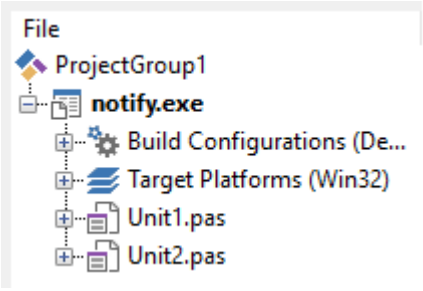
# QPlugins 示例分析第三章　Notify（通知）

## （一）通知系统的概念

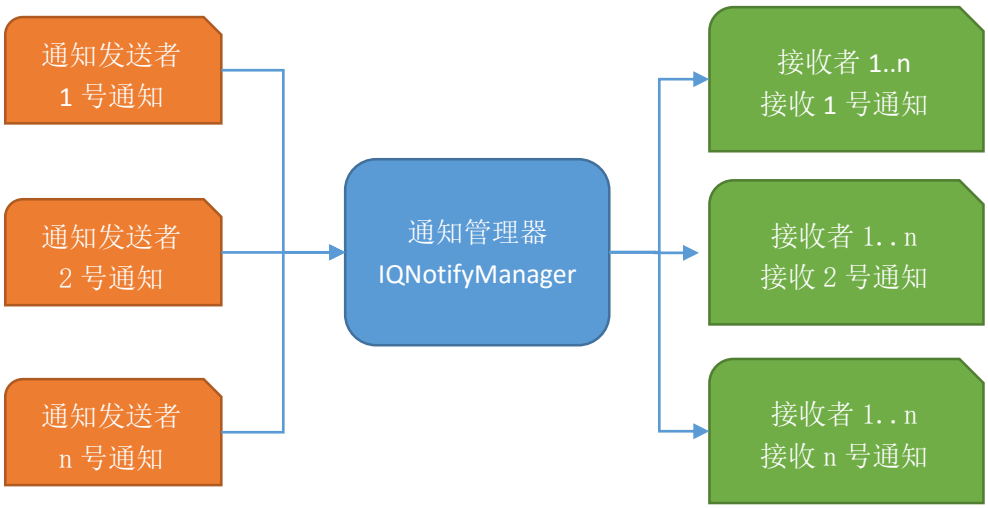
QPlugins 插件框架提供了一个方便的通知管理器接口，方便用户在不同的插件实例之间传递消息通知。我们通过它自带的 Nofity 示例程序来学习如何使用通知接口。

首先打开项目文件：




可以看到这个示例中并没有使用 dll 文件来演示插件，消息通知插件在同一 EXE 程序的不同窗体间演示。

在分析代码前，我们设想一下插件框架通知系统应该是什么样的。




在上图中，首先，通知发送者向通知管理器申请获得一个通知编号（想像成申请了一个广播频道，比如 1 频道）。第二步，通知接收者向通知管理器订阅通知编号（想像成用户向电视台购买了哪些频道）。第三步，通知发送者用申请到的通知编号（想像成通过什么频道），要

求通知管理器发送通知内容。第四步，通知管理器根据通知的编号，把通知内容分发到所有订阅了这个通知编号（频道）的用户。**在这样一个过程中，通知发送者与通知接收者之间互相并不用发生直接关系，也就是通过了通知管理器，实现了不同模块间的解耦**。

有了上面的大致概念，我们开始分析示例源代码，看它是如何工作的。

# （二）发通知的人

先分析 unit1 单元，也就是主窗口单元，它只引用了 QPlugins 和 Qplugins.qparams 两个插件单元，说明消息通知插件是 QPlugins 框架比较核心的功能。

```
unit Unit1;

interface

uses
  Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,
  System.Classes, Vcl.Graphics,
  Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.ComCtrls, QPlugins,
  QPlugins.qparams, Vcl.StdCtrls;
```

在窗体的定义中，有一个私有的变量，

```
type
  TForm1 = class(TForm)
    TrackBar1: TTrackBar;
    Label1: TLabel;
    Button1: TButton;
    procedure TrackBar1Change(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
    FChangeNotifyId: Integer;
  public
    { Public declarations }
  end;
```

