Introduzione a Qt Quick

Alessandro M. Rizzi

April 2, 2012

Outline I

Outline II

Cos' è Qt Quick?

Un insieme di tecnologie che comprendono:

- Un linguaggio dichiarativo di markup: QML
- Un IDE (Qt Creator) che supporta QML
- Un editor grafico
- C++ API per l' integrazione con applicazioni Qt

Noi ci concentreremo su QML

Le caratteristiche di Qt Quick

- Interfacce utente intuitive
- Design-oriented
- Prototipizzazione e produzione rapida
- Deployment facile

Cos'è QML?

Un linguaggio dichiarativo per gli elementi che compongono l' interfaccia utente

- Specifica l' interfaccia utente
  - Come gli elementi appaiono
  - Come gli elementi si comportano
- L' interfaccia utente è specificata mediante una struttura ad albero di elementi contenenti proprietà

Elementi di QML

- Sono le strutture del linguaggio di markup
  - rappresentano parti visibili e non visibili
- L' Item è il tipo base degli elementi visibili
  - non è visibile
  - ha posizione e dimensioni
  - può raggruppare elementi grafici
  - usato spesso come elemento radice
  - Rectangle, Text, TextInput, ...
- Gli elementi contengono proprietà
  - è possibile definirne di nuove

Proprietà

Gli elementi sono definiti da proprietà:

- Semplicemente definizioni nome-valore
  - width, height, color, ...
  - hanno valori di default
  - ognuna ha un tipo ben definito
  - separati da punto e virgola o andata a capo
- Usate per
  - identificare gli elementi (id)
  - personalizzarne l' aspetto
  - cambiarne il comportamento

Tipologie di proprietà

- Standard properties
  - è possibile attribuire valori
- Grouped properties
  - raggruppano insieme proprietà correlate
- Identity property
  - attribuisce un identificativo agli elementi
- Attached properties
  - proprietà non degli elementi ma che possono essere attaccate ad essi (es: KeyNavigation.tab)
- Custom properties
  - sono definite dall' utente
  - possono essere aggiunte a qualsiasi elemento

Binding

- Le proprietà possono contenere espressioni
- Il valore dato non è solo un assegnamento iniziale
  - Ogni volta che l' espressione assegnata cambia viene aggiornato il valore della proprietà
- Le espressioni sono valutate all' occorrenza

Identificare un elemento

- La proprietà id identifica un elemento
- Permette ad altri elementi di riferirsi ad esso
  - per posizionamento e dimensioni (vedremo dopo)
  - per utilizzare le sue proprietà
  - per cambiarne le proprietà (es: animazione)
  - per riusare elementi comuni (es: gradienti, immagini)
- Usato per creare relazioni tra elementi

Metodi

- Alcune azioni non possono essere mostrate come proprietà
- Gli elementi hanno metodi per eseguire azioni. Es:
  - TextInput ha il metodo selectAll()
  - Timer ha i metodi start(), stop() and restart()
- Tutti i metodi sono pubblici
- Alcuni metodi sono usati per eseguire conversioni tra tipi. Es:
  - Qt.formatDateTime(datetime, format)
  - Qt.md5(data)
  - Qt.tint(baseColor, tintColor)

Tipi base

Le proprietà sono di tipi differenti:

- Numeri (interi e in virgola mobile): (20 e 2.8)
- Valori booleani (true e false)
- Stringhe ("Hello Qt")
- Costanti: (AlignLeft)
- Liste: [...]
  - liste composte da un unico elemento possono essere scritte come l' elemento stesso
- Scripts
  - inseriti dirattamente nella definizione della proprietà
- Altri tipi:
  - Es: colori, date, punti, dimensioni, vettori 3D, ...
  - in genere creati tramite costruttori

Elementi annidati

- L' annidamento tra elementi è molto sfruttato (è una caratteristica importante di QML)
- Ogni elemento è posizionato in riferimento al padre

Colori

I colori possono essere specificati in vari modi:

- Come una stringa contentente in nomi (utilizzando nomi SVG):
  - "red", "green", "blue", ...
- Con una stringa contenente le componenti:
  - rosso, verde e blu: #<rr><gg><bb>
  - "#ff0000", "#008000", "#0000ff", ...
  - alpha, rosso, verde, e blu: #<aa><rr><gg><bb>
  - “#ff00ff00”,"#66008000", "#900000ff", ...
- Usando una funzione (red, green, blue, alpha):
  - Qt.rgba(0,0.5,0,1)
- Fornendo l' opacità:
  - tramite la proprietà opacity
  - valori da 0.0 (trasparente) a 1.0 (opaco)

