# 1 GUI

## GUI是什么

* Graphical User Interface(图形用户接口)
* java提供了一个 ***java.awt 包,***用于创建用户界面和绘制图形图像的所有类。
* 做j2ee方向的一般很少会用这个东西

## 1.2 GUI的简单使用

* 知识点：
  + 窗口，尺寸，位置、关闭、图标、动作监听、鼠标监听、键盘监听
* 代码

|  |
| --- |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) {  //创建一个窗口并显示  //1.创建窗口对象  Frame window = **new** Frame("我是一个窗口");    //2.设置窗口的大小  window.setSize(400, 400);    //3.设置位置  window.setLocation(200, 200);  //4.设置程序图标  window.setIconImage(Toolkit.*getDefaultToolkit*().createImage("qq.png"));  //5.监听窗口x 按钮  window.addWindowListener(**new** WindowAdapter() {  @Override  **public** **void** windowClosing(WindowEvent e) {  //释放资源  //.....    //退出程序  System.*exit*(0);  }  });      //6.添加按钮  //6.1创建按钮  Button btn1 = **new** Button("close");  //6.2添加到窗口  window.add(btn1);    //6.3再添加按钮  Button btn2 = **new** Button("mouse");  window.add(btn2);    //7.设置组件布局方式[流布局]  window.setLayout(**new** FlowLayout());    //8.监听按钮的点击  btn1.addActionListener(**new** ActionListener() {    @Override  **public** **void** actionPerformed(ActionEvent e) {  System.***out***.println("点击...");  //关闭程序  System.*exit*(0);  }  });    //9.监听鼠标的事件  btn2.addMouseListener(**new** MouseAdapter() {  @Override  **public** **void** mousePressed(MouseEvent e) {  // **TODO** Auto-generated method stub  System.***out***.println("鼠标按下...");  }    @Override  **public** **void** mouseReleased(MouseEvent e) {  // **TODO** Auto-generated method stub  System.***out***.println("鼠标松开...");  }  });    //10.键盘监听  Button btn3 = **new** Button("keyboard");  window.add(btn3);  btn3.addKeyListener(**new** KeyAdapter() {  @Override  **public** **void** keyReleased(KeyEvent e) {    //键盘对应字符  //e.getKeyCode():字符编码表  System.***out***.println(e.getKeyCode() + ":" + e.getKeyChar());  }  });      //显示窗口  window.setVisible(**true**);    }  } |
|  |

## 1.3事件处理要素

* 事件处理的三要素：事件、事件源、监听器
* 事件: 用户的一个操作，如：点击
* 事件源: 被操作的组件，如：按钮
* 监听器: 一个自定义类的对象, 实现了监听器接口, 包含事件处理方法,把监听器添加在事件源上, 当事件发生的时候虚拟机就会自动调用监听器中的事件处理方法

## 1.4 适配器模式

### 什么是适配器

* 在使用监听器的时候, 需要定义一个类事件监听器接口.
* 通常接口中有多个方法, 而程序中不一定所有的都用到, 但又必须重写, 这很繁琐.
* 适配器简化了这些操作, 我们定义监听器时只要继承适配器, 然后重写需要的方法即可.

### 适配器原理

* 适配器就是一个类, 实现了监听器接口, 所有抽象方法都重写了, 但是方法全是空的.
* 适配器类需要定义成抽象的,因为创建该类对象,调用空方法是没有意义的
* **目的就是为了简化程序员的操作,** 定义监听器时继承适配器, 只重写需要的方法就可以了

### 代码案例

|  |
| --- |
| **public** **class** Demo01 {  **public** **static** **void** main(String[] args) {  //适配器设计模式(掌握)  //1.创建按钮对象  Button btn = **new** Button();  btn.addMouseListner(**new** MouseAdapter() {  @Override  **public** **void** onceClick() {  System.***out***.println("单击。。");  }  });    }  }  **interface** MouseListner{  **public** **void** onceClick();//单击  **public** **void** doubleClick();//双击  **public** **void** rightClick();//右击  }  **abstract** **class** MouseAdapter **implements** MouseListner{  @Override  **public** **void** onceClick() {}    @Override  **public** **void** doubleClick() {}    @Override  **public** **void** rightClick() {}  }  **class** Button{    **public** **void** addMouseListner(MouseListner m){    }  } |

# 2 网络编程

## 2.1 计算机网络

是指将地理位置不同的具有独立功能的多台计算机及其外部设备，通过通信线路连接起来，在网络操作系统，网络管理软件及网络通信协议的管理和协调下，实现资源共享和信息传递的计算机系统。



