {
        //创建定时器对象
        Timer timer = new Timer();
        //执行定时任务，每隔30秒执行一次
        timer.schedule(new TimerTask() {

            @Override
            public void run() {
                SimpleDateFormat format = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
                String time = format.format(new Date());
                System.out.println("开始发送邮件,现在时间"+time);
                try {

                    //发送邮件
                }
            }
        }, new Date(), 30000);
    }
}
```

```
        MailUtils.sendMail("lisi@itheima.com", "你好");
    } catch (AddressException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (MessagingException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    }
    }, new Date(), 1000*30);
}
public void contextDestroyed(ServletContextEvent sce) {
}
}
```