Animation 2D pour les Android : Comparaison et mise en contraste de différentes techniques pour réaliser la même animation

Par WHITNEY F. - Lana Bauer (traducteur)

Date de publication : 19 septembre 2014

Les commentaires et les suggestions d'amélioration sont les bienvenus, alors, après votre lecture, n'hésitez pas. **Commentez**.

I - Introduction et animations de propriété....3
II - Emplacement emplacement emplacement....3
III - Configuration....4
IV - Animation de propriété....4
IV-A - Animation de valeur....5
IV-B - Animation d'objets....5
IV-C - Animation d'objets avec les images-clés....6
IV-D - Animation d'objets avec le XML....6
IV-E - Animation de propriété de vue....7
V - Résumé....8
VI - Références....8
VIII - Notes :....8

Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/

I - Introduction et animations de propriété

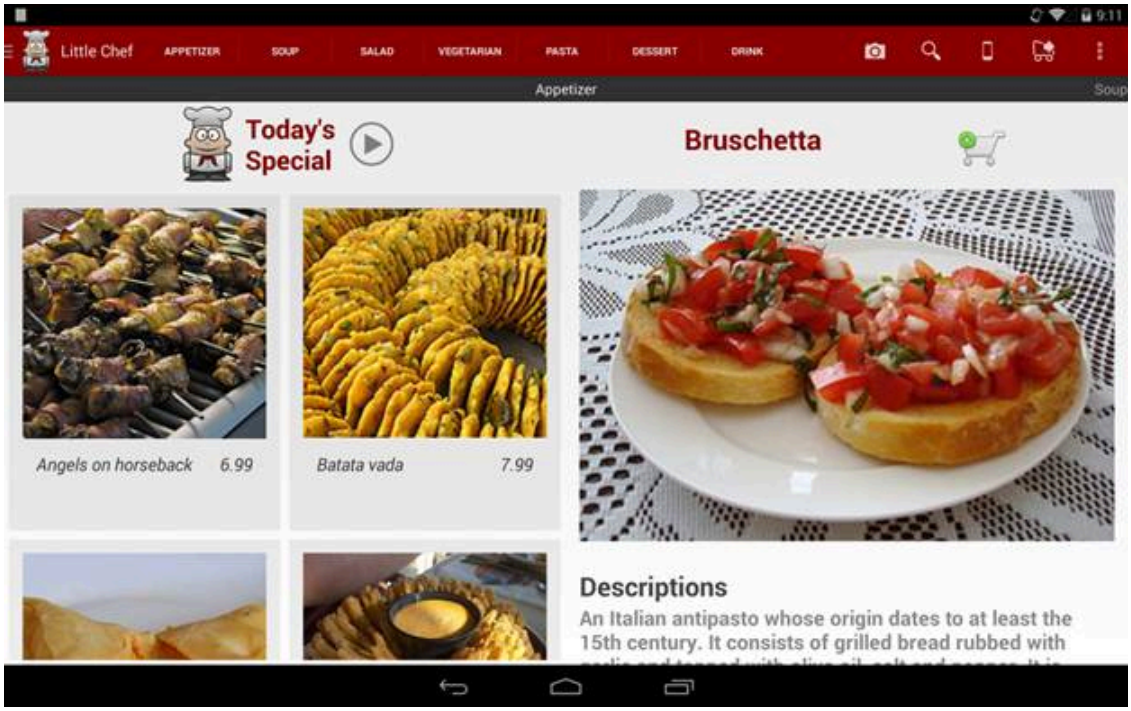
Dans cette série de billets de blog, je vais comparer et mettre en contraste les différents moyens pour réaliser la même animation de translation sur une vue dans Android*. Ceci va vous donner une meilleure idée sur le fonctionnement des diverses méthodes et les différences entre elles, vous permettant ainsi de comparer les avantages et les inconvénients de chacune et de déterminer laquelle est la meilleure pour vos besoins. Il existe plusieurs façons de faire une animation sur Android et cette série pourra seulement en explorer un ensemble, mais devrait être suffisante pour vous aider à démarrer. Le code est conçu pour être adaptable et s'intégrer facilement dans d'autres applications Android. J'utilise également le verrou vrai/faux afin d'illustrer les différents écouteurs d'animation.