## 2.2 网络编程

就是用来实现网络互连的不同计算机上运行的程序间可以进行**数据交换**。

## 2.3 网络编程三要素

### IP

* 每个设备在网络中的唯一标识
* 每台网络终端在网络中都有一个独立的地址，我们在网络中传输数据就是使用这个地址。
* IP地址的分类:IPv4 & IPv6
* IPv4
* IPv4：4个字节组成，4个0-255。大概42亿，30亿都在北美，亚洲4亿。2011年初已经用尽,IP格式: 192.168.1.168。
  + 测试网络连接 ping 192.168.1.68
  + 本地回路地址：ping 127.0.0.1(测试网卡有没插好)
  + 广播地址:255.255.255.255
* IP地址也**分公网地址（万维网使用）**和**私有地址(局域网使用)**,192.168.开头的就是私有址址
* IPv6：8组，每组4个16进制数。1a2b:0000:aaaa:0000:0000:0000:aabb:1f2f(IPv6暂先不用掌握)

### Port端口

* 端口号是每个程序在设备上的唯一标识
* 每个网络程序都需要绑定一个端口号，传输数据的时候除了确定发到哪台机器上，还要明确发到哪个程序，**端口号范围从0-65535**
* 编写网络应用就需要绑定一个端口号，**尽量使用1024以上的，1024以下的基本都被系统程序占用了**。
* 常用端口,面试会问
* mysql: 3306
* oracle: 1521
* web: 80
* tomcat: 8080

### 协议

协议是为计算机网络中进行数据交换而建立的规则、标准或约定的集合。

网络传输方式协议有TCP&UDP

TCP（传输控制协议）

面向连接（三次握手），数据安全，速度略低。分为客户端和服务端。

三次握手: 客户端先向服务端发起请求, 服务端响应请求, 传输数据

UDP(数据报传输协议)

面向无连接，数据不安全，速度快。不区分客户端与服务端。

## 2.4 Socket概述

* Socket翻译过来就是**套接字**
* 网络上具有唯一标识的IP地址和端口号组合在一起才能构成唯一能识别的标识符套接字。
* 通信的两端都有Socket，网络通信其实就是Socket间的通信。
* 数据在两个Socket间通过**IO流**传输。**Socket就是用来建立通信管道**
* Socket在应用程序中创建，通过一种绑定机制与驱动程序建立关系，告诉自己所对应的IP和port。

|  |
| --- |
|  |

## 2.5 TCP通信原理图解

|  |
| --- |
|  |

## 2.6 UDP通信原理图解

|  |
| --- |
|  |

## 2.7 案例：UDP数据传输

**实现步骤：**

**发送Send**

创建DatagramSocket, 随机端口号

创建DatagramPacket, 指定数据, 长度, 地址, 端口

使用DatagramSocket发送DatagramPacket

关闭DatagramSocket

**接收Receive**

创建DatagramSocket, 指定端口号

创建DatagramPacket, 指定数组, 长度

使用DatagramSocket接收DatagramPacket

关闭DatagramSocket

从DatagramPacket中获取数据

### 案例：实现UDP的发送和接收

|  |
| --- |
| **public** **static** **void** main(String[] args) **throws** Exception {  // 1.发送Send  // 创建DatagramSocket, 随机端口号  DatagramSocket sender = **new** DatagramSocket();    // 创建DatagramPacket【数据包】, 指定数据, 长度, 地址, 端口  String s = "你好吗？最近很想你";  **byte**[] bs = s.getBytes("UTF-8");  DatagramPacket packet = **new** DatagramPacket(bs, bs.length, Inet4Address.*getByName*("192.168.1.125"), 5288);    // 使用DatagramSocket发送DatagramPacket  sender.send(packet);    // 关闭DatagramSockets  sender.close();  } |
| **public** **static** **void** main(String[] args) **throws** IOException {  // 2.接收Receive  // 创建DatagramSocket, 指定端口号  DatagramSocket receiver = **new** DatagramSocket(5288);    // 创建DatagramPacket, 指定数组, 长度  **byte**[] buf = **new** **byte**[1024];  DatagramPacket packet = **new** DatagramPacket(buf, 1024);    // 使用DatagramSocket接收DatagramPacket  receiver.receive(packet);    // 关闭DatagramSocket  receiver.close();    // 从PackeDatagramt中获取数据  String s = **new** String(buf, 0, packet.getLength(), "UTF-8");  System.***out***.println("接收到数据:" + s);  } |

