DIVE INTO GENROPY

BAG E STRUCTURES

INTRODUZIONE

- La Bag è una struttura dati gerarchica accessibile tramite un *path*
- E' implementata sia in Python che Javascript
- Può rappresentare in memoria un XML e facilmente trasformarsi in un XML
- Può essere visualizzata come griglia o albero
- Ha qualche asso nella manica: *triggers* e *resolvers*

PERCHÉ PARLARE DELLA BAG

Perché in Genropy viene usata in tantissimi ambiti

- Nell'ORM di Genropy
- Lo store dati della pagina è una Bag
- La programmazione in Genropy è in grande misura costituita dall'utilizzo di *descrittori* basati su Bag
- Trasporto dati nella comunicazione client/server

LA BAG SOMIGLIA A UN DIZIONARIO

- La Bag somiglia a un dizionario gerarchico
  - Usa la sintassi [] in lettura e scrittura
  - Implementa keys, items, values
  - Ma la chiave può essere un path diviso da punti
  - Tutti i livelli intermedi di una Bag sono a loro volta Bag

```
from gnr.core.gnrbag import Bag

b = Bag()
b['ricetta.titolo'] = 'Spaghetti al pomodoro'
b['ricetta.ingredienti.i1'] = 'spaghetti'
```

SETITEM / GETITEM

- Ma posso leggere e scrivere i valori anche con i metodi
  - setItem(path, valore)
  - getItem(path, valore)
- Nella Bag Javascript esiste solo questa sintassi

```
b.setItem('ricetta.ingredienti.i2', 'salsa di pomodoro')
```

VEDIAMO LA BAG

```
>>> print (b)
0 - (Bag) ricetta:
    0 - (str) titolo: Spaghetti al pomodoro
    1 - (Bag) ingredienti:
        0 - (str) i1: spaghetti
        1 - (str) i2: salsa di pomodoro
```

```
>>> b.toXml()
```

La Bag della ricetta vista come XML formattato

```
<?xml version="1.0" ?>
<GenRoBag>
    <ricetta>
        <titolo>Spaghetti al pomodoro</titolo>
        <ingredienti>
            <i1>spaghetti</i1>
            <i2>salsa di pomodoro</i2>
        </ingredienti>
    </ricetta>
</GenRoBag>
```

LA BAG SOMIGLIA A UN XML

- Può essere istanziata da una stringa o da un file XML
- Può essere serializzatA come stringa o file XML

```
mybag = Bag('ricettario.xml')
mybag.toXml('ricettario.xml')
```

Ma un XML può avere anche degli attributi

```
<?xml version="1.0" ?>
<GenRoBag>
    <ricetta>
        <titolo>Spaghetti al pomodoro</titolo>
        <ingredienti>
            <i1 peso="200g" tipo="grossi" >spaghetti</i1>
            <i2 peso="100g" marca="posillipo">salsa di pomodoro</i2>
        </ingredienti>
    </ricetta>
</GenRoBag>
```

ANCHE LA BAG HA GLI ATTRIBUTI

Per scrivere contemporaneamente valori e attributi uso *setItem*

```
#aggiungo un ingrediente con attributo peso
ingredienti = b['ricetta.ingredienti']
ingredienti.setItem('i3', 'parmigiano', peso='30g')
```

Per leggere gli attributi posso usare la notazione [ ]

```
>>> ingredienti['i3?peso']
'30g'
```

RESOLVER

Il contenuto di una Bag può essere lazy e dinamico. Questo concetto è definito resolver.

```
from datetime import datetime
b.setCallBackItem('ora_esatta', datetime.now)
print (b['ora_esatta'])
```

Ogni volta che accedo al path 'ora_esatta' ottengo il valore calcolato al momento

