22. 图形用户界面

- 人机界面的发展
- 图形用户界面的基本概念

  结构，构造，相关问题
- 基于标准库包 **tkinter** 实现图形用户界面

  一些相关概念和机制

  对象和布局安排
- **GUI**编程实例

人机交互方式的演化

- 自从有了计算机，就出现了人如何与计算机交换信息的问题
- 早期需要与计算机交换信息时，人先准备好程序和数据，做成一个作业提交计算机，而后等待计算的结果
  - 人们把这种计算方式称为“批处理”
  - 批处理是最基本的计算方式，现在仍然在不同的地方存在
- 随着计算机应用的发展，人们发现批处理方式经常不能满足需要
  - 经常需要用户根据计算的进展情况提供计算所需的数据
  - 用户需要根据计算中的情况发布指令，指挥下步的工作
- 这类需求逐步发展出“交互式”的计算方式
  - 计算机在工作中不断输出一些信息或请求
  - 用户通过交互式输入，提供数据或指令，指导计算进程

人机交互方式的演化

- 早期与计算机交互的设备（通道）是类似打字机的输入设备和在连续卷纸上打印输出的打印机设备（输出即是print）

  数据传输的效率低而且成本很高

- 随着阴极射线管显示设备的发展，人机交互通道转移到显示屏，只有需要保存和反复阅读的材料才打印到纸面（称为硬拷贝）
  - 键盘输入立即复显在显示器屏幕，使人可以观察和更正
  - 程序输出也显示在屏幕，降低了交互成本
- 早期计算机显示器是“基于字符的显示器”，屏幕划分为一组字符行，每行划分为一系列字符位置，24*80 是一种常见配置
  - 这种显示器只能显示一个个标准大小的字符
  - 受限的显示形式限制了计算机应用的发展。有些专用计算设备配置昂贵的“图形显示器”，用于特殊需要(如工程设计)

人机交互方式的演化

- 基于字符行的人机交互，只能供专业人员使用，也只有特殊的专业人员群体愿意（花精力去）使用
  - 人机交互方式的特征大大限制了计算机应用的发展
  - 需要全新的思想和技术，满足社会发展的需要
- **1960**年代末，**Douglas C. Engelbart**发明了鼠标器，提供了人机交互的另一全新的输入通道（除键盘之外）

  Engelbart因人机交互方面的工作获**1997**年图灵奖
- 另一方面，显示器硬件技术不断发展，成本降低，为全新的人机交互方式的引入奠定了技术基础
- 但是，怎样一套新概念能更好满足更广泛的计算机用户的需要

  还要有一套实现技术，提供相对易用的编程接口，使开发者能比较容易地为各种具体应用系统建立人机界面

人机交互方式的演化

- 讨论现代人机接口技术，必须提到一个人/一个机构/一个语言
- **Alan Kay**在学习期间就一直思考如何构造一种儿童也能使用的计算机系统，**1960**年代末博士毕业后到斯坦福大学工作，后到施乐公司（**Xerox**）的帕洛阿尔托研究中心（**PARC**），其最重要的成果就是和其他人一起开发出**Smalltalk**语言和系统
- **Alan Kay**因为这些工作获**2003**年图灵奖
- 施乐是静电复印技术的鼻祖，发明了许多与现代文件处理有关的技术。设立**Palo Alto**研究中心是为了研究新型的文件处理技术。该中心汇集了一批优秀研究人员，在计算机领域的重要研究成果包括 **Ethernet**、新型个人计算机、图形用户界面 **(Graphic User Interface**，**GUI)** 和桌面交互模式/技术, 面向对象编程等
- **Smalltalk** 总结出今天面向对象语言的几乎所有基本概念，开发出的图形用户界面的范式和技术，被所有后来的系统所继承

图形用户界面的发展和广泛使用

- Steve Jobs在PARC看到Smalltalk非常震惊，回去后组织开发了苹果公司的Lisa计算机，以及后来第一种非常成功的基于GUI系统的Macintosh计算机（1984），使GUI进入个人计算领域
- 1985年底微软推出Windows操作系统，后推出GUI编程接口，支持用户自己开发基于GUI的应用系统，推动了GUI的普及
- 今天，绝大部分应用程序都采用GUI用户界面，所有重要操作系统都采用GUI界面，并支持应用程序的GUI编程
- 实际上，GUI的基本概念和模式都是Alan Kay等人在PARC开发Smalltalk系统时开发出来的，主要是：
  - 用一种指点设备（鼠标、手指等）加上键盘作为输入设备，通过手眼配合的方式提供另一条输入通道
  - 由Window(窗口)/Menu(菜单)/Icon(图标)/Button(按钮)/滚动条等组成的一套GUI概念、形式和功能，及其实现技术

图形用户界面的编程方式

- 支持GUI编程的系统都以某种形式提供一套GUI功能和相应机制，开发者编程时可以调用这些机制，以便
  - 设定自己需要的GUI界面
  - 把需要执行的操作关联于各种图形界面元素，当程序用户操作GUI界面的各种元素时，就会导致这些操作的执行
- GUI编程通常采用一种事件驱动的模式
  - 把用户的各种GUI操作看作事件
  - 为每种事件关联一种操作（可以动态地改变关联）
  - 用户操作事件发生时，GUI基础系统就会调用其关联操作
- 用户操作GUI界面元素，就会生成一个事件消息