### 案例：添加键盘和死循环功能

|  |
| --- |
| public class Demo\_Sender {  public static void main(String[] args) throws Exception {  // 1.发送Send  // 创建DatagramSocket, 随机端口号  DatagramSocket sender = new DatagramSocket();    //2 Scanner  Scanner scanner = new Scanner(System.in);    while(true){  // 创建DatagramPacket【数据包】, 指定数据, 长度, 地址, 端口  System.out.println("输入内容：");  String s = scanner.nextLine();  byte[] bs = s.getBytes("UTF-8");  DatagramPacket packet = new DatagramPacket(bs, bs.length, Inet4Address.getByName("192.168.1.125"), 5288);    // 使用DatagramSocket发送DatagramPacket  sender.send(packet);  }      // 关闭DatagramSockets  //sender.close();  }  } |
| package lesson05;  public class Demo\_Receiver {  public static void main(String[] args) throws IOException {  // 2.接收Receive  // 创建DatagramSocket, 指定端口号  DatagramSocket receiver = new DatagramSocket(5288);    while(true){  // 创建DatagramPacket, 指定数组, 长度  byte[] buf = new byte[1024];  DatagramPacket packet = new DatagramPacket(buf, 1024);    // 使用DatagramSocket接收DatagramPacket  receiver.receive(packet);    // 关闭DatagramSocket  //receiver.close();    // 从PackeDatagramt中获取数据  String s = new String(buf, 0, packet.getLength(), "UTF-8");  System.out.println(new Date() + "接收到数据:" + s);  }  }  } |

### 案例：使用双线程来收发数据

|  |
| --- |
| **public** **class** Demo {  **public** **static** **void** main(String[] args) {  // 案例:UDP数据传输+键盘和循环+双线程实现(了解)  **new** Receiver().start();  **new** Sender().start();  }  }  **class** Receiver **extends** Thread {  @Override  **public** **void** run() {  // **TODO** Auto-generated method stub  **try** {  // 创建DatagramSocket, 指定端口号  DatagramSocket receiver = **new** DatagramSocket(5288);  **while** (**true**) {  // 创建DatagramPacket, 指定数组, 长度  **byte**[] buf = **new** **byte**[1024];  DatagramPacket packet = **new** DatagramPacket(buf, 1024);  // 使用DatagramSocket接收DatagramPacket  receiver.receive(packet);  // 关闭DatagramSocket  // receiver.close();  // 从PackeDatagramt中获取数据  String s = **new** String(buf, 0, packet.getLength(), "UTF-8");  System.***out***.println(**new** Date() + "接收到数据:" + s);  }  } **catch** (Exception e) {  // **TODO**: handle exception  }  }  }  **class** Sender **extends** Thread {  @Override  **public** **void** run() {  **try** {  // 创建DatagramSocket, 随机端口号  DatagramSocket sender = **new** DatagramSocket();    //2 Scanner  Scanner scanner = **new** Scanner(System.***in***);    **while**(**true**){  // 创建DatagramPacket【数据包】, 指定数据, 长度, 地址, 端口  System.***out***.println("输入内容：");  String s = scanner.nextLine();  **byte**[] bs = s.getBytes("UTF-8");  DatagramPacket packet = **new** DatagramPacket(bs, bs.length, Inet4Address.*getByName*("192.168.1.125"), 5288);    // 使用DatagramSocket发送DatagramPacket  sender.send(packet);  }  } **catch** (Exception e) {  // **TODO**: handle exception  }  }  } |

## 2.8 案例：TCP数据传输【多练】

》客户端用键盘+死循环实现

|  |
| --- |
| **public** **class** Demo\_Client {  **public** **static** **void** main(String[] args) **throws** IOException {    //1.创建Socket对象  Socket socket = **new** Socket("192.168.1.125", 9999);    //2.获取输入流出  InputStream is = socket.getInputStream();  OutputStream os = socket.getOutputStream();    BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is));  PrintWriter pw = **new** PrintWriter(os);    //3.发送数据给服务端面  pw.write("Java 还是不错的...\r\n");  pw.flush();    //4.读服务响应的数据  System.***out***.println("读取响应数据:" + br.readLine());    //关流  br.close();  socket.close();  }  } |
| **public** **class** Demo\_Server {  **public** **static** **void** main(String[] args) **throws** Exception {  //1.创建个服务端Socket对象,绑定端口  ServerSocket serverSocket = **new** ServerSocket(9999);    //2.监听客户端面连接  **while**(**true**){  Socket clientSocket = serverSocket.accept();    //3.接收客户端面发送的数据  **new** Thread(){  **public** **void** run() {  **try** {  //读写数据都是通过IO  InputStream is = clientSocket.getInputStream();  OutputStream os = clientSocket.getOutputStream();    BufferedReader br = **new** BufferedReader(**new** InputStreamReader(is));  PrintWriter pw = **new** PrintWriter(os);    **while**(**true**){  //读取客户端的数据  String line = br.readLine();  **if**(line != **null**){  System.***out***.println(clientSocket + ":" + **new** Date() + "接收到客户端发送的数据：" + line);    //响应客户端面  pw.write("receiver success\r\n");  pw.flush();  }    }  } **catch** (Exception e) {  // **TODO**: handle exception  }    };  }.start();  }  }  } |