RESOLVER

- Quando si accede al path, il resolver può ritornare un valore ottenuto in qualunque modo:
  - Chiamate al sistema operativo
  - Letture dal database
  - Accesso a webservices di terze parti
- Ma tutto questo risulta trasparente per chi accede alla Bag
- Se si assegna il parametro *cachetime* il valore viene calcolato solo quando scade la la cache del resolver

TRIGGER

Su una Bag è possibile definire un osservatore di eventi **insert**, **delete** e **update** con il metodo *subscribe*

```
def insertLogger(**kwargs):
    print ('aggiunto ingrediente!')

def updateLogger(**kwargs):
    print ('modificato ingrediente!')

b['ricetta.ingredienti'].subscribe('log', insert=insertLogger, update=updateLogger)
b['ricetta.ingredienti.i4']='basilico'
b['ricetta.ingredienti.i1']='spaghettini'
```

RICAPITOLO BAG

- La Bag si scrive e legge con path gerarchici separati da punti rappresenta bene un albero in memoria
- Una Bag collezione di elementi con attributi omogenei può invece rappresentare una matrice
- Può contenere elementi lazy e dinamici: resolver
- Ha un sistema a eventi nativo: trigger
- Si legge da un XML e si serializza in un XML

STRUCTURES

- La classe GnrStructData eredita da Bag e permette di definire attraverso appositi metodi descrittori, realtà complesse composte da collezioni di entità innestate (es: HTML, XML, DB model, GUI)
- Il metodo più importante di GnrStructData è *child* che di fatto è un altro modo di fare una *setItem* su una Bag

STRUCTURES, IL METODO CHILD

```
from gnr.core.gnrstructures import GnrStructData

mystruct = GnrStructData()

mystruct.child(tag='ingrediente',nome='Spaghetti', peso='200g')
mystruct.child(tag='ingrediente',nome='Salsa', peso='100g')
f = mystruct.child(tag='fase', nome='Preparare sugo', t='10 min.')
f.child(tag='fase', nome='Tagliare cipolla')
f.child(tag='fase', nome='Soffriggere cipolla')
f.child(tag='fase', nome='Aggiungere salsa')
```

- La child equivale ad una *setItem* ad un path prefissato basato sull'attributo tag
- Inoltre restituisce l'oggetto appena aggiunto

STRUCTURES, DEFINIRE METODI DESCRITTORI

```
from gnr.core.gnrstructures import GnrStructData

class Ricetta(GnrStructData):

    def ingrediente(self, nome, peso):
        return self.child(tag='ingrediente', nome=nome, peso=peso)

    def fase(self, nome, descrizione=None, tempo=None):
        return self.child(tag='fase', nome=nome, tempo=tempo, descrizione=descrizione)

    def sezione(self, nome):
        return self.child(tag='sezione', nome=nome)
```

Definire questi metodi serve anche a scopo *documentativo* e permette un controllo dei parametri rispetto, rispetto all'uso di una Bag "libera"

USIAMO LA STRUCTURE D'ESEMPIO

```
def main():
    myricetta = Ricetta()
    ingr = myricetta.sezione('Ingredienti')
    ingr.ingrediente('Spaghetti', '100g')
    ingr.ingrediente('Salsa pomodoro', '80g')

    procedimento = myricetta.sezione('Procedimento')
    sughetto = procedimento.fase(nome='Preparare sugo', tempo='15 min')
    sughetto.fase(nome='Tagliare cipolla')
    sughetto.fase(nome='Soffriggere cipolla', tempo='3 min')
    sughetto.fase(nome='Aggiungere salsa')

    #infine esporto in xml per controllare il risultato
    myricetta.toXml('spaghetti.xml')
```

Notate l'annidamento sezione > fase > fase

UTILIZZI REALI DELLE STRUCTURES IN GENROPY

- Descrivere le pagine servite
- Descrivere il model del database
- Descrivere l'HTML delle stampe
- Poi ogni tipo di entità descritta avrà il suo costruttore, che la struttura descritta produrrà: oggetti Python, tabelle SQL o elementi di DOM