Indice generale

Documentazione Pand

Installazione e configurazione

Pand è correntemente pubblicata presso la repository JCenter, di conseguenza per installarla è sufficiente aggiungere la seguente riga di codice tra le dipendenze del file di gradle dell'app che si vuole sviluppare:

Usando una versione di gradle < 3.0
`compile 'it.unibas.flaviosantoro:pand:1.0.0'`

Versione di gradle >= 3.0
`implementation 'it.unibas.flaviosantoro:pand:1.0.0'`

Il framework a questo punto è pronto per essere configurato e usato successivamente. Per la configurazione è necessario invocare il metodo Pand.init, disponibile in due implementazioni. La prima prevede che gli venga passato come argomento unicamente l'istanza della classe android.app.Application, e utilizzerà una configurazione di default; la seconda richiede invece una istanza di Pand.Config, ottenibile tramite il metodo builder della classe; in questo caso è possibile specificare quali debbano essere gli Join Point e se utilizzare o meno il modello persistente automatico.

```
9   public class Application extends android.app.Application {
10      public static final String TAG = Application.class.getSimpleName();
11      private static Application singleton;
12      private Activity currentActivity = null;
13
14      public void onCreate() {
15          super.onCreate();
16          singleton = (Application) getApplicationContext();
17          singleton.registerActivityLifecycleCallbacks(new ActivityLifeCycle());
18
19          Pand.init(this);
20      }
```

Esempio di inizializzazione di Pand utilizzando la configurazione di default

```
public void onCreate() {
    super.onCreate();
    singleton = (Application) getApplicationContext();
    singleton.registerActivityLifecycleCallbacks(new ActivityLifeCycle());

    Pand.Config pandConfig = Pand.Config.build(this)
            .usePersistentModel(true)
            .addJoinPoint("add")
            .addJoinPoint("set")
            .addJoinPoint("update");
    Pand.init(pandConfig);
}
```

Esempio di inizializzazione utilizzando una configurazione personalizzata

Model

Il Framework mette a disposizione un contenitore per tutti i bean del modello che rappresentano lo stato lo stato dell'applicazione, ed è accessibile dal singleton di Pand: `IModel model = Pand.getInstance().getModel();`

L'interfaccia IModel offre i seguenti metodi:

```
Object putBean(String id, Object bean);
```
Inserisce un bean nel modello con il nome specificato, restituendo il riferimento al bean di tipo Proxy che è stato effettivamente inserito.

```
Object putBeanWithoutNotify(String id, Object bean);
```
Inserisce un bean nel modello con il nome specificato senza notificare gli osservatori, restituendo il riferimento al bean di tipo Proxy che è stato effettivamente inserito.

```
Object getBean(String id);
```
Restituisce il bean nel modello con il nome specificato.

```
Object popBean(String id);
```
Restituisce e rimuove il bean dal modello con il nome specificato.

```
void removeBean(String id);
```
Rimuove il bean dal modello con il nome specificato.

```
void addObserver(String beanDotName, Observer observer);
```
Aggiunge un observer per la proprietà del bean, specificata secondo la dot notation.

```
void removeObserver(String beanDotName, Observer observer);
```
Rimuove un observer per la proprietà del bean, specificata secondo la dot notation.

```
Map<String, List<Observer>> getObserverMap();
```
Restituisce una mappa di tutti gli observer per la proprietà del bean, specificata secondo la dot notation.

```
Map<Activity, Map<String, List<Observer>>> getObserverMapActivity();
```
Restituisce una mappa contenente, per ogni activity, tutti gli observer.

```
void cleanObserverForActivity(Activity activity);
```
Rimuove tutti gli observer da una activity.

```
void refreshActivity(final Activity activity);
```
Aggiorna tutti gli observer da una activity.

```
void notifyAdd(final String beanName);
```
Notifica gli observer che è stato aggiunto un bean.

```
void notifyChange(final String beanDotName);
```
Notifica gli observer che è stata modificata la proprietà del bean specificata tramite la dot notation.

È fondamentale notare che quando viene inserito un bean tramite i metodi putBean e putBeanWithoutNotify, viene restituito il nuovo Proxy Bean che incapsula quello che si voleva inserire. Di conseguenza eventuali modifiche effettuate al bean originale e non sull'oggetto Proxy non si rifletteranno sul resto dell'applicazione.

