Creazione di un gwWidget

Supponiamo di voler creare un nuovo widget di Geoweb: NeoWidget. Ecco i passi necessari:

Creazione class Java

Per prima cosa serve una class Java che definisca le caratteristiche del widget. Questa deve:

- estendere WidgetBase, o un'altra class derivata da essa.
- fare l'override della variabile CTRL_TYPE, scegliendone uno univoco per il widget.
- e' inoltre buona norma dare un alias ad XStream per la serializzazione (con @XStreamAlias), e marchiare i campi non necessari, per esempio i final static (XStreamOmitField)

```
    package com.geowebframework.transfer.model.widget;
    ...
    @XStreamAlias("NeoWidget")
    public class NeoWidget extends WidgetBase {
        @XStreamOmitField public final static int CTRL_TYPE = 1000;
//unique
        private String customProperty;
        public NeoWidget () {
            super();
        }
        ...
        //setters and getters customProperty, and eventual ovverrides
      }
```

Al bisogno si possono ovviamente fare gli override di variabili e metodi della superclasse. La classe puo trovarsi in un package qualsiasi. Per convenzione, i widget del modulo base di Geoweb, sono posizionati sotto *com.geowebframework.tranfer.model.widget*. In caso di widget definito dentro un plugin la convenzione è di metterlo sotto *com.geowebframework.plugin_name.model.widget*.

Per la scelta del CTRL_TYPE di widget sul modulo base, aumentare di 1 unita il valore presente nella riga commentata / / max ctrlTypeStandard=X, ed aggiornarlo.Per i widget dei plugin esistono intere serie dedicate (Es: 100-199 per plugin gwenterprice[ex calendar]; 200-299 per plugin ThreeDVisualizer). Comunque sia in caso si stia tentando di gestrare un widget con un CTRL_TYPE gia usato, viene generata un'opportuna eccezione.

Registrazione widget

Una volta creata la class del widget, dobbiamo registrarlo in un'apposita class di registro: gwRegistry.

```
    @Autowired
    private GwRegistry gwRegistry;
    ...
    gwRegistry.registerWidget(
```

```
        new CtrlTypeInfo(
            NeoWidget.CTRL_TYPE, "NeoWidget", "NEO_WIDGET",
            com.geowebframework.transfer.model.widget.NeoWidget.class,
            "neoWidgetVisualize","neoWidgetEdit",
            new Boolean(false), new Boolean(false), 650, 300
        ),
        new ArrayList<GwAttributeDataType>(
            Arrays.asList(new
GwAttributeDataType[]GwAttributeDataType.UNDEFINED}))
        )
    );
```

Questo viene fatto tramite il metodo registerWidget(CtrlTypeInfo ctrlTypeInfo, ArrayList<GwAttributeDataType> relatedDataType). Questo ha due argomenti:

- ***ctrlTypeInfo***, raccoglie informazioni riguardo al gwWidget, esso richiede nel costruttore i seguenti parametri, in ordine di uso:
  - ***ctrlType***: identifica il widget e viene persistito nella GWM_ATTRIBUTES.
  - ***className***: nome della class del widget.
  - ***alias***: alias del widget. Usato per farvi riferimento nelle interfacce del Geoadmin.
  - ***ctrlParamClass***: class Java del widget, serve per persistere una sua istanza nella GWM_ATTRIBUTES (come stringa xml serializzata tramite XStream).
  - ***detailJspName***: nome della pagina .jsp usata quando il gwClassDetail è in visualizzazione, (eventualmente comprensiva di path a partire dalla cartella jsp/commons/widget).
  - ***editJspName***: nome della pagina .jsp usata quando il gwClassDetail è in edit, (eventualmente comprensiva di path a partire dalla cartella jsp/commons/widget).
  - ***fullSizeWidget***: flag di utilità per il creatore della struttura della form. Va posto a true se il widget occuperà tutto lo spazio in larghezza (Es: ChildList)
  - ***hasCustomEditor***: flag che denota se il widget ha un editor ad hoc nell'Geoadmin.
  - ***editorDialogWidth***: width in pixel del dialog dell'editor del Geoadmin.
  - ***editorDialogHeight***: height in pixel del dialog dell'editor del Geoadmin.
- ***relatedDataType***: ArrayList<GwAttributeDataType> contenente oggetti della enum GwAttributeDataType. Serve per poter determinare nel Geoadmin, durante la configurazione di un gwAttribute, con quali tipologie di data type sarà possibile associare un widget di questo tipo.

I widget del progetto base (*webclient*) di Geoweb vengono registrati dentro il costruttore di gwRegistry. Se il widget sta invece in un plugin di Geoweb tale operazione va fatta in una class java di servizio annotata *@Service*, precisamente in un metodo annotato *@PostConstruct*. [La covenzione è che il nome della classe sia *camelCase(nome_plugin)+"InitializerService*", ed la signature del metodo sia *public void init()*].

```
    @Service
    public class NameOfPluginInitializerService{
        ...
        @PostConstruct
        public void init(){
            ...
```

```
        }
    }
```

Creazione pagine .jsp di visualizzazione e di modifica

E' necessario ora creare le pagine ***detailJspName*** e ***editJspName*** dichiarate nel costructor del CtrlTypeInfo alla registrazione del widget. Il contenuto di tali pagine jsp è libero, ma sono considerate buone prassi:

- applicare al widget dojo principale (od al più esterno, in caso di widget dalla struttura complessa) le classi, le proprietà e le implementazioni delle function delle shared js API, come descritto sotto
- per i widget che occupano tutto lo spazio in orizzontale (nascondendo la label di sinistra) adottare nel widget di layout piu esterno il parametro ***colspan*** valorizzato a ***2***.

Aggiunta eventuali risorse .css e .js usate nelle .jsp

Per il modulo base le classi css sono generalmente poste sotto il file /css/gwClassDetail.css, mentre i js sono sparsi in vari file, essendo richiamate spesso function di carattere generale. Per quanto riguarda risorse .css e .js ad uso esclusivo di un plugin, esistono due metodi di utilità per importarele in automatico. Metodi che vanno in fase di inizializzazione del plugin.

```
@Service
public class NameOfPluginInitializerService{
    ...
    @PostConstruct
    public void init(){
    ...
        gwRegistry.addJsResource("debug/js/plugin.js");
        gwRegistry.addCssResource("css/pluginRules.css");
    }
 }
```

Le risorse importate in tale modo vengono aggiunte all'inizio all'apertura del progetto Geoweb. Vengono però aggiunte solo dopo le risorse usate dal modulo base (e dopo il caricamento del modulo dojo, per quanto concerne i js).

Creazione eventuali Controller lato server

In generale un widget non ha bisogno di Controller java per il suo funzionamento. Talvolta capita che un widget, per necessità di recuperare ulteriori dati, sia costretto a fare un passaggio lato server per poter renderizzare una qualche sua parte. Tali controller sono per convenzione messi in apposite classi sotto *com.geowebframework.webclient.controller.widget* per il modulo base, mentre *com.geowebframework.plugin_name.controller.widget* per i plugin.

Integrare il widget nelle logiche del dettaglio di classe

I passi necessari sono:

- far dichiarare dal widget al contesto che esso è 'pronto', pubblicando un apposito evento dojo
- assegnare le class css come da specifiche
- implementare le shared JS API

Tutti questi passaggi sono trattati nello specifico qui: EventAttributes Guidelines