Работа с ConstraintLayout через XML-разметку

В марте Google выкатил релизное обновление ConstraintLayout. Презентовали его еще [на прошлом Google I/O](https://www.youtube.com/watch?v=sO9aX87hq9c). С того момента прошел почти год, ConstraintLayout стал лучше, быстрее и оброс новыми возможностями. Например, приятно, что появилась возможность объединения элементов в цепи — это позволяет использовать ConstraintLayout вместо LinearLayout.

Содержание:

[1. Добавляем в проект](https://habrahabr.ru/company/touchinstinct/blog/326814/#dobavlyaem-v-proekt)  
[2. Contsraints. Привязываем элементы друг к другу](https://habrahabr.ru/company/touchinstinct/blog/326814/#constraints-privyazyvaem-elementy-drug-k-drugu)  
[3. Задаем размеры View](https://habrahabr.ru/company/touchinstinct/blog/326814/#zadaem-razmery-view)  
[4. Задаем размеры View на основе соотношения сторон](https://habrahabr.ru/company/touchinstinct/blog/326814/#zadaem-razmery-view-na-osnove-sootnosheniya-storon)  
[5. Выставляем относительное расположение View внутри ConstraintLayout](https://habrahabr.ru/company/touchinstinct/blog/326814/#vystavlyaem-otnositelnoe-raspolozhenie-view-vnutri-constraintlayout)  
[6. Особенности привязки линий](https://habrahabr.ru/company/touchinstinct/blog/326814/#osobennosti-privyazki-liniy)  
[7. Особенности привязки по Baseline](https://habrahabr.ru/company/touchinstinct/blog/326814/#osobennosti-privyazki-po-baseline)  
[8. Создаем цепи (chains)](https://habrahabr.ru/company/touchinstinct/blog/326814/#sozdaem-cepi-chains)  
[8.1. Стиль spread](https://habrahabr.ru/company/touchinstinct/blog/326814/#stil-spread)  
[8.2. Стиль spread\_inside](https://habrahabr.ru/company/touchinstinct/blog/326814/#stil-spread_inside)  
[8.3. Стиль packed](https://habrahabr.ru/company/touchinstinct/blog/326814/#stil-packed)  
[8.4. Стиль weighted](https://habrahabr.ru/company/touchinstinct/blog/326814/#stil-weighted)  
[9. Указываем отступы с учетом скрытых элементов](https://habrahabr.ru/company/touchinstinct/blog/326814/#ukazyvaem-otstupy-s-uchetom-skrytyh-elementov)  
[10. Разбираемся с Guidelines](https://habrahabr.ru/company/touchinstinct/blog/326814/#razbiraemsya-s-guidelines)  
[11. Настраиваем параметры ConstraintLayout из кода](https://habrahabr.ru/company/touchinstinct/blog/326814/#nastraivaem-parametry-constraintlayout-iz-koda)  
[12. Коротко об анимации](https://habrahabr.ru/company/touchinstinct/blog/326814/#korotko-ob-animacii)  
[13. Стоит ли пользоваться ConstraintLayout?](https://habrahabr.ru/company/touchinstinct/blog/326814/#stoit-li-polzovatsya-constraintlayout)

Добавляем в проект

Обновляем версию Android Studio до 2.3;

1. Проверяем, что установлена последняя версия ConstraintLayout — это можно посмотреть в Android Studio → Settings(Preferences) → Appearance & Behavior → System Settings → Android SDK → SDK Tools → Support Repository;
2. Добавляем dependency в build.gradle модуля проекта:
3. dependencies {
4. ...
5. compile 'com.android.support.constraint:constraint-layout:1.0.2'
6. ...

}

1. Теперь можно использовать ConstraintLayout у себя в проекте:

<android.support.constraint.ConstraintLayout

android:id="@+id/my\_first\_constraint\_layout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:id="@+id/some\_constraint\_layout\_element"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Hello world!"/>

</android.support.constraint.ConstraintLayout>

Constraints. Привязываем элементы друг к другу

**Constraints** — это линии, на основе которых располагается view внутри ConstraintLayout. Constraints могут быть привязаны к **сторонам** самого ConstraintLayout или к **сторонам** других view внутри ConstraintLayout. Constraints можно разделить на вертикальные и горизонтальные.