Immagini

- Rappresentate dall' elemento Image
- Collegato al file immagine con la proprietà source
  - utilizzando URL assoluti
  - o relativi al file QML
- Può essere trasformata
  - scalata, ruotata
  - attorno a un' asse o un punto centrale

Proprietà dell' immagine

Scalatura:

- Definita dalla proprietà scale
- Di default il centro dell' elemento rimane nella stessa posizione

Rotazione:

- Definita dalla proprietà rotate
- Di default il centro dell' elemento rimane nella stessa posizione
- Con la proprietà transformOrigin è possibile specificare il centro di rotazione

Gradienti

Per definire un gradiente si ricorre alla proprietà gradient:

- Con un elemento Gradiente come valore
- Contenente due o più elementi GradientStop, ognuno con
  - una posizione: un numero tra 0 (punto iniziale) e 1 (punto finale)
  - un colore
- I punti iniziali e finali
  - sono i bordi superiori e inferiori dell' elemento
  - non possono essere riposizionati
- Problemi coi gradienti:
  - l' elaborazione richiede parecchia potenza di calcolo
  - possono non essere animati come previsto
  - meglio usare immagini di gradienti
- I gradienti sostituiscono le definizioni del colore

Text

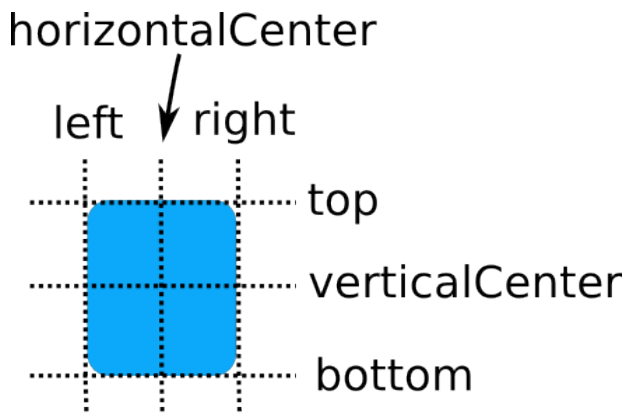
- Visualizza un testo semplice
- Larghezza e altezza stabiliti dalle dimensioni del font e dalla lunghezza del testo
- E' possibile usare rich text
  - si introducono tag HTML nel testo: "<html>Rich Text</html>"

Text input

- Testo semplice editabile
  - senza decorazioni (non è una QLineEdit widget)
- Ottiene il fuoco quando cliccato
  - necessita qualcosa da cliccare
- La proprietà text cambia all' input dell' utente

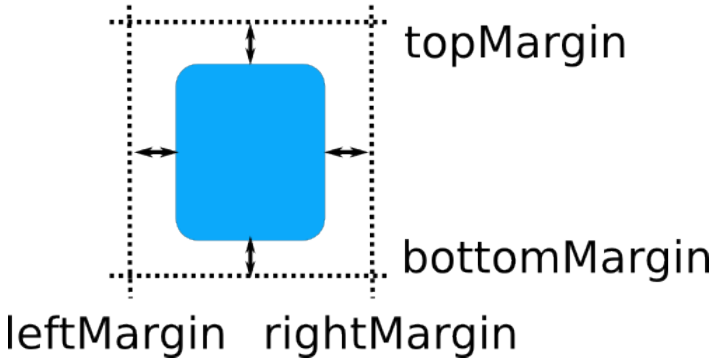
Anchors

- Usati per posizionare e allineare elementi
- Inchiodano i bordi o le linee centrali degli elementi
- Si riferiscono a:
  - altri elementi (centerIn, fill)
  - anchors di altri elementi (left, top)



Margini

- Usati con anchors per aggiungere spaziature
- Le distanze vengono espresse:
  - in pixel
  - tra elementi connessi con anchors