# 3 正则表达式

## 3.1 正则表达式的作用

用于匹配字符串，比如匹配手机号码，邮箱的格式

## 3.2 校验QQ

### 方式一：未使用正则

|  |
| --- |
| /\* 校验qq号码.  1:要求必须是5-15位数字  2:0不能开头  3:必须都是数字\*/    String qq = "10a101";    //1:要求必须是5-15位数字  **if**(qq.length() >=5 && qq.length() <= 15){  System.***out***.println("qq号的长度正确 ");    //2: 0不能开头  **if**(!qq.startsWith("0")){    //3.字符中是否有非数字字符  **for**(**int** i=0;i<qq.length();i++){  **char** ch = qq.charAt(i);  **if**( !(ch >= '0' && ch <= '9')){  System.***out***.println("不合法字符:" + ch);  }  }    }**else**{  System.***out***.println("qq号不能以0开头");  }  }**else**{  System.***out***.println("qq号码的长度不正确");  }  } |

### 方式二：使用正则

|  |
| --- |
| //1.qq的正则  String regex = "[1-9]\\d{4,14}";    //2.打印匹配结果  System.***out***.println("1030103135".matches(regex));  System.***out***.println("01030103135".matches(regex));  System.***out***.println("1030".matches(regex));  System.***out***.println("1030103135111112".matches(regex));  System.***out***.println("10a30".matches(regex)); |

## 3.3正则表达式的构造摘要

### 字符类

[abc] a、b 或 c（简单类）

[^abc] 任何字符，除了 a、b 或 c（否定）

[a-zA-Z] a 到 z 或 A 到 Z，两头的字母包括在内（范围）

[a-zA-Z\_0-9] a 到 z 或 A 到 Z，\_,0到9（范围）

[0-9] 0到9的字符都包括

[a-d[m-p]] a 到 d 或 m 到 p：[a-dm-p]（并集）

[a-z&&[def]] d、e 或 f（交集）

[a-z&&[^bc]] a 到 z，除了 b 和 c：[ad-z]（减去）

[a-z&&[^m-p]] a 到 z，而非 m 到 p：[a-lq-z]（减去）

|  |
| --- |
|  |

### 预定义字符类

. 任何字符

\d 数字：[0-9]

\D 非数字： [^0-9]

\s 空白字符：[ \t\n\x0B\f\r]

\S 非空白字符：[^\s]

\w 单词字符：[a-zA-Z\_0-9]

\W 非单词字符：[^\w]

|  |
| --- |
|  |

### 数量词

X? X，一次或一次也没有

X\* X，零次或多次

X+ X，一次或多次

X{n} X，恰好 n 次

X{n,} X，至少 n 次

X{n,m} X，至少 n 次，但是不超过 m 次

|  |
| --- |
|  |

## 3.4 正则表达式的分割功能

* public String[] split(String regex)
* 根据给定正则表达式的匹配拆分此字符串

|  |
| --- |
| /\*String s = "11-23-21-20-28";  String[] arr = s.split("-");  for(String str : arr){  System.out.println(str);  }\*/    String s = "11.23.21.20.28";  /\*\*  \* 如果是按.拆分，注意要添加转义字符  \*/  String[] arr = s.split("\\.");  **for**(String str : arr){  System.***out***.println(str);  } |

## 3.5 正则表达式的替换功能

* public String replaceAll(String regex,String replacement)

|  |
| --- |
| String s = "520 java 520 h5 520 c";    //把520都改成"我爱你"  //s = s.replaceAll("\\d{3}", "我爱你");  s = s.replaceAll("520", "我爱你");  System.***out***.println(s); |

## 3.6 Pattern和Matcher

### 正则的另一种写法

|  |
| --- |
| // 检验QQ  // boolean b1 = "101030".matches("[1-9]\\d{4,14}");  // System.out.println(b1);  //等效于上面一行代码  Pattern p = Pattern.*compile*("[1-9]\\d{4,14}");//正则  Matcher m = p.matcher("101030");  **boolean** b = m.matches();  System.***out***.println(b); |
|  |

### 扣出手机字符串

|  |
| --- |
| String s = "我的手机是18988888888,我曾用过18987654321,还用过18812345678";    //Matcher 的find和group方法    //1.匹配手机正则表达(11位)  String regex = "1[3789]\\d{9}";    //2.创建Pattern  Pattern p = Pattern.*compile*(regex);    //3.创建Matcher  Matcher m = p.matcher(s);    //4.找到匹配正则的内容  /\* System.out.println(m.find());  System.out.println(m.group());  System.out.println(m.find());  System.out.println(m.group());  System.out.println(m.find());  System.out.println(m.group());\*/  **while**(m.find()){  System.***out***.println(m.group());  } |