Quando verrà chiamato un metodo di un bean, inserito nel modello, che combacerà con uno dei Join Point impostati verrà automaticamente richiamato il metodo `notifyChange`. Nel momento in cui fosse necessario far sì che solo uno specifico metodo avvii il meccanismo di notifica da parte dell'aspetto, è possibile aggiungere, nella classe del bean, la seguente notazione sopra al metodo scelto, sostituendo a "{NAME}" il nome della proprietà che sta per essere modificata:

```
@PointCut(property = "{NAME}")
```

Linker

I linker sono realizzati come componenti grafici che utilizzano come base i più utilizzati elementi forniti di default dalla piattaforma Android, in modo da rendere il passaggio al framework il più immediato possibile, e sono utilizzabili nell'ambiente RAD di Android Studio.

```
<EditText
    android:id="@+id/examName"
    android:inputType="text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

Normale EditText

```
<it.unibas.pand.view.PandEditText
    pand:linkToBean="examToAdd.teaching"
    pand:observeBean="examToAdd.teaching"
    android:id="@+id/examName"
    android:inputType="text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

EditText fornita da Pand; si deve specificare il nome completamente qualificato ed è possibile aggiungere dei prametri

Ciascun linker eredita tutte le caratteristiche dell'elemento padre e ne ampliano le funzionalità implementando l'interfaccia `ILinker`, che è possibile utilizzare aggiungendo determinati parametri al relativo tag XML. Di seguito sono elencati i vari linker insieme ai relativi parametri aggiuntivi:

- `it.unibas.pand.view.PandEditText`
- `it.unibas.pand.view.PandCheckBox`
- `it.unibas.pand.view.PandDatePicker`
- `it.unibas.pand.view.PandRadioGroup`
- `it.unibas.pand.view.PandSwitch`

Ad ognuno di loro possono essere specificati i seguenti parametri aggiuntivi:

- `pand:linkToBean="bean.dot.notation"`: indica in quale proprietà di un bean, specificata con la dot notation, il valore verrà inserito nel momento in cui sarà sottomessa la form;
- `pand:observeBean="bean.dot.notation"`: crea un observer che aggiornerà il valore del componente grafico prendendolo dalla proprietà del bean specificato utilizzando la dot notation.

Il valore che andrà inserito in questi parametri deve rispettare la seguente struttura:

```
<id del bean>[.<nome della proprietà>.<nome della proprietà>...]
```

Ad esempio, se abbiamo il bean `Person`, inserito nel modello con l'id `person`, se vogliamo indicare la sua proprietà `name` al linker/observer, la stringa risultante sarà:

```
person.name
```

Se invece tra le proprietà avessimo una oggetto di tipo `Address`, contenente le informazioni di recapito, indicato con il nome `mainAddress`, potremmo indicare le sue proprietà come l'esempio seguente:

```
person.mainAddress.street
```

Gli altri elementi grafici forniti da Pand sono i seguenti:

- `it.unibas.pand.view.PandForm`
  elemento di layout che estende `FrameLayout`, delineando una form. Tutti gli elementi figlio di tipo `ILinker` verranno convalidati ed eventualmente sottomessi nel momento in cui verrà premuto il bottone di *commit*. Se verrà premuto il bottone di *rollback* questi elementi verranno svuotati. Per indicare ciascuno di questi bottoni, che possono anche non essere figli dell'elemento, si utilizzano i due seguenti parametri XML, a cui deve essere passato l'id del bottone desiderato:
  - `pand:buttonCommit="@id/buttonCommitId"`
  - `pand:buttonRollback="@id/buttonRollbackId"`

  Altro parametro che può essere specificato è:
  `pand:validAction="fully.qualified.class.IValidFormAction"`
  Il parametro da passare è il nome completamente qualificato di una classe che implementerà `IValidFormAction`, che conterrà l'azione da eseguire con i dati della form una volta che saranno stati completamente verificati.

A livello di codice è possibile chiamare per la form il seguente metodo: `public int setAutoCreate(String beanToCreateId, Class beanClass)`
Con questo metodo si specifica alla form di creare un oggetto della classe specificata e di aggiungerlo automaticamente al modello con l'id impostato, una volta che tutti i valori inseriti risultino corretti.

- `it.unibas.pand.view.PandListView`
  una `ListView` che accetta come parametri:
  - `pand:observeBean="dot.notation.to.list"`
  - `pand:adapter="fully.qualified.class.PandBaseAdapter"`

  Il bean da indicare dovrà essere di tipo `List`, mentre come adapter bisognerà passare il nome completamente qualificato di una classe di tipo `PandBaseAdapter`. Questa classe offre una semplificazione rispetto il classico `BaseAdapter`, e verrà discussa in seguito.

