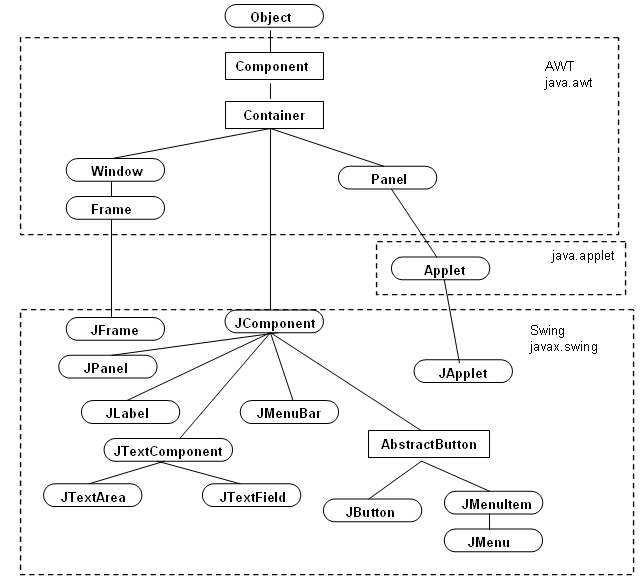
# 一、技术背景介绍

AWT是Abstract Window ToolKit (抽象窗口工具包)的缩写，这个工具包提供了一套与本地图形界面进行交互的接口。AWT 中的图形函数与操作系统所提供的图形函数之间有着一一对应的关系，我们把它称为peers。 也就是说，当我们利用 AWT 来构件图形用户界面的时候，我们实际上是在利用操作系统所提供的图形库。由于不同操作系统的图形库所提供的功能是不一样的，在一个平台上存在的功能在另外一个平台上则可能不存在。为了实现Java语言所宣称的"一次编译，到处运行"的概念，AWT 不得不通过牺牲功能来实现其平台无关：**AWT 所提供的图形功能是各种通用型操作系统所提供的图形功能的交集。**由于AWT 是依靠本地方法来实现其功能的，我们通常把AWT控件称为重量级控件。

Swing 是在AWT的基础上构建的一套新的图形界面系统，它提供了AWT所能够提供的所有功能，并且**用纯粹的Java代码对AWT 的功能进行了大幅度的扩充**。例如说并不是所有的操作系统都提供了对树形控件的支持，Swing利用了AWT 中所提供的基本作图方法对树形控件进行模拟。由于 Swing 控件是用100%的Java代码来实现的，因此在一个平台上设计的树形控件可以在其他平台上使用。由于在Swing 中没有使用本地方法来实现图形功能，我们通常把Swing控件称为轻量级控件。

AWT和Swing之间的基本区别：**AWT 是基于本地方法的C/C++程序，其运行速度比较快；Swing是基于AWT 的Java程序，其运行速度比较慢。**对于一个嵌入式应用来说，目标平台的硬件资源往往非常有限，而应用程序的运行速度又是项目中至关重要的因素。在这种矛盾的情况下，简单而高效的AWT 当然成了嵌入式Java的第一选择。而在普通的基于PC或者是工作站的标准Java应用中，硬件资源对应用程序所造成的限制往往不是项目中的关键因素，所以在标准版的Java中则提倡使用Swing，也就是通过牺牲速度来实现应用程序的功能。



NOTE1:

invokeLater()方法调用完毕后会被销毁，匿名内部类是作为临时变量存在的，分配的内存在方法调用完毕会被释放。这对于只需要在一个地方使用是可以节省内存的，而且这个匿名内部类是不可以被其他方法或类使用，只能被EventQueue.invokeLater()使用。如果需要一个在很多地方都能用到的类，而不是只在某一个类或方法里面，定义成匿名内部类是不可取的。

All GUI related things in java should always go through a single thread. The thread is our legendary "AWT-EventQueue-0" .Hence all GUI related actions should necessarily go through the AWT Event thread. If not so you may end up in a deadlock. For small programs, this might never happen. But for a huge java application if you try frame.setVisible(true) kind of thing in main thread, you will soon find yourself searching a new job. What invokeLater() does is to post your Runnable in the AWT thread's event queue. So the code in your run method will be executed in the AWT-Event queue thread.

plus:

lamada

EventQueue.invokeLater(() -> {

try {

// 启动登陆界面

CatLogin frame = new CatLogin();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

});

是lambda写法，等价于：

EventQueue.invokeLater(new Runnable() {

@Override

public void run() {

try {

// 启动登陆界面

CatLogin frame = new CatLogin();

frame.setVisible(true);

} catch (Exception e) {

e.printStackTrace();

}

}

});

**NOTE2**

JPanel is a Swing’s lightweight container which is used to group a set of components together. JPanel is a pretty simple component which, normally, does not have a GUI (except when it is being set an opaque background or has a visual border).

**NOTE3**

Q1: Why is paintComponenet used twice, for the name of the method and as we call a method from super (JPannel)?

A1: It's not "used" twice. It is overridden once, but you want to call the parent (JPanel) class's super method so that you're sure that it does its own house-keeping painting, including painting its children and clearing out any dirty bits from the screen.

Q2: What is Graphics g, isn't it just a reference variable for an object of Graphics since we don't set it equal to = new Graphics();

A2: It's a Graphics parameter. You don't set it = new Graphics() because the JVM does this for you. It calls the method behind the scenes when needed, and provides the parameter.

Q3: Why does the method name in my class have to be paintComponent to call upon the method paintComponent from JPannel or super

A3: It has to override the super's method so that the JVM calls the right method when it wants to draw the GUI.