Consigli

- Gli anchors possono essere usati solo riferiti all' elemento padre o a un fratello
- Gli anchors lavorano sui vincoli
  - alcuni elementi devono avere posizione e dimensioni ben definite
  - elementi senza dimensioni di default dovrebbero essere posizionati rispetto a elementi fissi o ben definiti
- Gli anchors creano dipendenze sulle posizioni di altri elementi
  - definiscono un ordine in cui le posizioni vengono calcolate
  - vanno evitate dipendenze cicliche (es. padre $\rightarrow$ figlio $\rightarrow$ padre)
- I margini vanno usati solo se l' anchor corrispondente è definito:
  - es. leftMargin necessita che left sia presente

Strategie d' uso

Identificare elementi con ruoli differenti nell' interfaccia utente:

- Elementi fissi
  - assicurarsi che abbiano l' id definito
  - a meno che possano essere facilmente identificati come elementi padre
- Elementi che dominano l' interfaccia utente
  - assicurarsi che abbiano l' id definito
- Elementi che vengono influenzati dai cambienti delle dimensioni delgi elementi dominanti
  - fissare questi elementi ai dominanti o ai fissi

Mouse area

Una mouse area specifica parti dello schermo dove può avvenire input col cursore del mouse

- Posizionata e dimensionata come un normale elemento
  - con anchor se necessario
- Due modi per controllare l' input da mouse:
  - Usare i segnali
  - Sfruttare i legami delle proprietà

Consigli sulle mouse area

- Una mouse area reagisce solo ai suoi acceptedButtons
  - gli handler non sono chiamati per altri buttoni, ma
  - ogni click riguardante un bottone permesso è riportato
  - la proprietà pressedButtons contiene tutti i bottoni
  - anche quelli non permessi, se premuti assieme a un bottone permesso
- Con hoverEnabled impostato a false
  - containsMouse può essere true se la mouse area è cliccata

Signal vs assegnamento proprietà

Quale usare?

- I segnali sono più facili da usare in alcuni casi
  - quando un segnale influenza solo un altro elemento
- I legami delle proprietà si basano sugli elementi nominati
  - molti elementi possono reagire a un cambiameto reiferendo alla proprietàmany items can react to a change by referring to a property
- Usare sempre l' approccio più intuitivo
- Preferire assegnamenti semplici a script complessi

Input di testo

L' input da tastiera è trattato in due diversi modi:

- Accettando il testo in ingresso
  - TextInput eTextEdit
- Navigando tra gli elementi
  - cambiando l' elemento selezionato
  - frecce direzionali, tab e backtab
- È anche possibile maneggiare raw input da tastiera

Gestione del fuoco

- Nelle interfacce con un solo TextInput
  - il fuoco è assegnato automaticamente
- Con più di un TextInput
  - bisogna cambiare il fuoco con un click del mouse
- Che succede se un TextInput non contiene testo?
  - non è possibile cliccarlo
  - a meno che non abbia una larghezza o usi anchor
  - impostare la proprietà focus per assegnarli il fuoco

Raw input

- L' input raw da tastiera può essere gestito da elementi
  - con handler predefiniti per tasti comunemente usati
  - sono disponibi informazioni complete sull' evento di tastiera
- La gestione del fuoco è la stessa di un text input
  - abilitato settando la proprietà focus
- La gestione dei tasti non è una proprietà ereditata dagli elementi
  - abilitata utilizzando l' attached property Keys
- Gli eventi di tastiera possono essere inoltrati ad altri oggetti
  - abilitato utilizzando l' attached property Keys.forwardTo
  - accetta una lista di oggetti

Stati

Gli stati operano su elementi nominati

- Rappresentati dall' elemento State
- Ogni oggetto può definire un insieme di stati
  - con la proprietà states
  - lo stato corrente è impostato con la proprietà state
- Le proprietà sono impostate quando si entra in uno stato
- È possibile anche:
  - modificare gli anchor
  - cambiare il padre degli oggetti
  - eseguire script

Cambiare le proprietà

Gli stati cambiano le proprietà mediante l' elemento PropertyChanges

- Agiscono su un elemento bersaglio con la proprietà target
  - target si riferisce a un id
- Applicano le definizioni di proprietà all' elemento bersaglio
  - una PropertyChanges può ridefinire diverse proprietà
- Le definizioni di proprietà sono valutate quando si entra nello stato
- PropertyChanges specifica nuovi valori di proprietà per un oggetto
  - i nuovi valori sono assegnati quando si entra nello stato
  - alle proprietà non specificate vengono assegnati i loro valori di default