- `it.unibas.pand.view.PandRadioButton`
  Deve essere usato all'interno di un `PandRadioGroup`, e accetta il seguente parametro:
  `pand:value="value"`
  Questo parametro conterrà il valore che verrà salvato nel modello se il `PandRadioButton` dovesse essere selezionato al momento della sottomissione della form.

- `it.unibas.pand.view.PandProgressBar`
  Una semplice `ProgressBar` che può accettare il seguente valore:
  `pand:observeBean="dot.notation.to.int"`
  Il valore indicato deve essere un intero, che indicherà l'avanzamento della progressbar.

- `it.unibas.pand.view.PandTextView`
  Una semplice `TextView` che può accettare i seguenti valori:
  `pand:observeBean="dot.notation.to.value"`
  `pand:formatText="format string"`
  Il valore di `formatText` può essere una stringa o un riferimento ad essa, e indicherà come sarà formattato il risultato che verrà mostrato nella `PandTextView`.

A supporto della `PandListView` vi è un adapter semplificato, il `PandBaseAdapter`, pensato per visualizzare nel modo più semplice possibile il layout personalizzato per ogni elemento da mostrare nella vista, rimuovendo al programmatore il compito di reimplementare metodi e righe di codice praticamente sempre identici nella stragrande maggior parte dei casi. Inoltre utilizza il pattern del `ViewHolder`, consigliato dagli

stessi sviluppatori di Android poiché aumenta sensibilmente la velocità di renderizzazione di ogni riga.
Per implementare questo adapter bisogna estendere la classe astratta `it.unibas.pand.adapter.PandBaseAdapter<Item, VH extends PandBaseAdapter.ViewHolder>`, specificando il tipo dell'elemento da visualizzare per ogni riga e del `ViewHolder` personalizzato. I metodi da implementare sono i seguenti:

```
public int getViewId();
```

In questo metodo bisognerà solo restituire l'id del layout XML che che contiene gli elementi della riga.

```
protected CustomViewHolder getViewHolder(View view);
```

Questo metodo dovrà restituire una nuova istanza dell'implementazione personalizzata del ViewHolder.

```
public void onBindViewHolder(VH holder, Item item, int position);
```

Questo metodo verrà richiamato ogni volta che dovrà essere mostrata una nuova riga all'utente; di conseguenza bisognerà impostare per i vari elementi che compongono il ViewHolder un valore preso dall'i-esimo oggetto.

Per creare un ViewHolder bisogna estendere la classe astratta `PandBaseAdapter.ViewHolder`. L'unico metodo da implementare è il costruttore, a cui verrà passata una `View` che non sarà altro che la riga di layout (istanziata o riciclata) da mostrare. A questo punto si individuano i vari elementi grafici che dovranno mostrare le informazioni all'utente, elementi che verranno utilizzati all'interno del metodo `onBindViewHolder`.

```
public class ExamAdapter extends PandBaseAdapter<Exam, ExamAdapter.CustomViewHolder> {
    @Override
    public int getViewId() { return R.layout.exam_row; }

    @Override
    protected CustomViewHolder getViewHolder(View view) { return new CustomViewHolder(view); }

    @SuppressLint("SetTextI18n")
    @Override
    public void onBindViewHolder(CustomViewHolder holder, Exam exam, int position) {
        holder.textViewExamName.setText(exam.getTeaching());
        holder.textViewExamCredit.setText(exam.getCredits() + " - " + exam.getTestoDataRegistrazione());
        String vote = ""+exam.getVote();
        if(exam.isPraise()){
            vote += "+";
        }
        holder.textViewExamVote.setText(vote);
    }

    class CustomViewHolder extends PandBaseAdapter.ViewHolder{
        TextView textViewExamName;
        TextView textViewExamCredit;
        TextView textViewExamVote;

        CustomViewHolder(View row) {
            super(row);
            textViewExamName = row.findViewById(R.id.exam_name);
            textViewExamCredit = row.findViewById(R.id.exam_credits_date);
            textViewExamVote = row.findViewById(R.id.exam_vote);
        }
    }
}
```

Esempio di implementazione di un PandBaseAdapter

Observer

Per ogni componente grafico esiste un relativo observer. Tipicamente non è necessario crearli programmaticamente, poiché vengono automaticamente creati dai linker se viene specificato loro il bean da "osservare" tramite il parametro `observeBean`.