Горизонтальные constraints:

* правой стороны (Right), левой стороны (Left);
* начальной стороны (Start), конечной стороны (End).

Вертикальные constraints:

* верхней стороны (Top), нижней стороны (Bottom);
* базовой линии (Baseline).

Вертикальное и горизонтальное constraints друг с другом не связаны.

Напомню, что **Baseline** — это линия выравнивания контента элемента. Пример — для TextView это линия строки, на которой пишется текст. Если у view выставлен Baseline сonstraint, то базовая линия элемента будет находиться на уровне базовой линии view, к которой привязан сonstraint.

Для начала, проще всего рассматривать сonstraints, как стороны view. То есть можно, например, привязать левую сторону view B к правой стороне view A — тогда view B будет располагаться справа от view A.

Общий формат атрибутов для привязки сonstraint выглядит следующим образом:

app:layout\_constraint{X}\_to{Y}Of="{Z}"

Где:

* **X** — сonstraint привязываемой view;
* **Y** — **сторона** view, к которой привязываются;
* **Z** — id view, к которой привязываются, или parent, если привязать нужно к стороне ContraintLayout.

Пример привязок:

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<!-- Левый constraint view\_1 привязан к левой стороне ConstraintLayout.

Верхний constraint view\_1 — к верхней стороне ConstraintLayout.

Правый и нижний constraint не привязаны -->

<TextView

android:id="@+id/view\_1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_1"

android:textSize="24sp"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintTop\_toTopOf="parent"/>

<!-- Левый constraint у view\_2 привязан к правой стороне view\_1.

Constraint базовой линии view\_2 — к базовой линии view\_1.

Нижний и верхний constraint привязывать нельзя, так как есть привязка по базовой линии.

Правый constraint не привязан. -->

<TextView

android:id="@+id/view\_2"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_2"

android:textSize="14sp"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"

app:layout\_constraintBaseline\_toBaselineOf="@id/view\_1"/>

<!-- Левый constraint view\_3 привязан к правой стороне view\_1.

Верхний constraint view\_3 — к нижней стороне view\_1.

Правый и нижний constraint не привязаны. -->

<TextView

android:id="@+id/view\_3"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_3"

android:textSize="14sp"

app:layout\_constraintTop\_toBottomOf="@id/view\_1"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"/>

<!-- Левый constraint view\_4 привязан к правой стороне view\_2.

Нижний constraint view\_4 — к нижней стороне view\_2.

Правый и верхний constraint не привязаны. -->

<TextView

android:id="@+id/view\_4"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

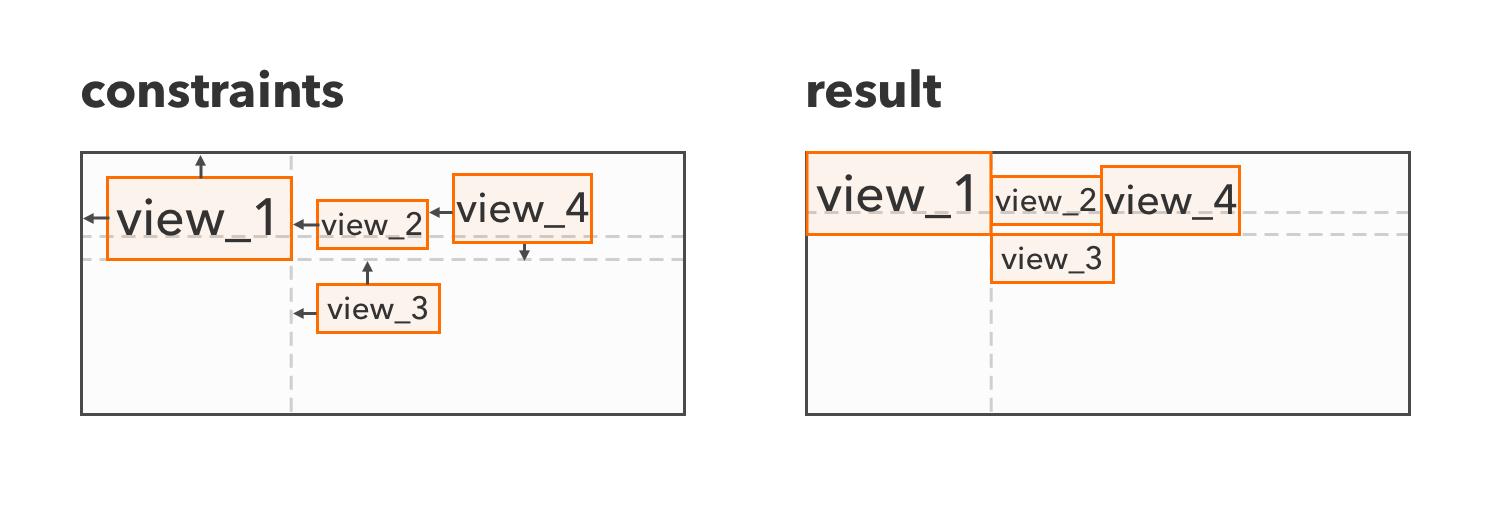
android:text="view\_4"