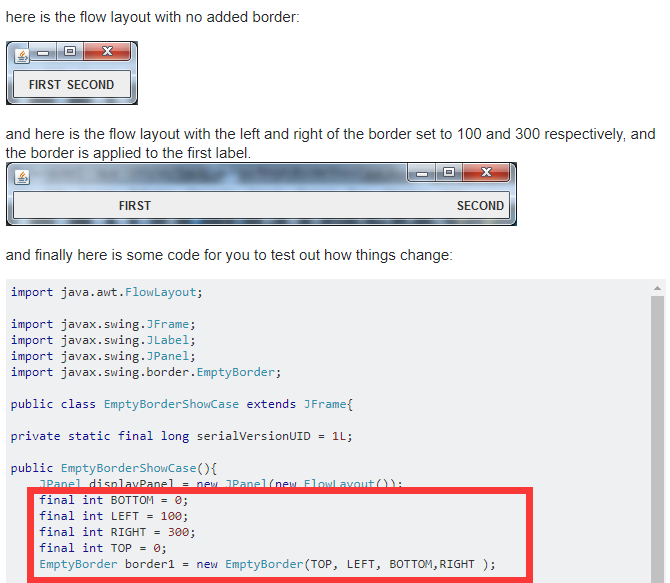
Q4: The method in my class paintComponent takes the parameter of a Graphics object but when does paintComponent even get called and when is the parameter of Graphics inserted.

A4: Again, it is called by the JVM when either your program wants to repaint the GUI, such as when you call repaint() or when the operating system wants to repaint a window such as if a window is minimized and restored.

**NOTE4**

EmptyBorders are as the name implies, a border without any content, basically all it does is add an invisible border around the component its applied to.

A class which provides an empty, transparent border which takes up space but does no drawing.



**NOTE5**

java中的properties文件是一种配置文件，主要用于表达配置信息，文件类型为\*.properties，格式为文本文件，文件的内容是格式是"键=值"的格式。加上你写一个方法来获取配置信息的内容，也就是读取.properties文件。方法设置返回值，可以用来返回等号后面的信息，比如你想获取8888的话，只需要给写的方法传一个参数PORT，就能返回8888。工程里很多地方都会用到配置信息里的东西，如果没有配置文件，将来要修改端口号或者HOST的时候就比较麻烦，需要改代码。有配置文件就不一样了，只修改配置文件里等号后面的数据就可以了。工程里其他地方用HOST和PORT都是用给读取配置文件的方法传参数的形式调用数据的，所以只修改配置文件的内容就能全部修改为想要的数据。最主要的是不用修改代码，这点很重要，所以工作中配置文件往往比java代码还要多。，当然不止是.properties类型的，更多的是.xml类型

**NOTE6**

IP地址+端口号组成了所谓的Socket，Socket是网络上运行的程序之间双向通信链路的终结点。

**NOTE7**

1.Java序列化与反序列化

Java序列化是指把Java对象转换为字节序列的过程；而Java反序列化是指把字节序列恢复为Java对象的过程。

2.为什么需要序列化与反序列化

我们知道，当两个进程进行远程通信时，可以相互发送各种类型的数据，包括文本、图片、音频、视频等， 而这些数据都会以二进制序列的形式在网络上传送。那么当两个Java进程进行通信时，能否实现进程间的对象传送呢？答案是可以的。如何做到呢？这就需要Java序列化与反序列化了。换句话说，一方面，发送方需要把这个Java对象转换为字节序列，然后在网络上传送；另一方面，接收方需要从字节序列中恢复出Java对象。

当我们明晰了为什么需要Java序列化和反序列化后，我们很自然地会想Java序列化的好处。其好处一是实现了数据的持久化，通过序列化可以把数据永久地保存到硬盘上（通常存放在文件里），二是，利用序列化实现远程通信，即在网络上传送对象的字节序列。

3.如何实现Java序列化与反序列化

1）JDK类库中序列化API

java.io.ObjectOutputStream：表示对象输出流

它的writeObject(Object obj)方法可以对参数指定的obj对象进行序列化，把得到的字节序列写到一个目标输出流中。

java.io.ObjectInputStream：表示对象输入流

它的readObject()方法源输入流中读取字节序列，再把它们反序列化成为一个对象，并将其返回。

2）实现序列化的要求

只有实现了Serializable或Externalizable接口的类的对象才能被序列化，否则抛出异常。

3）实现Java对象序列化与反序列化的方法

假定一个Student类，它的对象需要序列化，可以有如下三种方法：

方法一：若Student类仅仅实现了Serializable接口，则可以按照以下方式进行序列化和反序列化

ObjectOutputStream采用默认的序列化方式，对Student对象的非transient的实例变量进行序列化。

ObjcetInputStream采用默认的反序列化方式，对对Student对象的非transient的实例变量进行反序列化。

方法二：若Student类仅仅实现了Serializable接口，并且还定义了readObject(ObjectInputStream in)和writeObject(ObjectOutputSteam out)，则采用以下方式进行序列化与反序列化。

ObjectOutputStream调用Student对象的writeObject(ObjectOutputStream out)的方法进行序列化。

ObjectInputStream会调用Student对象的readObject(ObjectInputStream in)的方法进行反序列化。

方法三：若Student类实现了Externalnalizable接口，且Student类必须实现readExternal(ObjectInput in)和writeExternal(ObjectOutput out)方法，则按照以下方式进行序列化与反序列

serialVersionUID主要是辅助序列化和反序列化，序列化和反序列化的时候，serialVersionUID必须一致，反序列化才会成功。具体的序列化过程是这样的：序列化操作的时候系统会把当前类的serialVersionUID写入到序列化文件中，当反序列化时系统会去检测文件中的serialVersionUID，判断它是否与当前类的serialVersionUID一致，如果一致就说明序列化类的版本与当前类版本是一样的，可以反序列化成功，否则失败。