Программное создание ConstraintLayout и позиционионирование

Последнее обновление:

Для создания контейнера в коде Java применяется одноименный класс **ConstraintLayout**, для создания объекта которого в конструктор передаются значения для ширины и высоты элемента:

```
ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutPara
                (ConstraintLayout.LayoutParams.WRAP_CONTENT , ConstraintLayo
```

Первый параметр устанавливает щирину элемента, а второй - высоту. `ConstraintLayout.LayoutParams.WRAP_CONTENT` указывает, что элемент будет иметь те размеры, которые необходимы для того, чтобы вывести на экран его содержимое. Кроме `ConstraintLayout.LayoutParams.WRAP_CONTENT` можно применять константу `ConstraintLayout.LayoutParams.MATCH_CONSTRAINT`, которая аналогична применению значения "0dp" в атрибутах `layout_width` и `layout_height` и которая растягивает элемент по ширине или высоте контейнера.

Также можно использовать точные размеры, например:

```
ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.LayoutPara
                (ConstraintLayout.LayoutParams.MATCH_CONSTRAINT, 200);
```

Для настройки позиционирования внутри ConstraintLayout применяется класс **ConstraintLayout.LayoutParams**. Он имеет довольно много функционала. Рассмотрим в данном случае только те поля, которые позволяют установить расположение элемента:

- **baselineToBaseline**: выравнивает базовую линию элемента по базовой линии другого элемента, id которого присваивается свойству.
- **bottomToBottom**: выравнивает нижнюю границу элемента по нижней границе другого элемента.
- **bottomToTop**: выравнивает нижнюю границу элемента по верхней границе другого элемента.
- **leftToLeft**: выравнивает левую границу элемента по левой границе другого элемента.
- **leftToRight**: выравнивает левую границу элемента по правой границе другого элемента.
- **rightToLeft**: выравнивает правую границу элемента по левой границе другого элемента.
- **rightToRight**: выравнивает правую границу элемента по правой границе другого элемента.
- **startToEnd**: выравнивает начало элемента по завершению другого элемента.
- **startToStart**: выравнивает начало элемента по началу другого элемента.
- **topToBottom**: выравнивает верхнюю границу элемента по нижней границе другого элемента.
- **topToTop**: выравнивает верхнюю границу элемента по верхней границе другого элемента.
- **endToEnd**: выравнивает заврешение элемента по завершению другого элемента.
- **endToStart**: выравнивает завершение элемента по началу другого элемента.

В качестве значения эти поля принимают id (идентификатор) элемента, относительно которого выполняется позиционирование. Если расположение устанавливается относительно контейнера ConstraintLayout, то применяется константа **ConstraintLayout.LayoutParams.PARENT_ID**

Рассмотрим простейший пример:

```
1 package com.example.viewapp;
```

```
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);

        ConstraintLayout constraintLayout = new ConstraintLayout(this);
        TextView textView = new TextView(this);
        // установка текста текстового поля
        textView.setText("Hello Android");
        // установка размера текста
        textView.setTextSize(30);

        ConstraintLayout.LayoutParams layoutParams = new ConstraintLayout.La
                (ConstraintLayout.LayoutParams.WRAP_CONTENT , ConstraintLayo
        // позиционирование в левом верхнем углу контейнера
        // эквивалент app:layout_constraintLeft_toLeftOf="parent"
        layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
        // эквивалент app:layout_constraintTop_toTopOf="parent"
        layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
        // устанавливаем размеры
        textView.setLayoutParams(layoutParams);
        // добавляем TextView в ConstraintLayout
        constraintLayout.addView(textView);

        setContentView(constraintLayout);
    }
}
```

ширины и высоты указывает, что элемент будет иметь те размеры, которые необходимы для того, чтобы вывести на экран его содержимое.

Далее выравниваем левую границу элемента по левой стороне контейнера:

```
layoutParams.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
```

Эта установка аналогична использованию атрибута `app:layout_constraintLeft_toLeftOf="parent"`.

Затем выравниваем верхнюю границу элемента по верхней стороне контейнера:

```
layoutParams.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
```

Эта установка аналогична использованию атрибута `app:layout_constraintTop_toTopOf="parent"`.

И в конце применяем объект ConstraintLayout.LayoutParams к TextView:

```
constraintLayout.addView(textView);
```

В итоге элемент TextView будет расположен в верхнем левом углу ConstraintLayout:

Рассмотрим другой пример - установку расположения элементов относительно друг друга:

```
package com.example.viewapp;

import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        //setContentView(R.layout.activity_main);
```

```
16
17          ConstraintLayout constraintLayout = new ConstraintLayout(this);
18
19          EditText editText = new EditText(this);
20          editText.setHint("Введите Email");
21          editText.setId(View.generateViewId());
22
23          Button button = new Button(this);
24          button.setText("Отправить");
25          button.setId(View.generateViewId());
26
27          ConstraintLayout.LayoutParams editTextLayout = new ConstraintLayout
28                  (ConstraintLayout.LayoutParams.WRAP_CONTENT , ConstraintLay
29          editTextLayout.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID
30          editTextLayout.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
31          editTextLayout.rightToLeft = button.getId();
32          editText.setLayoutParams(editTextLayout);
33          constraintLayout.addView(editText);
34
35          ConstraintLayout.LayoutParams buttonLayout = new ConstraintLayout.La
36                  (ConstraintLayout.LayoutParams.WRAP_CONTENT , ConstraintLay
37          buttonLayout.leftToRight = editText.getId();
38          buttonLayout.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
39          button.setLayoutParams(buttonLayout);
40          constraintLayout.addView(button);
41
42          setContentView(constraintLayout);
43      }
44  }
```

При расположении одного элемента относительно другого, нам нужно знать id вторрого элемента. Если элемент определен в коде Java, то вначале надо сгенерировать идентификатор:

```
1  editText.setId(View.generateViewId());
2  button.setId(View.generateViewId());
```

Затем можно применять идентификаторы элементов для установки позиционирование позиционионирование. Так, правая граница EditText выравнивается по левой границе кнопки:

```
1  editTextLayout.rightToLeft = button.getId();
```

```
1  buttonLayout.leftToRight = editText.getId();
```


20:30
ViewApp
Введите Email
ОТПРАВИТЬ