这个 FChangeNotifyId 就是用来保存主窗口申请得到的通知编号（可以想像成频道编号）的。

在**主窗体的建立事件**中，首先向通知管理器申请得到了通知编号：

```
FChangeNotifyId := (PluginsManager as IQNotifyManager)
    .IdByName('Tracker.Changed');
```

可以看到，通过**(PluginsManager as IQNotifyManager）**转型操作，框架的插件管理器 **PluginsManager** 直接变身变成了通知管理器 **IQNotifyManager**。说明插件管理器核心同时支持插件管理接口和通知管理接口，根据需要随时变化，简直是多功能工具箱。
接着，通过 **IQNotifyManager** 接口的 **IdByName** 方法，传入一个容易理解的通知名称（比如

新闻联播），返回一个通知的唯一编号（比如 CCTV1 号频道）。

在主窗体的 TrackBar1Change 事件中，我们看到了发送通知的代码：

```
procedure TForm1.TrackBar1Change(Sender: TObject);
begin
  (PluginsManager as IQNotifyManager).Send(FChangeNotifyId,
    NewParams([TrackBar1.Position]));
end;
```

首先还是 PluginsManager 的变身技能 PluginsManager as IQNotifyManager，全局的插件管理器变量**以通知管理接口**方式出现。接着，调用接口的 **Send 方法**，首先接受的参数就是**通知编号**（刚才我们以'Tracker.Changed'名字申请得到的编号 **FChangeNotifyId**，这里不接受通知名字，只接受编号值）；第二个参数是我们真正要传送的通知内容，这里是 TrackBar1 的 Position 值，只不过需要用 NewParams 函数包装一下。因为 Send 函数的定义是这样的：

```
// 通知管理器
IQNotifyManager = interface
  ['{037DCCD1-6877-4917-A315-120CD3E403F4}']
  function Subscribe(ANotifyId: Cardinal; AHandler: IQNotify)
    : Boolean; stdcall;
  procedure Unsubscribe(ANotifyId: Cardinal; AHandler: IQNotify); stdcall;
  function IdByName(const AName: PWideChar): Cardinal; stdcall;
  function NameOfId(const AId: Cardinal): PWideChar; stdcall;
  procedure Send(AId: Cardinal; AParams: IQParams); stdcall;
  procedure Post(AId: Cardinal; AParams: IQParams); stdcall;
  function GetCount: Integer; stdcall;
```

可以看到，需要发送的通知内容都要以 IQParams 的形式包装起来。为避免干扰本次分析的主线，我们暂时放过 IQParams 到底是什么的研究，只要知道它是通知内容的包装方式即可（想像成发快递需要包装盒吧）。

至此，一个发送通知的发送者已经分析完了，接下来分析通知的接收者。

# （三）通知接收者

打开 unit2.pas，可以看到它也同样引用了 QPlugins 和 QPlugins.qparams 两个单元。

```
type
  TForm2 = class(TForm, IQNotify)
    Gauge1: TGauge;
    CheckBox1: TCheckBox;
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
    procedure CheckBox1Click(Sender: TObject);
  private
    { Private declarations }
    FNotifyId: Integer;
    procedure Notify(const AId: Cardinal; AParams: IQParams;
      var AFireNext: Boolean); stdcall;
  public
    { Public declarations }
  end;
```

作为接收者，它内部定义了一个 FNotifyID，用来保存它要接收的通知编号（想像成它购买的收费频道编号吧）。还定义了一个 Notify 过程，以实现 IQNotify 接口，用来在接到通知时执行一些处理操作。

那么，首先它就要向通知管理器来申请要接收的通知编号（想像成订阅某个收费频道），本例中，这个操作在窗体的建立事件中完成：

```
procedure TForm2.FormCreate(Sender: TObject);
var
  AMgr: IQNotifyManager;
begin
  AMgr := PluginsManager as IQNotifyManager;
  // 注册通知，事件的名称不需要特意注册，IdByName 在名称不存在时，会自动注册并生成一个新的 ID 返回
  // 多个不同的 Id 就多次订阅即可
  FNotifyId := AMgr.IdByName('Tracker.Changed');
  AMgr.Subscribe(FNotifyId, Self);
end;
```

