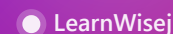


Risorse incorporate e resource.wx

Guida della lezione · Guida al lab / esame



Guida della lezione

Distribuire le immagini dentro l'assembly: Build Action impostata su Embedded Resource, l'handler resource.wx con e senza qualificatore di assembly, le estensioni css, js, html, jpg, jpeg, png, gif, svg e bmp che serve, e il file distribuito che può sovrascrivere una risorsa incorporata senza ricompilare. Che cosa significa per uno sviluppatore che conosce le basi di C# e ha già lavorato un po' con i controlli e il designer di Wisej.NET — e come affrontare questo modulo.



Focus di produzione

Spedire le immagini dentro l'assembly: la Build Action impostata su Embedded Resource, l'handler resource.wx con e senza qualificatore di assembly, le estensioni css, js, html, jpg, jpeg, png, gif, svg e bmp che accetta di servire, e il file distribuito che può sovrascrivere una risorsa incorporata senza ricompilare.

1. Il problema in produzione

Fino a qui ogni asset di IconDesk è stato qualcosa che l'applicazione ha trovato su disco o ha chiesto al tema. Funziona finché qualcuno non distribuisce il sito dimenticando una cartella: il browser chiede un'icona, il server risponde 404 e l'utente vede un rettangolo vuoto dove dovrebbe esserci il badge di stato. In revisione non se ne accorge nessuno, perché sulla macchina di sviluppo il file c'è ancora.

Incorporare trasforma la grafica in parte dell'output di compilazione: l'immagine viaggia con l'assembly, versionata dallo stesso numero di build, impossibile da perdere in un passaggio di copia. È anche il meccanismo su cui è costruito ogni pacchetto di icone Wisej.NET, ed è ciò che rende possibile il Modulo 6.

2. Concetti chiave

Concetto	Perché è importante
Build Action = Embedded Resource	La proprietà di file di Visual Studio che compila un file dentro il manifest dell'assembly invece di copiarlo accanto all'eseguibile, così l'asset viene spedito dentro la DLL.
handler resource.wx	L'handler HTTP di Wisej.NET che serve al browser le risorse incorporate; quello che assegna è una stringa di sorgente immagine, mai un oggetto immagine lato server.
URL di risorsa non qualificato	<code>resource.wx/company-Logo.svg</code> chiede a Wisej.NET di cercare fra gli assembly caricati una risorsa con quel nome: comodo e leggermente ambiguo.
URL di risorsa qualificato con l'assembly	<code>resource.wx/MyCompany.Branding/company-Logo.svg</code> indica l'assembly (senza <code>.dll</code>) così che la ricerca non possa finire sul file omonimo di qualcun altro.
Tipi di risorsa consentiti	L'handler documenta esattamente nove estensioni: <code>.css</code> , <code>.js</code> , <code>.html</code> , <code>.jpg</code> , <code>.jpeg</code> , <code>.png</code> , <code>.gif</code> , <code>.svg</code> e <code>.bmp</code> ; qualsiasi altra cosa torna indietro come 404.
Override tramite file distribuito	Un file distribuito insieme all'applicazione può essere servito al posto della risorsa incorporata allo stesso percorso, il che permette a un'installazione di ri-brandizzare o correggere al volo la grafica senza ricompilare l'assembly.
Nome della risorsa nel manifest	Il nome che .NET memorizza davvero, derivato dal nome dell'assembly, dal namespace predefinito e dalla cartella in cui si trova il file; tirarlo a indovinare è il motivo abituale per cui un URL di risorsa non restituisce nulla.

3. Il filo della lezione

Incorporare è una Build Action, non una chiamata API

Non c'è nessun metodo da chiamare e nessun attributo da applicare. Aggiungi il file al progetto, apri la Properties window e porti la Build Action dal suo valore predefinito a **Embedded Resource**. Il compilatore scrive i byte nel manifest dell'assembly, il file smette di comparire nella cartella di output e da quel momento l'asset fa parte del binario. Nel lab lo fai cinque volte: `logo.svg`, `status-ok.svg`, `status-warning.svg`, `photo.png` e `badge.gif`, tutti dentro una cartella `Resources` nell'applicazione IconDesk.

La stessa proprietà funziona in una class library, ed è questo il caso interessante. Una libreria il cui unico compito è portare la grafica è un sacco di icone ridistribuibile: referenziale, e ogni schermata che la referencia può indirizzare quei file. I pacchetti di icone Wisej.NET sono esattamente questo, diverse centinaia di SVG in un unico assembly, e il Modulo 6 te ne fa costruire uno tuo. Nella tecnica non cambia nulla fra un'applicazione e una libreria; cambia solo il nome che potresti dover mettere nell'URL.

Mettilo a confronto con le alternative. Un file Content con *Copy to Output Directory* resta sciolto: facile da sostituire, facile da perdere. Un URL esterno sposta il problema su un altro host. Incorpora quando l'asset appartiene al codice e deve essere versionato con esso; la decisione è operativa, non stilistica.

