OpenLaszlo:
Come caricare DATI

Punto della situazione

- Abbiamo visto come:
  - definire il **LAYOUT** di una interfaccia *(LZX, tag, attributi ...)*
  - programmare la sua **LOGICA** *(JavaScript, method, handler ...)*
- Fino a questo punto, tuttavia, l'interfaccia risulterà poco utile, in quanto è **statica**
- Serve un metodo per renderla **dinamica**
- Analogo alla differenza che c'è in HTML tra *pagine statiche e dinamiche (PHP, JSP...)*

Il tag <dataset>

- E’ il tag che **permette di “comunicare”** con qualsiasi server nel back-end:

- Lo scambio dati avviene in **XML** (è obbligatorio avere un solo nodo radice nel documento XML)
- Il tag *<dataset>* punterà ad uno **script che genererà l’XML** che descrive i dati richiesti

- I dati possono essere caricati:
  - a **compile-time** (direttamente nel codice)

```
<dataset>
<myXML>
    <person surname=”Ferrari” name=”Stefano” sex=”M” />
    <person surname=”Lovato” name=”Laura” sex=”F” />
<myXML>
</dataset>
```

  - a **compile-time** (da file di testo)

```
<dataset src="myXMLDoc.xml" />
```

  - a **run-time** (da sorgente HTTP esterna)

```
<dataset type=”http” src="myXMLDoc.xml" />
```

NB: in questo caso si possono verificare problemi nella trasmissione!!

- Nel caricamento a **run-time**, come di consueto, **si devono gestire gli eventi**:
  - **ondata()**
    Generato al momento della ricezione corretta dei dati (dati arrivati e corretti!)
  - **onerror()**
    Generato in caso di errore durante il caricamento dei dati oppure nel caso di ricezione dei dati in formato errato
  - **ontimeout()**
    Generato al passare di un certo intervallo di tempo senza che i dati siano stati ricevuti (default: 30 s)

- ☆ Tale codice **non può essere scritto** dentro il tag dataset stesso (verrebbe interpretato come XML)

- Esempio:

```
<dataset>
   <handler name="ondata">
   ..............
   </handler>
</dataset>
```

Il codice di gestione degli eventi **non può essere messo qui**: verrebbe interpretato come XML!!

- Soluzione 1: usare gli *handler inline*

```
<dataset  ondata="...codice..."  onerror="...."  ontimeout="..."> 
..................
</dataset>
```

- Soluzione 2: usare l'attributo *reference*

```
<dataset name="myDATA"> ..................</dataset>

<handler  name="ondata"  reference="canvas.myDATA">
..................
</handler>
```

Come caricare i dati

- L'attributo **src** del tag <dataset> indica l'URL da cui caricare i dati
- I dati vengono caricati in **modo asincrono** (cioè l'interfaccia non si blocca ad attendere i dati)
- I dati sarano caricati da questo URL:
  - **in automatico**, al lancio dell'applicazione oppure ogniqualvolta viene modificato l'attributo *src*, se l'attributo *request* è settato a *true* *(default: false)*
  - **esplicitamente**, con il metodo *doRequest()* del dataset

- Le richieste possono essere **GET** o **POST**:
  (differenti modalità per passare i parametri al server)
  - nella modalità **GET** i parametri vengono passati nella *querystring* dell'url che punta allo script
    (es: www.miosito.com/news.php?lang=it&section=latest)
  - con **POST**, invece, vengono passati per mezzo di un classico *web-form*
- ★ GET è pratico e veloce, ma i valori della querystring sono intercettabili!!!
  es: www.miosito.com/login.php?user=ale&passwd=pippo)
- ★ POST è più macchinoso, ma sicuro

Tag <dataset>: dettagli

- **Attributi**

  src, timeout, rawdata
  request, autorequest, multirequest
  querystring, querytype, params

- **Metodi**

  doRequest(), abort(), setQueryParam(), setQueryString()
  getErrorString(), getLoadTime()
  getPointer()

- **Eventi**

  ondata(), onerror(), ontimeout()

Come accedere ai dati

- Il modo più semplice (detto "*DataPointer model*") consiste nello **specificare il percorso al dato**:
  - tramite l'attributo **datapath** di un oggetto
    In questo modo **si "lega" quell'oggetto ad un particolare nodo del documento XML**, dandone il percorso (path) secondo una sintassi ben definita *(standard XPath)*
  - tramite dei puntatori (tag **<datapointer>**)
    E' possibile specificare un puntatore ad un nodo del documento XML come prima, ma inoltre **questo puntatore può essere spostato** attraverso la struttura del documento

Attributo *datapath*

- Tramite l'attributo *datapath* **si "lega" un oggetto ad un particolare nodo** del documento XML caricato

  es: `<dataset name="myData"><myXML>ciao<myXML></dataset>`
  `<text datapath="myData:/myXML/text()" />`