这里，用了一个 AMgr 变量，用来暂时保存对插件管理器变型后的通知管理器的引用，减少后面敲代码的数量。

接着，通过 IdByName 方法，向通知管理器查询某个名字的通知对应的编号。

然后，我们以这个通知编号为参数，调用 Subscribe 方法向通知管理器正式订阅 AMgr.Subscribe(FNotifyId, Self); 订阅时还得把自己的身份证告诉通知管理器，不然它就不知道谁订阅了通知，所以第二个参数是 self。至此，这个窗体的身份证（也就是 self 指针）已经在通知管理器完成注册，而且它告诉了通知管理器需要接收哪个（或哪些）通知。实际上，通过多次调用 Subscribe，同一个接收者可以订阅好几个不同类型的通知。

完成订阅后，我们就可以坐等通知上门了。由于我们已经把自己的地址告诉了通知管理器，

在有通知时，通知管理器会逐个向订阅者发送通知。还记得本窗体在定义时还实现了 IQNotify 接口吗？

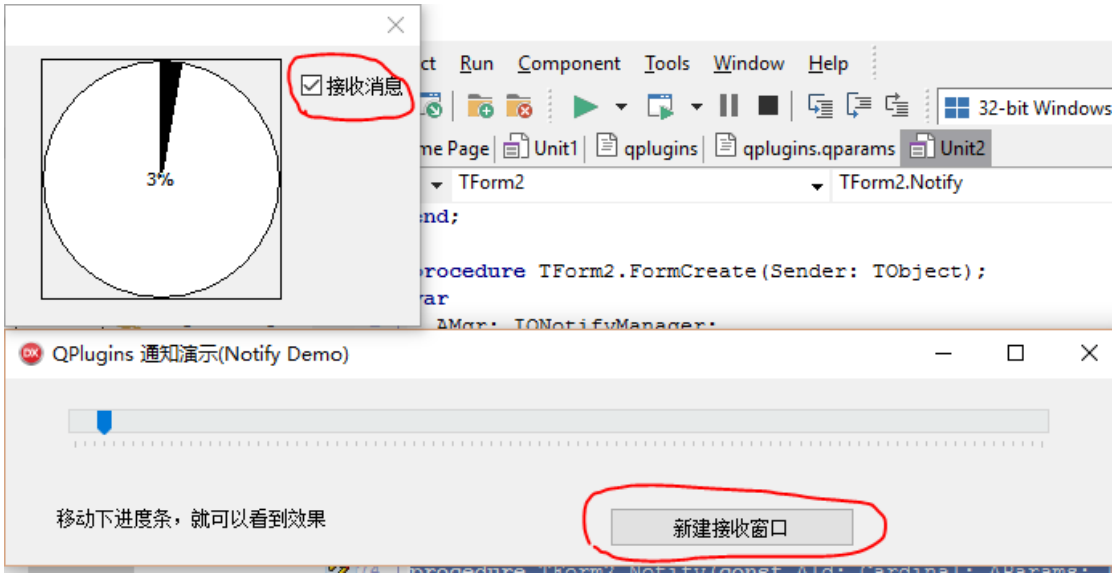
```
TForm2 = class(TForm, IQNotify)
```

**通知管理器要求每个订阅者都要有一个 IQNotify 接口并实现接口方法 Notify**。如果不能理解，想像一下你家里的数字电视接口设备吧，你要接收通知至少得有一个接收器的。

```
procedure TForm2.Notify(const AId: Cardinal; AParams: IQParams;
  var AFireNext: Boolean);
begin
  if AId = FNotifyId then //  多个通知关联到同一个对象时，通过 AId 进行进行区分
  begin
    if not Visible then
      Show;
    Gauge1.Progress := AParams[0].AsInteger;
  end;
end;
```