Indirizzarlo: resource.wx con e senza il nome dell'assembly

Una volta incorporato il file, il browser lo raggiunge attraverso l'handler `resource.wx`. Il valore che assegna a `ImageSource` è una stringa, e il client la risolve come URL; il server non materializza mai un `System.Drawing.Image` per essa. È una distinzione che conta soprattutto per l'SVG, che non è affatto un formato `System.Drawing`: è markup che disegna il browser, ed è esattamente per questo che resta nitido a 16, 24 o 96 pixel.

```
// IconDesk - ResourceLab.cs
// ogni file qui sotto ha Build Action = Embedded Resource

// non qualificato: Wisej.NET cerca fra gli assembly caricati
picStatusOk.ImageSource = "resource.wx/status-ok.svg";
picPhoto.ImageSource    = "resource.wx/photo.png";
picBadge.ImageSource     = "resource.wx/badge.gif";

// qualificato con il nome dell'assembly, senza ".dll" - nessuna ambiguità
picLogo.ImageSource      = "resource.wx/IconDesk/logo.svg";

// lo stesso URL funziona fuori dalle proprietà dei controlli, ad es. nel CSS:
// .brand-mark { background-image: url('resource.wx/IconDesk/logo.svg'); }
```

Ometti il segmento dell'assembly e Wisej.NET cerca fra gli assembly caricati una risorsa corrispondente. Va bene finché il nome appartiene a un solo assembly. Aggiungi una seconda libreria di icone, o l'assembly di branding di un cliente, e due file chiamati `logo.svg` possono rispondere entrambi. Qualificare il percorso elimina la domanda e documenta la proprietà per chi legge dopo di te. Il lab ti fa qualificare esattamente una sorgente, così che la differenza sia visibile nel tuo stesso codice.

Poiché l'handler parla semplice HTTP, la stessa forma di URL può comparire in un foglio di stile, in un frammento di HTML o in un pezzo di JavaScript, non solo in una proprietà immagine. Nota anche che l'attributo `[assembly: WisejResources]` appartiene a un discorso più ampio di scoperta e bundling per script, stili e temi incorporati; un URL di risorsa funziona con o senza di esso, quindi non aggiungere attributi meccanicamente solo per mostrare un'immagine.

Che cosa l'handler serve e che cosa no

L'elenco documentato è breve e vale la pena impararlo a memoria, perché il modo in cui fallisce è silenzioso. L'handler serve `.css`, `.js`, `.html`, `.jpg`, `.jpeg`, `.png`, `.gif`, `.svg` e `.bmp`. Sei delle nove sono immagini; le altre tre esistono perché un assembly di libreria porta spesso un foglio di stile o uno script accanto alla propria grafica. Tutto il resto — un file di icona, un font, un PDF — torna indietro come 404 per quanto correttamente sia stato incorporato.

Quindi l'estensione è una decisione presa prima che il file entri nel progetto: un logo vettoriale diventa `.svg`, una fotografia `.png` o `.jpg`, un marcatore animato `.gif`. Se qualcuno ti consegna un formato che l'handler non serve, convertirlo fa parte del lavoro.

Quando vince un file distribuito, e come diagnosticare un 404

Wisej.NET permette a un file distribuito insieme all'applicazione di avere la precedenza sulla risorsa incorporata allo stesso percorso logico. È voluto ed è utile: l'installazione presso un cliente può metterci il

proprio `logo.svg` e l'applicazione si ri-brandizza da sola senza che nessuno ricompili o rilasci di nuovo l'assembly delle risorse, e la stessa via corregge al volo un'icona sbagliata. Il lab ti fa eseguire un override e catturare il prima e il dopo.

Il rischio è che la directory di distribuzione entri ora a far parte del tuo confine di fiducia: chiunque possa scriverti dei file può cambiare ciò che l'applicazione mostra. Il secondo rischio è la deriva, perché l'applicazione in esecuzione non corrisponde più all'assembly da cui è stata compilata. Tratta la possibilità di override come configurazione, con un responsabile e un registro.

Quando un'immagine non compare, non tirare a indovinare i nomi nel manifest. Percorri invece un elenco fisso; ogni passo esclude una causa:

- Conferma che la Build Action sia Embedded Resource, non Content o None.
- Conferma che l'estensione sia una delle nove che l'handler serve.
- Conferma che l'assembly sia davvero caricato nel processo in esecuzione.
- Conferma l'assembly e i segmenti del percorso nell'URL, carattere per carattere.
- Controlla se un cambio di namespace predefinito, di cartella o di nome logico ha spostato il nome della risorsa nel manifest.
- Ricompila, poi richiedi l'URL esatto nel browser e leggi il codice di stato.



Dai prima un nome al meccanismo

Prima di assegnare qualsiasi immagine, dichiara quale meccanismo stai usando: un oggetto `System.Drawing.Image` che il server disegna come PNG, una stringa di sorgente immagine che il browser risolve, un'immagine del tema indicata per nome, una risorsa incorporata `resource.wx` o un font di icone disegnato come HTML. Assegna una sola proprietà immagine per controllo, così che le altre non possano azzerarla di nascosto, mantieni la stringa della sorgente significativa invece di un percorso magico incollato, preferisci l'SVG per le icone di interfaccia così che restino nitide a ogni dimensione e scala, e dai a ogni icona che porta un significato un'alternativa testuale.