  GUI基础系统通过一个消息循环，将消息传给处于活动状态的对象。应接受该消息的对象截获自己的消息，执行相应动作

Python的tkinter

- Python标准库包 tkinter 提供了一套跨平台的GUI 功能，可以在任何支持Python官方实现的环境中使用（如Windows，Linux，IOS）。它基于著名的界面开发语言Tcl/Tk
- 下面介绍如何基于这个包做GUI编程

  Python还有几个比较流行的GUI包，其他编程环境中的GUI功能也类似，下面的介绍具有普遍的参考价值
- 下面将一直用下面形式导入tkinter包

  ```
  import tkinter as tk
  ```

  这里的的as子句引入一个新名字，为使用方便

  注意，这样导入tkinter包，其中功能只能通过圆点形式调用
- 人们在写Python程序时，经常采用import-as形式的写法，为引入的包提供简称，简化程序描述

tkinter及其使用

- 最大的GUI对象是一个窗口，在tkinter里的相关概念是Frame。其中可以有许多部件，一般称为物件（witgets）

  下面将采用面向对象的编程方式，把一个窗口包装为一个整体，使其相关信息成为对象的内部细节

  这只是使用tkinter编程的一种方法，也可以不这样做。在一些书籍中可能看到其他做法

  采用面向对象方式的优点是良好的信息封装，能更好地避免不同程序部分的相互干扰，适合用于编写较复杂的程序

- GUI编程通常牵涉到许多比较琐碎的细节
  - 一个窗口通常会包含许多物件
  - 每个物件可能有许多细节，如位置、文本显示、相关操作等
  - GUI编程实际上是比较繁琐的工作，还有感官设计问题

一个简单窗口

- 下面将采用实例的方式介绍tkinter及其使用
- 首先，下面代码片段定义了一类简单窗口

```
class FirstWin(tk.Frame): # 继承Frame定义自己的GUI类
    def __init__(self, title):
        tk.Frame.__init__(self) # 初始化Frame实例
        # master 的值是 Frame 实例对象所在的主窗口
        self.master.title(title) # 设置窗口标题
        self.pack() # 安置本Frame
        # 建立窗口中的文字对象，这是一个 Label 对象
        # 第一个参数指定父物件
        self._label = tk.Label(
            self, text = "下午好，这里是北京大学数学学院!")
        self._label.pack()
        self.mainloop() # 启动窗口的消息循环，接受消息并响应
```

显示一张图片的窗口

- 下面窗口理显示一张图片：

```
class ImageWin(tk.Frame):
    def __init__(self): # 可把图片文件等作为创建对象的参数
        """Sets up the window and widgets."""
        tk.Frame.__init__(self)
        self.master.title("北京大学")
        self.pack()
        self._image = tk.PhotoImage(file = "pku.png")
        self._imageLabel = tk.Label(
            self, image = self._image)
        self._imageLabel.pack()
        self._textLabel = tk.Label(
            self, text = "北京大学西门")
        self._textLabel.pack()
        self.mainloop()
```

标准库的一个实例

```
class Application(tk.Frame):
    def __init__(self):
        tk.Frame.__init__(self)
        self.master.title("Main"); self.pack()
        self.createWidgets() # 调用方法设置窗口物件
        self.mainloop()

    def createWidgets(self): # 在窗口里设置两个按钮
        self._hi = tk.Button(self) # 分别设置属性
        self._hi["text"] = "Hello World\n(click me)"
        self._hi["command"] = self._say_hi # 关联命令 _say_hi
        self._hi.pack(side="top") # 位置在上
        self._quit = tk.Button( # 创建时提供属性值
            self, text="QUIT", fg="red",
            command=self.master.destroy) # 关联的命令是退出主窗口
        self._quit.pack(side="bottom") # 位置在下

    def _say_hi(self): print("hi there, everyone!")
```

一些基本概念

- **window**（窗口），通常指屏幕上的一个矩形区域
- **top-level window**（顶层窗口），指屏幕一个独立窗口，具有标准的框架形式，可以在桌面上移动、调整大小等
- **widget**（物件），组成**GUI**的所有构件，如按钮、选择按钮、正文区、框架、标签等
- **frame**（框架），是**tkinter**里组织复杂版面编排的基本单位，是一个矩形区域，其内部可以包含其他物件
- **child, parent**（子，父），在建立物件时需要建立其父子关系，例如把一些物件放入一个框架，就是作为其子物件
- 在构建**GUI**中的物件时，需要做两步：
  1. 创建物件并设置其属性（很多有默认值）
  2. 指定其布局（**layout**）位置

