SWT：实现自我绘制的Button组件 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/144/2021_2022_SWT_EF_BC_9A_E5_AE_9E_E7_c104_144684.htm 在所有SWT组件中，Button几乎是最常用的，其功能在对于一般的情况来说也足够丰富了。你可以为Button组件设置要显示在其中的文本或者图像、设定ToolTip，甚至只要修改一个风格样式就能得到一个看上去相当不错的方向箭头按钮。 然而，我对Button组件还是不能感到满意。最大的遗憾就是：对它的外观，所能做的工作也就仅限于此了。如果你想让按钮拥有一个漂亮的、渐变色的背景和一些特殊的文字效果，怎么办呢？答案是没有办法。Button类里面似乎没有任何方法提供我想要的功能。 我曾尝试过的第一个想法是用Button.addPaintListener来修改按钮的外观。但是，结果令人失望虽然它显示出来的时候的确按照预想进行绘制了，但是当你用鼠标去按它的时候，马上又变回了原本灰头土脸的样子。显然，在按下按钮的时候，它并不是触发paint事件，而是按照自己的想法画出原本的按钮，于是我的工作全部白费了。 如果尝试为按钮设定图像会怎么样呢？这也不是一个好主意。首先，不管你选择什么样的图像，都没办法去掉按钮四周的边框，而正是这些边框严重破坏了图像的和谐感；其次，如果你的程序有几十甚至上百个按钮，为每个按钮都维护一幅图像（甚至更多理论上每个按钮在普通状态和被按下、禁用的状态下，甚至当鼠标移进移出按钮的时候，都应当显示不同的图像）明显是在浪费系统资源；如果你们的美工听说需要做几百个图片，大概也不会给你好脸色看。此外，图像有一个严重的缺点

是：它所拥有的像素数目是固定的，难以随着界面的放大和缩小同时变化。如果强制进行缩放的话，会出现明显的锯齿和失真，最终让你精心设计的窗口变得惨不忍睹。最好还是放弃这个想法。 如果以Canvas为基础，设计一个伪装的按钮组件又如何呢？听起来好像很不错，因为采用这种办法的话，我们对如何绘制组件的表面就有了完整的控制权。不过这也意味着你必须对按钮的状态进行手工维护。虽然Button本身是一个很简单的组件，但是重复去做标准按钮已经作好的工作似乎还是有点无谓。还有一件事情是应当考虑的：我们知道，JFace中的Action机制可以将标准按钮、菜单项和工具栏按钮这三种界面组件纳入一个统一的事件处理体系。然而，如果我们从Canvas派生去模拟一个按钮的话，不论你模拟到多么相似的地步，它毕竟不是一个真正的Button，Action也不会给它同等的待遇。也就是说手工制作的按钮无法和JFace Action体系协同工作除非你去修改Action的处理方法，让它去接纳新的按钮对象。这可不是一件轻松的工作。 如果上面的方法都行不通的话，应当怎么办呢？我们知道，和Swing这样的框架不同，SWT中的按钮其实就是操作系统底层所实现的按钮（这一点也可以用SPY 或者Winsight32之类的工具证实）。同时我们也知道，操作系统至少是Windows系统，对按钮已经提供了自我绘制的机制，这就是所谓的Owner Draw（称为所有者绘制的原因是因为默认情况下绘制消息是发送给按钮的父窗口处理的，但是父窗口也可以把这个皮球再踢回给按钮，让它自己解决）。在Win32 API中，凡是使用BS_OWNERDRAW风格创建、并且能够（通过消息反射）响应WS_DRAWITEM消息的按钮，都可以获得这种定制的能

力。 了解这一点，接下来的任务就是研究Button组件有没有开放这个接口供我们修改了。对Button组件的源代码进行粗略的浏览后，我发现了如下的方法：package org.eclipse.swt.widgets.public class Button extends Control { … LRESULT wmDrawChild (int wParam, int lParam) { if ((style &amp. SWT.ARROW) == 0) return super.wmDrawChild (wParam, lParam). DRAWITEMSTRUCT struct = new DRAWITEMSTRUCT (). .... 其中DRAWITEMSTRUCT结构的出现是一个明显的提示：这里就是WM_DRAWITEM消息的响应函数，很幸运它没有声明为final的，只要重载它并提供自己的实现就行了。 看起来是个小case，实际上也是。不过，还有一处小麻烦需要克服。注意wmDrawChild方法没有使用任何访问限定符，这意味着它是package friendly的同一个包中的对象可以访问和重载此方法，其他包中的对象就没有这个权力了。也就是说，要定制按钮对象，我们新建的对象也需要放在同一个包（org.eclipse.swt.widgets）中。看起来有点像在使用Hack手段，不过为了突破SWT给我们的限制，眼下也只好稍稍将就一下。好在swt的包没有密封（Sealed），不然我就不得不再次宣称此路不通了。 既然障碍已经扫清，接下来我们可以来实现前面的想法了。这里我做了一个决定，在上述包中只加入一个抽象类，目的是把必要的接口暴露出来；至于如何绘制按钮，则留给具体的按钮对象根据应用程序的需求来决定。这样，不管你希望实现Windows XP风格的按钮、还是卡通风格的按钮、或是平面样式的，总之不论什么千奇百怪的风格，只要继承一个类并重载一个绘制方法就行了，而不必每次都要和 Button类的内部打交道。 基于这种

考虑，实现自绘按钮的抽象类如下：package org.eclipse.swt.widgets.import org.eclipse.swt.internal.win32.*.public abstract class OwnerDrawButton extends Button{ public OwnerDrawButton(Composite parent, int style) { super(parent, style). int osStyle = OS.GetWindowLong(handle, OS.GWL_STYLE). osStyle |= OS.BS_OWNERDRAW. OS.SetWindowLong(handle, OS.GWL_STYLE, osStyle). } LRESULT wmDrawChild(int wParam, int lParam) { super.wmDrawChild(wParam, lParam). DRAWITEMSTRUCT struct = new DRAWITEMSTRUCT(). OS.MoveMemory(struct, lParam, DRAWITEMSTRUCT.sizeof). ownerDraw(struct). return null. } protected abstract void ownerDraw(DRAWITEMSTRUCT dis).}