Ad ogni modo, i vari observer forniti da Pand sono compatibili anche con gli elementi grafici base forniti da Android, ed è possibile utilizzarli creandoli programmaticamente.

I vari observer disponibili sono:

- `it.unibas.pand.observer.TextViewObserver`
  Compatibile con TextView, AppCompatTextView, Button, CheckedTextView, EditText, EmojiTextView, RowHeaderView
- `it.unibas.pand.observer.CompoundObserver`
  Compatibile con: CheckBox, RadioButton, Switch, SwitchCompat, ToggleButton
- `it.unibas.pand.observer.DatePickerObserver`
  Compatibile con DatePicker
- `it.unibas.pand.observer.ProgressBarObserver`
  Compatibile con: ProgressBar
- `it.unibas.pand.observer.ListViewObserver`
  Compatibile con: ListView

I primi quattro observer hanno la stessa semantica per il costruttore, ovvero due argomenti di cui il primo è il riferimento all'elemento grafico su cui si rifletteranno i cambiamenti nel modello, e il secondo è il nome della proprietà da osservare seguendo la dot notation sopra descritta. L'ultimo di essi richiede invece anche una stringa contenente il nome completamente qualificato di un `PandBaseAdapter`. Avremo quindi, a livello di codice Java, istruzioni del tipo:

- `new TextViewObserver(textView, "bean.field");`
- `new CompoundObserver(checkBox, "bean.field");`
- `new DatePickerObserver(datePicker, "bean.field");`
- `new ProgressBarObserver(progressBar, "bean.field");`
- `new ListViewObserver(listView, "bean.field", CustomPandBaseAdapter);`

Validator

Per verificare la correttezza dei dati immessi in una form si utilizzano due tipi di validatori: `IValidator` e `IFormValidator`. Il primo si può applicare a tutti gli elementi di tipo `ILinker`, e il suo scopo è verificare che il dato immesso in

quell'elemento rispetti delle regole; mentre il secondo solo ad un elemento di tipo `PandForm` e dovrà controllare l'effettiva coerenza di tutti gli elementi della form.

Pand fornisce anche una serie di `IValidator` pronti, che coprono molti casi d'uso, oltre a mostrare una corretta implementazione di `Ivalidator`. Tra questi abbiamo:

- public StringNotNullValidator([[int minLenght], int maxLenght])
  Verifica che la stringa inserita non sia vuota, o che abbia una lughezza minima e/o una massima.
- public IntegerRangeValidator(Integer minLenght, Integer maxLenght)
  Verifica che il valore sia un intero e che sia compreso tra minLenght e maxLenght. Per non specificare uno o entrambi i valori è possibile passare al loro posto null.
- public EmailValidator(Integer minLenght, Integer maxLenght)
  Verifica che la stringa inserita sia un indirizzo email valido.

```
PandForm pandForm = view.findViewById(R.id.examForm);
pandForm.setAutoCreate(Constants.EXAM_TMP, Exam.class);

final PandEditText editTextName = view.findViewById(R.id.examName);
final PandEditText editTextCredits = view.findViewById(R.id.examCredits);
final PandEditText editTextGrade = view.findViewById(R.id.examGrade);
final PandCheckBox isLode = view.findViewById(R.id.examLode);

editTextName.addValidator(new StringNotNullValidator(1, 50));
editTextCredits.addValidator(new CreditValidator());
editTextGrade.addValidator(new GradeValidator());

pandForm.addValidator(new FormValidator() {
    @Override
    public void validate(PandForm form) {
        if(isLode.getValue() && editTextGrade.getIntegerValue() != 30){
            addErrorMessage(0, R.string.form_error_laude);
        }
    }
});
```

Esempio di PandForm con i relativi validatori

Proguard

Nel momento in cui volessimo offuscare l'applicazione ottenuta sviluppata con l'ausilio di Pand utilizzando Proguard, ovvero il software che ottimizza, minimizza e offusca il codice Java, dovremmo aggiungere le seguenti regole al file “proguard-rules.pro”, sostituendo a “{PACKAGE}” il package delle classi che andranno inserite nel modello.

```
-keep class {PACKAGE}.** { *; }
-keep public class * extends it.unibas.pand.**
```