android:textSize="18sp"

app:layout\_constraintBottom\_toBottomOf="@id/view\_1"

app:layout\_constraintLeft\_toRightOf="@id/view\_2"/>

</android.support.constraint.ConstraintLayout>



Основные правила привязки сторон:

* привязывать между собой можно только Start и End, Left и Right, Top и Bottom. То есть, нельзя, например, привязать Left к Start или Baseline к Top;
* Baseline можно привязать только к Baseline другой view;
* при привязке Start/End игнорируются привязки Left/Right;
* при привязке Baseline игнорируются привязки Top/Bottom;
* не привязывайте view с внешней стороны ConstraintLayout, например, layout\_constraintRight\_toLeftOf="parent". ConstraintLayout обработает такую привязку, но как себя при этом поведет, сложно предсказать.

Задаем размеры View

Чтобы задать размеры view, используются обязательные атрибуты layout\_width и layout\_height, и необязательные атрибуты layout\_constraintWidth\_default и layout\_constraintHeight\_default.

Значение атрибутов layout\_constraintWidth\_default и layout\_constraintHeight\_default по умолчанию равно spread.

Размер view могут быть указан, как (на примере высоты):

* layout\_height="100dp" — обозначим, как fixed\_size. View будет указанного размера;
* layout\_height="wrap\_content" — обозначим, как any\_size. Размер вычисляется самой view, может быть любым;
* layout\_height="0dp" + layout\_constraintHeight\_default="spread" — обозначим, как match\_constraint\_spread. Размер view будет равен расстоянию между constraints. Для высоты, например, это расстояние между верхним constraint и нижним constraint;
* layout\_height="0dp" + layout\_constraintHeight\_default="wrap" — обозначим, как match\_constraint\_wrap. Размер вычисляется самой view, но не может выйти за рамки constraints.