这里，我们只关心 **FNotifyId 和 AParams，FNotifyId** 是我们关心的通知编号。前面说过，一个接收者可以订阅多个通知，在这里我们就可以判断通知的编号来区别对待处理。**AParams** 里面是传递的通知内容。

到此，发送者、接收者和通知管理器之间的故事就讲完了。看一下运行效果：




红圈部分是本人对示例程序修改的内容。接下来结合一些问题，来讲讲这些补充的内容。

## （四）通知的退订

我们有时订阅了某个收费内容后，也会偶尔不想要而退订。通知管理器也允许一个接收者取

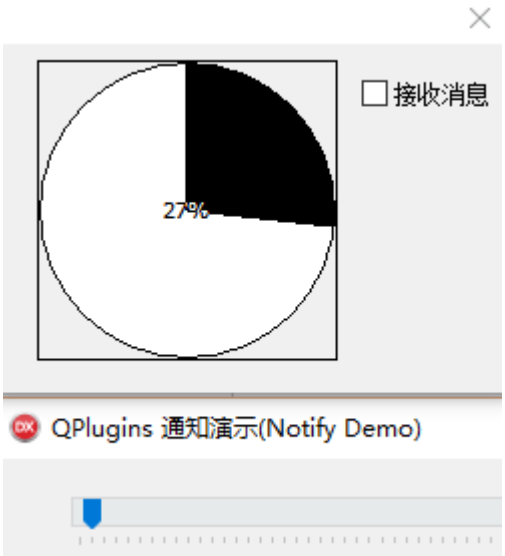
消订阅，方法是调用 unSubscribe 方法。看下面我在示例中添加的演示代码：

```
procedure TForm2.CheckBox1Click(Sender: TObject);
var
   AMgr: IQNotifyManager;
begin
   AMgr := PluginsManager as IQNotifyManager;
   // 注册通知，事件的名称不需要特意注册，IdByName 在名称不存在时，会自动注册并生成一个新的 ID 返回
   // 多个不同的 Id 就多次订阅即可
   FNotifyId := AMgr.IdByName('Tracker.Changed');

   if CheckBox1.Checked    then
       AMgr.Subscribe(FNotifyId, Self)
   else
      AMgr.unSubscribe(FNotifyId, Self);

end;
```

在接收窗体中，我添加了一个复选框 CheckBox1，允许用户订阅或取消订阅。在取消订阅时，它就无法接收到主窗体的变化通知了，如下图所示：




## （五）多个接收者

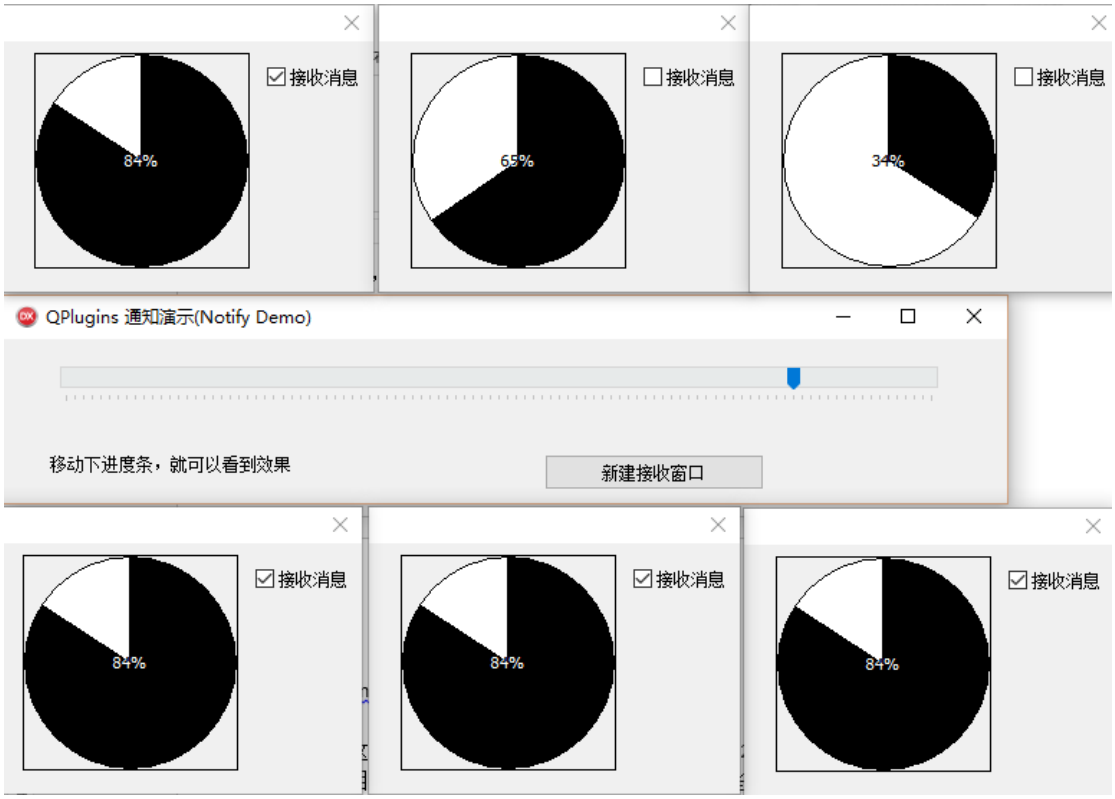
一个通知可以发送到许多个接收者。所以我增加了一段代码来演示。在主窗体中添加一个按钮，标题为“新建接收窗口”，然后写如下事件代码：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
      TForm2.Create(application);
```