State condition

Un modo per usare gli stati:

- Lasciar decidere agli stati quando essere attivi
  - usando condizioni per determinare se uno stato è attivo
- Si specifica la proprietà when
  - usando un' espressione che viene valutata vera o falsa
- Solo uno stato in una lista di stati dovrebbe essere attivo
  - Assicurarsi che when sia vero per un solo stato

Transizioni

- Specificano come gli oggetti cambiano quando lo stato cambia
- Vengono applicate a due o più stati
- Normalmente descrivono come gli oggetti sono animati

Transizioni reversibili

- Utili quando due transizioni operano sulla stessa proprietà
- La trasizione si applica dallo stato A allo stato B
  - e viceversa da B ad A
- Non è necessario definire due transizioni distinte

Usare stati e transizioni

- Evitare la definizione di diagrammi di stati complessi
  - non un solo diagramma di stati per gestire l' intera interfaccia utente
  - in generale definita per ciascun componente
- Impostare lo stato con uno script
  - facile da fare, ma potrebbe essere difficile da gestire
  - non è possibile usare transizioni reversibili
- Impostare lo stato con state condition
  - ha uno stile più dichiarativo
  - può essere difficile specificare le condizioni
- Usare animazioni durante le transizioni
  - non specifiare le proprietà from e to
  - usare elementi PropertyChanges nelle definizioni degli stati

Animazioni

Le animazioni possono essere aggiunte a qualsiasi elemento visibile

- Le animazioni cambiano le proprietà causando una cambiamento visibile
- Tutte le animazioni sono proprietà animate
- Tipi specifici di animazioni:
  - NumberAnimation per modifiche a proprietà numeriche
  - ColorAnimation per modifiche a proprietà colore
  - RotationAnimation per modifiche all' orientazione degli oggetti
  - Vector3dAnimation per movimenti nello spazio tridimensionale
- Le easing curve creano animazioni a velocità variabile
- Le animazioni sono usate per creare effetti visivi

Property animation

Le property animation modificano proprietà nominate di un target

- Definite separatamente dall' elemento bersaglio
- Applicate alle proprietà del bersaglio
  - properties è una lista di nomi separati da virgola
- Spesso usata come parte di una Transition
- Non eseguita di default
  - si imposta la proprietà running a true

Number animation

Le number animation modificano il valore di proprietà numeriche

- Sono applicate direttamente alle proprietà con la keyword on
- Possono essere definite separatamente dalla proprietà
- Sono un tipo particolare di Property animation

Color animation

- ColorAnimation specifica i cambiamenti di colore agli oggetti
- Esegue un blending per componente dei valori RGBA

Rotation animation

- RotationAnimation specifica le rotazioni di oggetti
- Più facile da usare che la NumberAnimation per lo stesso scopo
- Applicata alla proprietà rotation di un elemento
- Il valore della proprietà direction controlla la rotazione:
  - RotationAnimation.Clockwise
  - RotationAnimation.Counterclockwise
  - RotationAnimation.Shortest – la direzione avente angolo minore tra i valori from e to

L' elemento behavior

Behavior imposta un' animazione quando una determinata proprietà cambia

Easing curve

Applica una easing curve a un' animazione

- Imposta la proprietà easing.type
- Modifica il tempo di esecuzione dell' animazione
  - a un valore interpolato tra i valori from e to
  - usando una funzione determinata da easing.type

Animazioni sequanziali e parallele

Le animazioni possono essere esseguite in sequenza o in parallelo

- SequentialAnimation definisce una sequenza
  - con ogni animazione figlio eseguita in sequenza
- ParallelAnimation definisce un gruppo parallelo
  - con tutte le animazioni figlio eseguite contemporaneamente

Animazioni sequenziali

- Le SequentialAnimation non hanno un target
  - rappruppano soltanto altre animazioni (con un proprio target)

Pausa tra animazioni

- PauseAnimation è utilizzata per inserire una pausa tra le animazioni
- Non ha una proprietà target
- Ha solo una proprietà duration

Animazioni parallele

- Le ParallelAnimation non hanno un target
  - rappruppano soltanto altre animazioni (con un proprio target)

Custom item e componenti

Due modi per creare componenti riutilizzabile:

- Custom items
  - definiti in un file separato
  - un elemento principale per file
  - usati allo stesso modo degli elementi standard
  - possono aver associato un numero di versione
- Componenti
  - usati con model e view
  - definiti con l' elemento Component item
  - usati come template per gli elementi