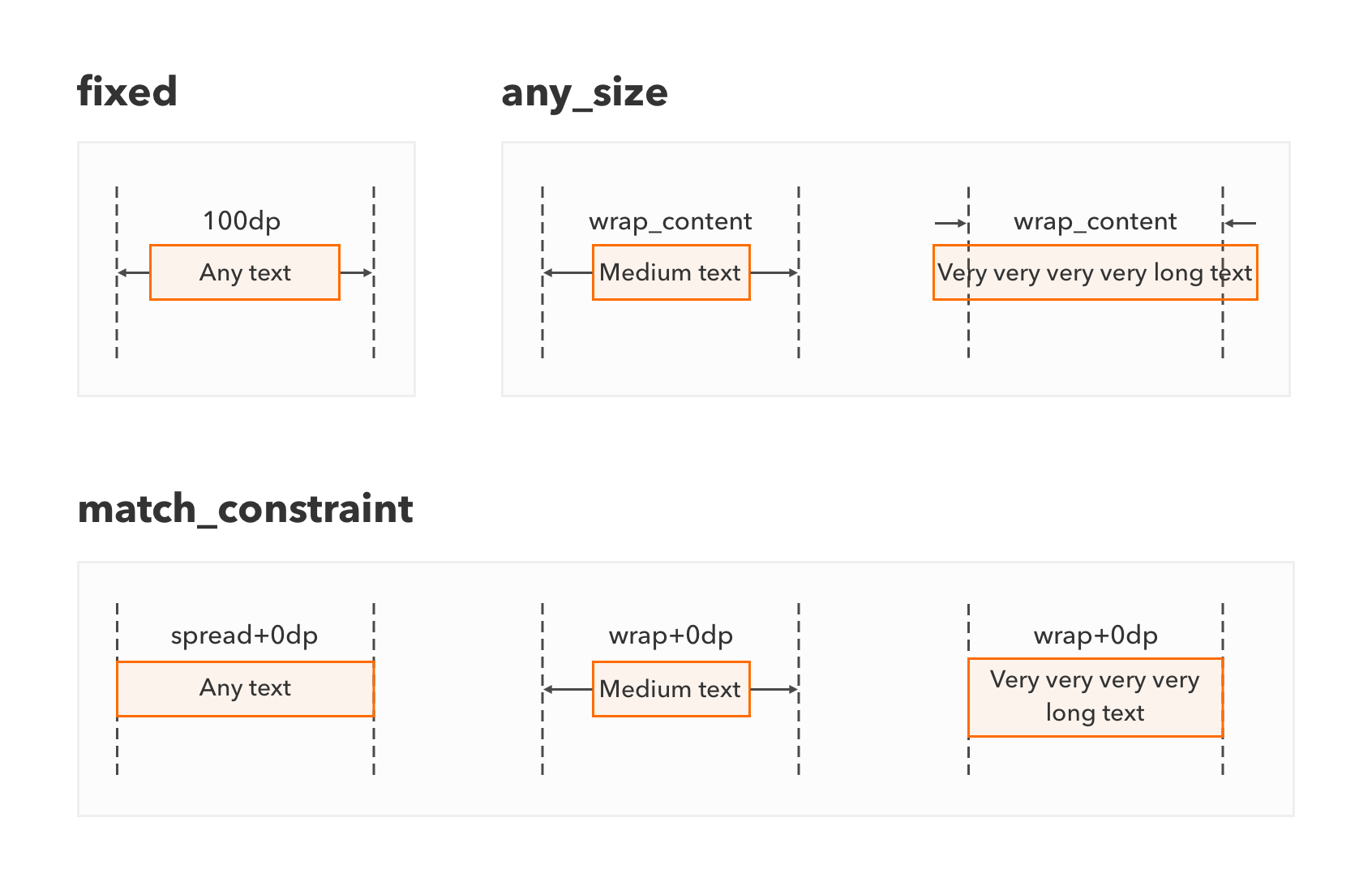
**Важно!** Указывать размер match\_parent или fill\_parent запрещено. Чтобы размер view совпадал с размерами ConstraintLayout, достаточно просто привязать constraints к сторонам ConstraintLayot и использовать размер match\_constraint\_spread.

Если указан размер match\_constraint\_wrap или match\_constraint\_spread, стоит учесть, что:

* чтобы такой тип размера работал корректно, у view должны быть привязаны два constraint: для ширины это Left и Right или Start и End. Для высоты — Top и Bottom;
* размер view не может выйти за рамки constraints;
* можно выставить минимальный и максимальный размер view в рамках constraints. Для этого используются атрибуты layout\_constraintWidth\_min, layout\_constraintHeight\_min, layout\_constraintWidth\_max, layout\_constraintHeight\_max;
* не стоит выставлять такой тип размера для высоты, если у view привязан Baseline constraint — вероятно, высота элемента будет рассчитываться неверно.

Для других типов размеров стоит учитывать, что:

* если указан any\_size или fixed\_size, то элемент может выходить за рамки constraints. Например, в примере из раздела "Constraints", если в текстовых view задать длинный текст, то он будет выходить за рамки ConstraintLayout;
* на размеры ConstraintLayout влияют view с fixed\_size, any\_size и match\_constraint\_wrap. Если покажется, что размер ConstraintLayout рассчитан неверно, скорее всего, виновата одна из view с такими размерами;



Задаем размеры View на основе соотношения сторон

ConstraintLayout позволяет рассчитывать высоту или ширину view на основе заданного соотношения сторон. То есть, например, при соотношении сторон 16:9, если высота будет 900dp, то ширина рассчитается, как 1600dp.