```
end;
```

这里，主窗体所在的 unit1 单元需要引用 unit2.pas，破坏了通知系统能实现不同单元解耦的目的。但作为一个简单演示，暂时不讲究这些。这段代码的作用是每按一次按钮，就新建一个接收者窗口。每个接收者窗口我们都已经添加了一个复选框，可以控制订阅或退订。

现在重新运行示例代码，并且多按几次这个新按钮，再拖动进度条。我们发现还是只有一个接收窗体出现呀，为什么？其实是因为窗体都显示在同一位置，手动拖动一下位置就可以看到后面的窗体了。下面是运行界面，当然是手动调整接收窗体位置的结果 。




注意上面第二第三个窗体没有勾选“接收消息”，它们在不同的时间点取消了订阅，所以没有同步到最新的消息状态。

## （六）接收者的生命期

有时候，一个接收者订阅了通知，但是我们可能在某个地方把它给 Free 了，此时通知管理器却不一定知道，它依然用接收者**生前**向它注册的地址发送通知（还记得接收者在订阅时递交的那个 self 吗？），这就会产生内存访问冲突（你拨打的电话已关机？）。

所以，作为接收者，要在自己结束生命期的时候，主动退订通知，做一个和谐好公民。
为此，我们在接收者窗体 unit2.pas 中添加了这么一个代码：

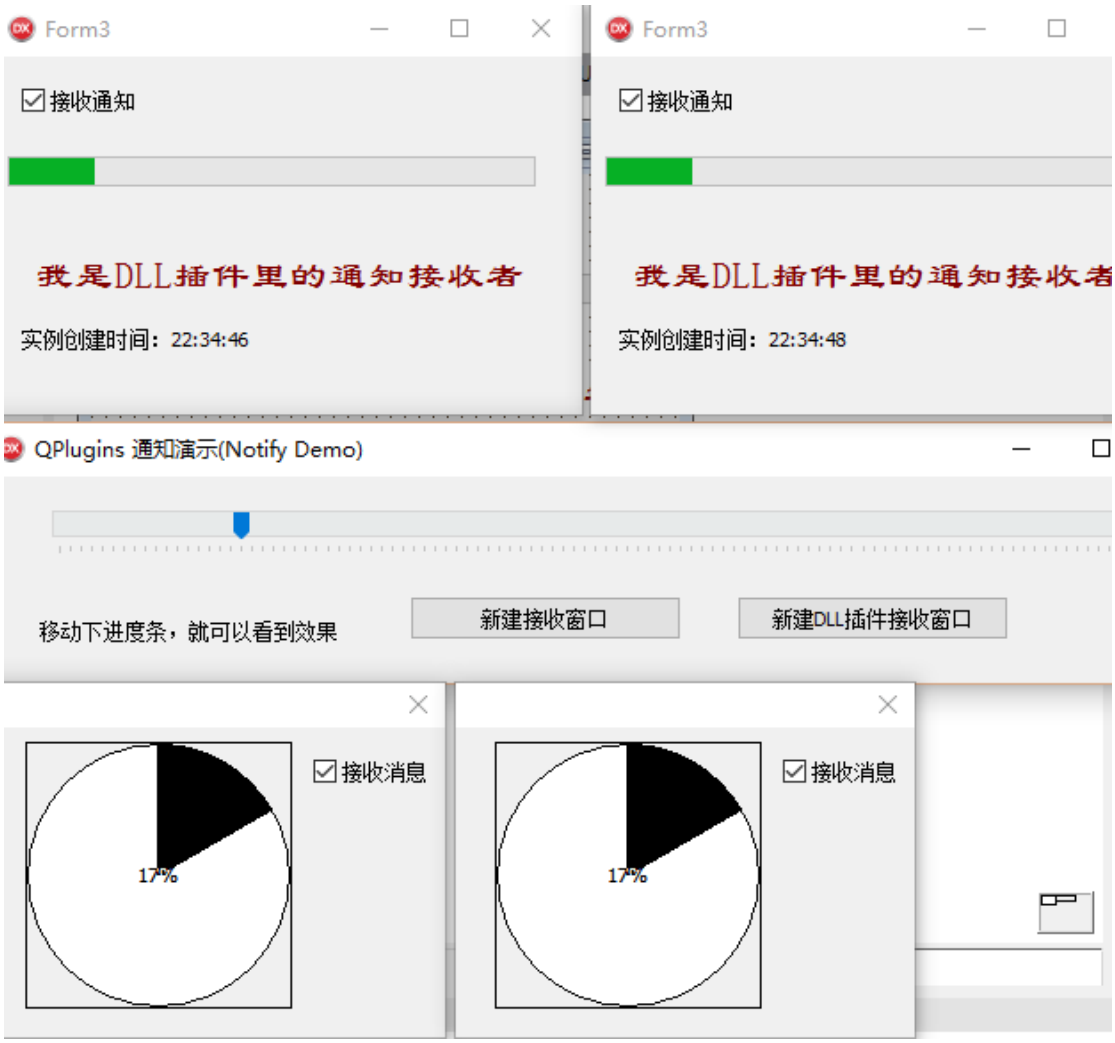
```
procedure TForm2.FormClose(Sender: TObject; var Action: TCloseAction);
var
  AMgr: IQNotifyManager;