Obiettivi di apprendimento

Alla fine di questo modulo sarai in grado di:

- ✓ Incorporare file di immagine supportati in un assembly di applicazione o di class library.
- ✓ Usare `resource.wx` con e senza il nome dell'assembly.
- ✓ Elencare i tipi di risorsa supportati dall'handler delle risorse incorporate di Wisej.NET.
- ✓ Spiegare come i file distribuiti possano sovrascrivere le risorse incorporate e perché può essere utile.

Guida al lab / esame

Che cosa costruirai nel lab pratico, l'approccio suggerito e i deliverable richiesti.



Lab pratico

Aggiungi all'applicazione IconDesk una cartella Resources che contiene logo.svg, status-ok.svg, status-warning.svg, photo.png e badge.gif, e imposta su ognuno di essi la Build Action Embedded Resource. Costruisci una pagina ResourceLab che li mostra tutti e cinque attraverso sorgenti immagine resource.wx, scrivi una di quelle sorgenti con il nome dell'assembly dell'applicazione davanti al file così che la ricerca non possa essere ambigua, e lascia le altre non qualificate. Poi metti un file sostitutivo accanto all'applicazione distribuita per sovrascrivere una risorsa incorporata e cattura il prima e il dopo. Concludi sbagliando di proposito la scrittura di un URL di risorsa e annotando i passi che hanno individuato il problema.

Punto di partenza

Apri IconDesk come si trova dopo il Modulo 4: il pannello dei comandi, la pagina ImageLab, la IconGallery costruita con il designer e la pagina IconCompare che mette affiancati due pacchetti di icone ufficiali. Finora ogni asset arriva dal tema, da un file sciolto del progetto, da un URL o da un assembly di pacchetto; nulla viaggia ancora dentro il binario di IconDesk stesso, e nel progetto non c'è nessuna cartella

`Resources` .

Approccio suggerito

- ✓ Aggiungi prima la cartella `Resources` e i cinque file, poi imposta la Build Action su Embedded Resource su ciascuno e ricompila prima di scrivere una riga di C#.
- ✓ Aggiungi una pagina `ResourceLab` con cinque controlli PictureBox o Label e assegna a ogni `ImageSource` una stringa `resource.wx` , qualificando solo il logo con il nome dell'assembly `IconDesk` .
- ✓ Esegui la pagina, apri il pannello di rete del browser e conferma che ogni asset restituisce 200 e che gli SVG restano nitidi quando li ingrandisci.
- ✓ Pubblica o copia l'applicazione in una cartella di distribuzione, metti un file sostitutivo sopra un asset incorporato, ricarica senza ricompilare e cattura uno screenshot di entrambi gli stati.
- ✓ Rompi di proposito un URL, poi percorri nell'ordine l'elenco di troubleshooting e annota quale passo ha rivelato la causa.

Deliverable richiesti

- ✓ Cartella Resources con logo.svg, status-ok.svg, status-warning.svg, photo.png e badge.gif tutti marcati Embedded Resource
- ✓ Pagina ResourceLab che mostra ogni asset incorporato tramite una sorgente immagine resource.wx
- ✓ Una sorgente immagine qualificata con il nome dell'assembly dell'applicazione e le altre lasciate non qualificate

- ✓ File distribuito che sovrascrive una risorsa incorporata, catturato prima e dopo
- ✓ Nota di troubleshooting che elenca i passi con cui è stato individuato un URL di risorsa scritto male

Domande di autovalutazione

- ✓ Due assembly referenziati incorporano entrambi un file chiamato `logo.svg`. A che cosa si risolve una sorgente non qualificata, e che cosa cambieresti perché la risposta smetta di dipendere dall'ordine di caricamento?
- ✓ Il tuo override funziona sul server di test ma non dopo il rilascio successivo. Quali parti del processo di deploy potrebbero averlo rimosso, e dove avrebbe dovuto essere registrato?
- ✓ Un collega incorpora `icon.ico` e segnala che `resource.wx` è rotto. Che cosa sta succedendo davvero, e qual è la modifica più piccola che lo risolve?



Dai prima un nome al meccanismo

Prima di assegnare qualsiasi immagine, dichiara quale meccanismo stai usando: un oggetto `System.Drawing.Image` che il server disegna come PNG, una stringa di sorgente immagine che il browser risolve, un'immagine del tema indicata per nome, una risorsa incorporata `resource.wx` o un font di icone disegnato come HTML. Assegna una sola proprietà immagine per controllo, così che le altre non possano azzerarla di nascosto, mantieni la stringa della sorgente significativa invece di un percorso magico incollato, preferisci l'SVG per le icone di interfaccia così che restino nitide a ogni dimensione e scala, e dai a ogni icona che porta un significato un'alternativa testuale.