Lorsque vous créez une application, vous désirez la rendre visuellement fascinante, donc pourquoi ne pas y ajouter quelques animations ! Dans cette série, j'anime l'icône du petit chef dans une application de restaurant pour attirer l'attention sur le plat du jour. Dans la première partie, nous allons répondre à certaines préoccupations d'emplacement sur l'écran qui viennent avec la translation de l'objet, l'installation de notre application restaurant, puis nous entamerons les animations de propriété. Par la suite, nous allons visualiser les animations dans la deuxième partie, puis des animations légèrement différentes dans la troisième partie, avec des surfaces dessinables et des canevas, suivi par la quatrième partie qui va couvrir les SurfaceView et TextureView, plus complexes.

II - Emplacement emplacement emplacement

Vous ne voulez pas que votre animation sorte de l'écran ou qu'elle s'exécute d'une façon inattendue. Cela devient de plus en plus complexe, si on considère la gamme d'appareils de tailles différentes sur le marché et la nature dynamique des fragments et dispositions Android. Vous pouvez avoir à tenir compte de la gamme de tailles. Pour les premières techniques d'animation, je choisis de récupérer la position de la vue sur l'écran par rapport à son parent, puis de déplacer cette vue de la moitié de cette distance vers la gauche, puis de revenir en arrière avec la même distance à droite pour terminer là où elle a commencé.

D'autres animations discutées dans les parties suivantes de cette série n'auront pas la même capacité dans notre cas et devront ainsi être traitées d'une façon légèrement différente ; ceci sera discuté plus en profondeur dans chacune de leurs sections.



Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/

III - Configuration

Nous sommes en train d'animer notre petite image du chef dans une application pour restaurant. Vous pouvez voir le code pour l'interface utilisateur de cette application ici : **http://software.intel.com/en-us/android/articles/building-dynamic-ui-for-android-devices**. Vous pouvez l'insérer dans n'importe quelle classe qui contrôle l'écran, là où vous voulez que l'animation ait lieu. Ceci est fait dans le *MenuGridFragment.java* dans notre cas. Commencez par définir les variables nécessaires.

```
ImageView mLittleChef;
boolean simpleLock;
int mShortAnimationDuration
```

Ajoutez la balise ImageView au fichier de disposition *fragment_menu_grid.xml.*

```
<ImageView
            android:id="@+id/imageView1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:src="@drawable/dish_special" />
```

Dans onCreateView, initialisons les valeurs de nos variables. Notez que notre page de menu est un fragment à l'intérieur d'un visionneur de pages, vous pourriez avoir besoin de le faire dans onCreate ou ailleurs en fonction de votre code.

```
mLittleChef= (ImageView)rootView.findViewById(R.id.imageView1);
simpleLock=false;
mShortAnimationDuration=4000;
```

Dans notre page de menu, chaque animation est démarrée en appelant sa méthode respective. Ceci est déclenché en cliquant sur un élément dans le menu, seulement pendant 4 % du temps, afin de ne pas submerger l'utilisateur avec les animations. Vous pouvez utiliser une minuterie à la place, mais l'introduction de la fonction de clic améliorera les chances que l'utilisateur utilise activement l'application lorsque l'animation se déclenche. Nous vérifions aussi la variable simpleLock puisque nous voulons que l'animation s'achève entièrement avant d'avoir à recommencer. Cela permettra également de régler la position quand l'animation commence à partir de la position actuelle, au lieu de commencer de la position initiale, lorsqu'elle redémarre pour certaines méthodes.

```
/* Traiter l'évènement de l'article, clic dans la grille de menu */
public void onGridItemClick(MenuItem itemSelected, int position) {
                /* Code de clic dans d'autres grilles */
         // Ajouter l'animation
         mDetailViewFragment.update(itemSelected);
         int animChance= (int) (Math.random() * 5);
         int animChance2= (int) (Math.random() * 5);
         if(!mGridViewFragment.simpleLock & (animChance== animChance2)){
                 //Choisit la méthode d'animation ici
                 mGridViewFragment.doValueAnimator();
                 mGridViewFragment.simpleLock=true;
        }
}
```