begin
  AMgr := PluginsManager as IQNotifyManager;
  // 注册通知，事件的名称不需要特意注册，IdByName 在名称不存在时，会自动注册并生成一个新的 ID 返回
  // 多个不同的 Id 就多次订阅即可
  FNotifyId := AMgr.IdByName('Tracker.Changed');
  AMgr.unSubscribe(FNotifyId, Self);
  action := CaFree;
end;
```

这样，当一个接收者不存在的时候，就不会发生依然被通知的情况。

**（备注：如果接收窗体不想在关闭的时候直接 Free 掉，那么在窗体的 FormDestroy 里写上退订过程，会更加合适）**

# （七）不同模块里的接收者

这个示例时的发送者和接收者在同一个 EXE 文件里，可能有人会觉得多此一举，直接调用 unit2 里的相关单元不是更简单？所以，我增加一个 DLL 插件里的接收者来作为演示。
具体见修改过的 Notify 示例程序，基本是综合了前面 2 章里的插件框架知识和这章的通知框架内容，暂时不作更多讲解了。看运行效果：


Form3
接收通知
我是DLL插件里的通知接收者
实例创建时间：22:34:46
Form3
接收通知
我是DLL插件里的通知接收者
实例创建时间：22:34:48
QPlugins 通知演示(Notify Demo)
移动下进度条，就可以看到效果
新建接收窗口
新建DLL插件接收窗口
接收消息
17%
接收消息
17%