Aggiunta di proprietà

- Sintassi: property <type> <name>[: <value>]

Alias di proprietà

- Sintassi: property alias <name>: <referred_property>
- Cambiamenti all' alias risultano cambiamenti alla proprietà
  - se cambia una cambia anche l' altra

Aggiunta di segnali

- Gli elementi standard definiscono segnali e handler
- Elementi custom possono definire i loro segnali
- Sintassi del segnale: signal <name>[(<type> <value>, ...)]
- Sintassi dell' handler: on<Name>: <expression>

Disporre elementi

Positioner e repeater facilitano l' uso di molti oggetti

- Positioner dispongono oggetti in layout standard
  - in una colonna: Column
  - in una riga: Row
  - in una griglia: Grid
  - come le parole su una pagina: Flow
- I repeaters creano oggetti da un template
  - per usarli coi positioner
  - usando dati da un model
- Combinando questi è facile gestire molti oggetti

Trucchi e consigli sulla disposizione

- Anchors nella Row, Column or Grid
  - si applicano a tutti gli elementi che contengono

Model e view

Model e view forniscono un modo per maneggiare insiemi di dati

- I model mantengono dati o oggetti
- Le view visualizzano dati o oggetti
  - utilizzando delegate

Model

Model puri forniscono accesso ai dati:

- ListModel
- XmlListModel

Visual model forniscono informazioni su come mostrare i dati:

- Visual item model: VisualItemModel
  - contiene oggetti figli che sono forniti alle view
- Visual data model: VisualDataModel
  - contiene un' interfaccia ad un modello sottostante
  - fornisce un delegate per il rendering

ListModel

- I list model contengono semplici sequenze di elementi
- Ogni ListElement contiene
  - uno o più pezzi di dati
  - definiti usando proprietà
  - nessuna informazione su come mostrarsi
- I ListElement non hanno proprietà predefinite
  - tutte le proprietà sono custom properties

```
ListModel {
ListElement { ... }
ListElement { ... }
...
}
```

Definire un ListModel

- Definire un ListModel
  - con un id affinchè sia identificabile
- Definire degli oggetti figlio ListElement
  - ognuno con una proprietà
  - la proprietà sarà utilizzata da un delegate

Definire un delegate

- Definire un Component da usare come delegate
  - con un id affinchè sia identificabile
  - specifica come i dati verranno mostrati
- Le proprietà dei list element possono essere richiamate
- Nell' elemento dentro un Component
- la proprietà parent si riferisce alla view

Lavorare con oggetti

- Un ListModel è una lista dinamica di oggetti
- Oggetti possono essere aggiunti, inseriti, rimossi e spostati

View

- La ListView mostra una classica lista di oggetti
  - dove è possibile disporre gli oggetti in orizzontale o in verticale
- La GridView mostra gli oggetti in una griglia
  - come le icone in un file manager

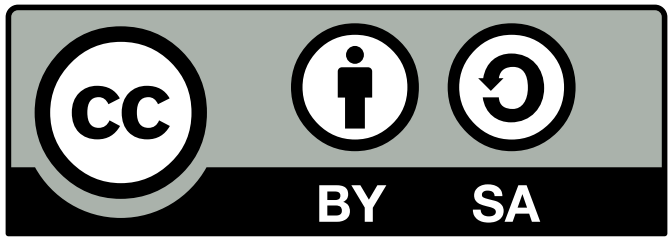
List view

- Come le view in Qt
  - si deve impostare un model
- A differenza delle view in Qt
  - non c' è un delegate di default
  - bisogna creare e utilizzare il delegate altrimenti non verrà visualizzato nulla
- View non clipped disegnano al di fuori della loro area
  - bisogna impostare la proprietà clip per abilitare il clipping
- Le view sono posizionate come gli altri oggetti
  - la view sovrastante riempie il padre

Grid view

- Si usa allo stesso modo della ListView
- Usa i dati da un list model
  - a differenza della table view di Qt
  - in modo simile alla list view di Qt in modalità icone

Licenza



Based on a work at qt.nokia.com.

Basato su

Nokia

Qt Essentials - Qt Quick for C++ Developers

http://qt.nokia.com/learning/online/training/materials/qt-essentials-qt-quick-edition.