IV - Animation de propriété

Vous pouvez animer n'importe quel objet ou valeur que vous voulez, en utilisant l'animation de propriété. Nous allons animer une vue, afin de mieux comparer les autres façons d'animer une vue. Lorsque vous utilisez une animation de propriété, les propriétés réelles vue/objet vont changer, donc il s'agit réellement d'un déplacement dans notre cas et non seulement d'une mise à jour d'où l'objet a été dessiné.

Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/

Notez également que nous faisons l'animation par la mise à jour de la valeur x de la vue. Vous pouvez choisir de mettre à jour la valeur gauche/droite de la vue, mais ce sera simplement un étirement de la vue dans le cas des animations de propriété.

IV-A - Animation de valeur

Dans l'animation de valeur, vous définissez votre ValueAnimator qui crée un ensemble de valeurs parmi celles proposées. Cet ensemble d'animations est complètement séparé de la vue réelle que vous souhaitez animer. Pensez au ValueAnimator plus comme un outil pour vous donner une liste de valeurs à appliquer sur l'objet réel. Pour obtenir la valeur animée courante, vous devriez ajouter un AnimatorUpdateListener à votre animation et appeler getAnimatedValue. Une fois la vue réglée avec la nouvelle valeur, son appel du *setter* (ou mutateur)invalide automatiquement la vue de sorte qu'elle se redessine à l'écran avec la valeur mise à jour. Si vous utilisez un autre appel, vous aurez à appeler manuellement view.invalidate(), ou en cas de changement de sa disposition, vous appellerez requestLayout() sur son parent. Notez que l'interpolation par défaut est AccelerateDecelerateInterpolator, nous allons donc la mettre à jour pour utiliser le LinearInterpolator à sa place.

```
public void doValueAnimator(){
        ValueAnimator animation = ValueAnimator.ofFloat(mLittleChef.getX(), mLittleChef.getX()/2,
 mLittleChef.getX());
        animation.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
                @Override
                public void onAnimationUpdate(ValueAnimator animation) {
                        Float value = (Float) animation.getAnimatedValue();
                        mLittleChef.setX(value);

                        // setX() fait appel pour nous à:
                        mLittleChef.invalidate();
                }
        });
        animation.addListener(new AnimatorListenerAdapter(){
                @Override
                public void onAnimationEnd(Animator animation) {
                        simpleLock= false;
                }
        });
        // L'interpolateur par défaut est AccelerateDecelerateInterpolator, donc nous allons le
 modifier
        animation.setInterpolator(new LinearInterpolator());
        animation.setDuration(mShortAnimationDuration);
        animation.start();
}
```

IV-B - Animation d'objets

Pour définir votre ObjectAnimator, vous devez lui donner un objet et une propriété d'objet, en plus des valeurs que vous souhaitez animer. La propriété devrait avoir une méthode *setter* pour l'ObjectAnimator à appeler. Rappelez-vous que les appels automatiques des *setter* de la vue annulent le rafraîchissement de la vue, de sorte que vous n'avez pas à l'appeler vous-même. Comme ObjectAnimator hérite de la classe ValueAnimator, vous devrez aussi, manuellement, régler l'interpolation pour qu'elle soit linéaire.

```
public void doObjectAnimator(){
        ObjectAnimator animation= ObjectAnimator.ofFloat(mLittleChef, "x",
 mLittleChef.getX(),mLittleChef.getX()/2,mLittleChef.getX());
        animation.addListener(new AnimatorListenerAdapter(){
                @Override
                public void onAnimationEnd(Animator animation) {
                        simpleLock= false;
                }
        });
        animation.setInterpolator(new LinearInterpolator());
        animation.setDuration(mShortAnimationDuration);
        animation.start();
}
```


Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/

IV-C - Animation d'objets avec les images-clés

Vous pouvez également faire de l'animation d'objet avec des images-clés au lieu d'utiliser un PropertyValuesHolder. Les images-clés sont comme des instantanés en temps de la façon dans laquelle votre objet apparaîtra. Quand vous les passez à PropertyValuesHolder, il va automatiquement interpoler les valeurs entre les images-clés. L'avantage d'utiliser les images-clés sur les autres moyens d'animation, c'est que vous pouvez définir le temps entre eux beaucoup plus facilement et d'une manière beaucoup plus condensée. Au lieu de définir plusieurs animations à des moments différents, vous pouvez créer un ObjectAnimator et fragmenter le temps entre les différentes images-clés.

```
public void doObjectAnimatorKeyframes(){
        float startingPoint= mLittleChef.getX();
        Keyframe kf0 = Keyframe.ofFloat(0f, startingPoint);
        Keyframe kf1 = Keyframe.ofFloat(.5f, startingPoint/2);
        Keyframe kf2 = Keyframe.ofFloat(1f, startingPoint);
        PropertyValuesHolder trans = PropertyValuesHolder.ofKeyframe("x", kf0, kf1, kf2);
        ObjectAnimator animation1 = ObjectAnimator.ofPropertyValuesHolder(mLittleChef, trans);
        animation1.addListener(new AnimatorListenerAdapter(){
                @Override
                public void onAnimationEnd(Animator animation) {
                        simpleLock= false;
                }
        });
        animation1.setInterpolator(new LinearInterpolator());
        animation1.setDuration(mShortAnimationDuration);
        animation1.start();
}
```

IV-D - Animation d'objets avec le XML

La troisième façon de faire l'animation d'un objet est d'utiliser un fichier XML pour contenir toutes les valeurs. Le fichier XML devrait être placé dans *res/animator ;* vous pouvez le placer dans *res/anim,* mais le précédent est plus optimal et recommandé. Bien que cela le sépare du code afin de le rendre plus facilement modifiable et partageable, vous êtes aussi limité par cette séparation, car les valeurs sont statiques et codées en dur en pixels. Cela pourrait rendre difficile une animation de translation puisque la taille de l'écran de l'appareil n'est pas toujours la même et vous devriez modifier la “valueTo” vers laquelle on anime.

Si votre animation va rester en place et conserver relativement la même taille, alors vous n'aurez pas ce genre de préoccupations. Pour tenir compte des appareils de différentes tailles, vous pouvez créer des fichiers XML séparés pour les différentes tailles d'appareils et les placer dans le dossier approprié qui se termine par -<taille>, par exemple, *res/animator-large*.

Assurez-vous également d'utiliser des valeurs qui se terminent par dp pour mettre à l'échelle la densité en pixels de l'appareil (pour en savoir plus sur la manipulation des différentes tailles d'écran, voir les liens dans la section ressources à la fin). Cela n'aide pas tout à fait dans notre cas, comme nous ne connaissons pas la valeur d'animation de retour. Nous pourrions créer notre propre propriété avec les *setter* et *getter* qui vont le déplacer à un certain pourcentage de l'écran, mais nous avons déjà un certain nombre d'options d'animation plus simples parmi lesquelles on peut choisir.

```
public void doObjectAnimatorXML(){
        AnimatorSet object = (AnimatorSet)
 AnimatorInflater.loadAnimator(getActivity(),R.animator.property_animator);
        object.addListener(new AnimatorListenerAdapter(){
                @Override
                public void onAnimationEnd(Animator animation) {
                        simpleLock= false;
                }
        });
        object.setTarget(mLittleChef);
        object.start();
}
```


Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/

Ajouter *property_animator.xml* à partir de *res/animator-xlarge* :

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android" android:ordering="sequentially" >
        <objectAnimator
            android:interpolator="@android:anim/linear_interpolator"
            android:propertyName="x"
            android:duration="1000"
            android:valueTo="138"
            android:valueType="floatType"/>
        <objectAnimator
            android:interpolator="@android:anim/linear_interpolator"
            android:propertyName="x"
            android:duration="1000"
            android:valueTo="276"
            android:valueType="floatType"/>
</set>
```

IV-E - Animation de propriété de vue

Ici vous pouvez animer directement avec la vue en utilisant la méthode view.animate(). Vous avez la possibilité d'enchaîner ensemble différentes animations simultanées, rendant le code très compact, par contre vous ne pouvez pas enchaîner ensemble les animations de même type. Ainsi votre animation doit être imbriquée si vous voulez que l'une ait lieu après l'autre.

Une autre chose à noter est que, lors de la lecture de la documentation pour la méthode TranslationX, on dit « va provoquer l'animation de la propriété TranslationX de vue vers la valeur spécifiée ». Ce n'est que la moitié de l'histoire, et vous devriez regarder la propriété TranslationX pour plus de précision : « La position horizontale de cette vue relative à sa position de gauche, en pixels. ». Ainsi la méthode TranslationX ne va pas s'animer à la position absolue, mais à la valeur spécifiée par rapport à la position de gauche de la vue qui est considérée comme '0,0'. Par exemple, si votre vue est dans la position d'origine et que vous souhaitez la déplacer vers 'origine +2', vous devrez mettre '2' comme valeur et puis, pour le retour, utiliser la valeur '0 '.

```
public void doViewPropertyAnimator(){

        // On peut enchaîner (pas de translation X) les animations simultanées ici
        mLittleChef.animate()
                    .translationX(-mLittleChef.getX()/2)
                    .setDuration(mShortAnimationDuration/2)
                    .setInterpolator(new LinearInterpolator())
                    .setListener(new AnimatorListenerAdapter() {
                            @Override
                            public void onAnimationEnd(Animator animation) {

                                    // Reprendre l'animation au point de départ
                                    mLittleChef.animate()
                                                .translationX(0)
                                                .setDuration(mShortAnimationDuration/2)
                                                .setListener(new AnimatorListenerAdapter() {
                                                        @Override
                                                        public void onAnimationEnd(Animator
animation) {
                                                                simpleLock = false;
                                                        }
                                                });
                            }
                    });
}
```


Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/

V - Résumé

Une fois la configuration de base de notre application exécutée, nous ajoutons des animations à celle-ci et ce de différentes façons pour faire l'animation de propriété. L'animation de propriété est la meilleure pour animer les valeurs de propriétés de votre objet que vous pouvez ensuite appliquer à votre objet lui-même.

Retrouvez également de nombreux outils de développement Android sur la **plate-forme Intel Developer Zone**.

VI - Références

Intel Developer Zone Android

Outils de développement Intel

Article original (en anglais)

http://developer.android.com/guide/topics/graphics/index.html

http://developer.android.com/training/basics/supporting-devices/screens.html

http://developer.android.com/guide/practices/screens_support.html

VIII - Notes :

*Les autres noms et marques peuvent être revendiqués comme propriété d'autrui.

**Cet échantillon de code source est sorti sous le contrat de licence de Sample Source Code d'Intel.

Copyright ® 2014 Intel. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc. sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à trois ans de prison et jusqu'à 300 000 € de dommages et intérêts.
https://intel.developpez.com/tutoriels/android/animation-2d-comparaison-mise-contraste-differente-technique-realiser-animation/