- Il percorso al nodo desiderato viene specificato tramite la **sintassi XPath**

- **Può essere relativo** all'attributo datapath del tag padre (nel caso di tag annidati)

Sintassi XPath

- E' lo **standard del W3C** ideato per accedere a parti di documenti XML (che hanno una *struttura ad albero*)
- XPath è un **insieme di regole/istruzioni** per identificare i nodi dell'albero XML
- La notazione utilizzata è analoga a quella usata per accedere al *filesystem* in LINUX

Sintassi base: (sintassi completa nel manuale)

- per identificare il nodo (= testo) che ci interessa, **si deve specificare il path** (relativo o assoluto) che lo identifica, separando i vari "pezzi" (= tag XML) con il simbolo "**/**"
- nel percorso deve essere anche specificato il **dataset di origine** (analogo a specificare il *drive* nel filesystem) mettendo il nome del dataset prima del path
  es: datapath = "myDataset:/myXML/persone...ecc..ecc..."
- si possono **filtrare i nodi** da estrarre per mezzo di espressioni racchiuse tra parentesi quadre "**[.....]**"
- esistono **funzioni per estrarre il testo** racchiuso nei tag (es. **name()** e **text()**) e negli attributi (simbolo "**@**")

- **Replication manager**:
  - è una delle caratteristiche più importanti e potenti messe a disposizione da OpenLaszlo
  - quando tramite l'attributo datapath di un oggetto si seleziona **più di un nodo**, allora l'oggetto stesso viene replicato (clonato)
  - una copia dell'oggetto viene quindi creata per ciascun tag XML selezionato dal datapath
- ⭐ Metodo molto pratico, ma quando si devono replicare molti oggetti può diventare inefficiente!

⭐ **Pooling**:

- normalmente, quando il *datapath* cambia, gli oggetti clonati vengono distrutti e poi ricreati ogni volta
- per evitare questo, basta usare l'attributo *pooling*
- con *pooling="true"* gli oggetti creati con la replicazione, non vengono distrutti, ma ri-utilizzati internamente
- è un **metodo di ottimizzazione** molto efficiente
- gli **aggiornamenti dell'interfaccia** risulteranno **più veloci e fluidi**, in quanto OpenLaszlo deve solo rimappare i nuovi dati in oggetti già esistenti
- il costo maggiore, infatti, è *istanziare i nuovi oggetti*

⭐ **Lazy replication**:

- tuttavia se gli oggetti da replicare sono molti, il costo per istanziarli tutti potrebbe essere comunque notevole nonostante il pooling
- **altra strategia di ottimizzazione**: in questo caso vengono create solo le viste che sono effettivamente visualizzate (da qui il nome "*lazy*", cioè *pigro*)
- Esempio: *se si ha una griglia con 1000 righe, in cui però se ne visualizzano solo 10 per volta, si creano solo le 10 righe necessarie e di volta in volta si aggiornano i dati di queste righe*
- molto efficiente, ma è vincolata a certe restrizioni per il suo utilizzo (viene usata internamente in griglie e liste)

Il tag <*datapointer*>

- Rappresenta un **puntatore ad un nodo XML** dentro al documento caricato
- **Può essere spostato** nel documento e permette la modifica dell'XML contenuto:
  - setXPath(), xpathQuery()
  - get/setNodeName(), get/setNodeText(), get/setNodeAttribute()
  - selectParent(), selectChild(), selectNext(), selectPrev()
- Possiede gli **eventi** *ondata*, *onerror* e *ontimeout*

- Vengono utilizzati **quando serve manipolare** i dati in qualche modo:

```
<dataset name="myData" src="resources/myShowData.xml" />

<datapointer datapath="myData:/" ondata="processData()">
<method name="processData">
this.selectChild(2);
do {
    if (this.xpathQuery( '@show' ) == 'south park')
        Debug.debug("%w", this.xpathQuery('lastName/text()'));
} while ( this.selectNext() );
</method>
</datapointer>
```

Dati dinamici

- I dati passati all'interfaccia OpenLaszlo così visti fino ad ora sono però statici
- Per renderli dinamici, **i documenti XML verranno creati dinamicamente** tramite un linguaggio di scripting server-side
- Il linguaggio scelto è **PHP** (semplice e sufficientemente potente), che gira nativamente su un secondo server (Apache).

Come interagiscono i due server:

- La GUI OpenLaszlo *(sul server Tomcat alla porta 8080)* effettua una richiesta per avere dei dati tramite il tag <dataset>
- L'url specificato punta ad uno script PHP *(sul server Apache alla porta 80)*. I due server possono risiedere sulla stessa macchina.
- Questo script crea dinamicamente il documento XML e lo restituisce al browser tramite protocollo HTTP
- La GUI viene avvisata della ricezione dei dati con i soliti tre eventi *ondata()*, *onerror()* oppure *ontimeout()*