建立主窗口

- 采用面向对象的方式建立一个**GUI**，通常把**GUI**的接口创建为类**Frame**的一个派生类
  - 首先调用**Frame**的**__init__**做必要的初始化
  - 而后设置这个界面的具体物件，包括为一些物件关联必要的命令，称为“回调函数”，就是在有关事件发生时执行的动作。所有回调函数都是无参数的函数
- 建立物件时需要给定其父结构，通常就是本对象**self**。相应属性可以在创建时给定，也可在创建后通过字典元素赋值的方式

  实际上，物件的属性可以在运行中随时修改，从而改变物件的显示形式和内容
- 构造好的物件可以通过布局位置。**pack**是一个简单的布局命令，最常用也是功能更强大的命令**grid**下面介绍

物件

- 前面讨论的实例已经涉及了一组常用物件
  - 每个物件都是一个类
  - 构造物件类的实例时，可以通过构造的参数为物件的一些属性提供值，没提供值的属性将采用默认值
  - 物件是容器，可以通过属性赋值的方式为属性设定值，包括修改已经设定的值。从这个角度，一个物件就像一个字典
  - 每个物件类有一组实例方法，至少支持布局操作
- 一些材料：
  - 网页上的**tkinter**手册
  - 有关**pack**和**grid**布局管理器的说明文件
  - 下面网页包含最全的材料：**http://effbot.org/tkinterbook/**

物件

- 物件通常都有很多属性，通常（常用的）有：
  - 与大小有关的属性**height, width**
  - 与颜色有关的**bg**（**background**）和**fg**（**foreground**）
  - 正文属性**text**，放置位置**anchor**（**LEFT/CENTER/RIGHT**），一些物件有正文属性，如**Label**，**Button**等
  - 关联操作**command**
- 具体属性情况可以查阅**thinter**手册，从目录中找到相应的物件，在具体章节可以查到一个属性列表
- 每个物件支持一批操作，有一些专门操作，可以在手册中有关物件的章节找到

  另外有一批所有物件支持的通用操作，见第**26**节，**Universal widget methods**

物件的布局

- **pack是最简单的布局方法**
  - **默认方式是简单放置，默认定位方式是 CENTER（居中）**
  - **默认情况多个pack物件从上向下逐项排列**
  - **可用属性 side=LEFT 要求从左到右逐项排列，用 RIGHT 或 BOTTOM 要求从右到左或从下向上排列（默认是 TOP）。这些布局属性并不适合混合使用，复杂的布局应该用grid**
- **grid是最重要的布局管理器**
  - **如果遇到复杂布局问题，应该用grid。不要混合使用grid和pack，得到的效果无法预计**
  - **下面通过几个例子说明grid的使用方式。grid有一批布局参数，采用参数默认值时grid的效果与pack类似**
  - **下面通过几个例子展示grid布局参数的情况**

grid布局示例

- 代码和效果（设master是父对象，row/column定行列）：

```
Label(master, text="First").grid(row=0)
Label(master, text="Second").grid(row=1)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
```

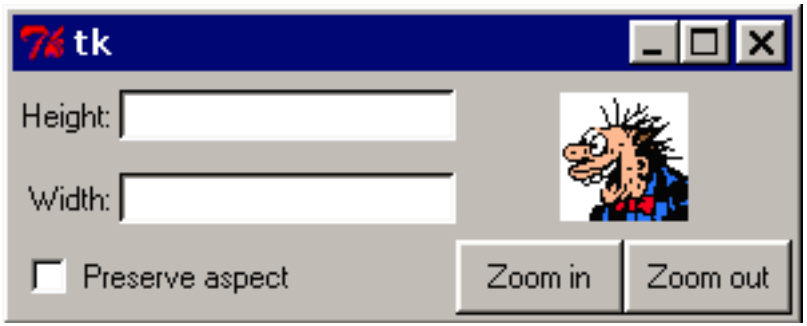
- 代码和效果（W表示西，sticky说明扩展物件后的对齐方向）：

```
Label(master, text="First").grid(row=0, sticky=W)
Label(master, text="Second").grid(row=1, sticky=W)
e1 = Entry(master)
e2 = Entry(master)
e1.grid(row=0, column=1)
e2.grid(row=1, column=1)
```

grid布局示例

- 代码和效果（忽略创建对象的操作）：

```
label1.grid(sticky=E) # 对齐在东
label2.grid(sticky=E)
entry1.grid(row=0, column=1)
entry2.grid(row=1, column=1)
checkbutton.grid(columnspan=2, sticky=W)
image.grid(row=0, column=2, columnspan=2, rowspan=2,
           sticky=W+E+N+S, padx=5, pady=5)
button1.grid(row=2, column=2)
button2.grid(row=2, column=3)
```



columnspan描述列伸展数，rowspan是行伸展数，W+E+N+S表示向四面扩展（居中），padx和pady描述在x和y方向给物件增加的像素数

GUI变量的使用

- Entry类物件用于输入，需要关联一个字符组容器
  - StringVar是供存放字符序列的字符组容器类
  - IntVar是供存放整数的字符组容器类
  - DoubleVar是供存放浮点数的字符组容器类
- 下面有几个例子里展示了这种变量容器的使用
- 下面看一些GUI示例