За это отвечает атрибут layout\_constraintDimensionRatio. Задать соотношение сторон можно в двух форматах: текстовом 16:9 или числовом 1.8. При этом перед значением можно указать символ стороны, которая находится в числителе соотношения. Например, H,16:9 будет означать, что 16 — это значение, соотвествующее высоте (H), а 9 — ширине (W).

Значение в layout\_constraintDimensionRatio учитывается при расчете размеров view, только если хотя бы одна из сторон выставлена в match\_constraint\_wrap или match\_constraint\_spread.

Пример:

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<!-- Соотношение сторон 16:9

Все constraints у view привязаны к сторонам ConstraintLayout.

Ширина view будет равна ширине ConstraintLayout,

так как layout\_width — match\_constraint\_spread.

Высота view будет рассчитана на основе соотношения,

так как layout\_height — any\_size. -->

<ImageView

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:scaleType="centerCrop"

app:layout\_constraintDimensionRatio="16:9"

app:layout\_constraintLeft\_toLeftOf="parent"

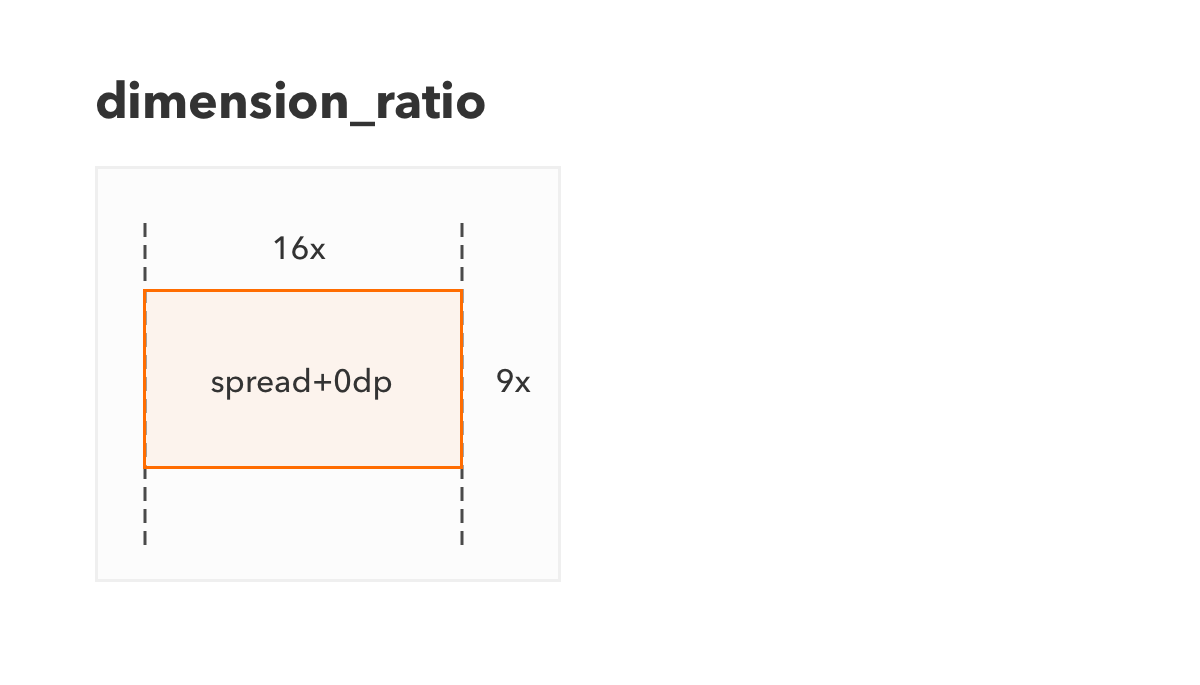
app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintBottom\_toBottomOf="parent"

android:background="@drawable/big\_widescreen\_image"/>

</android.support.constraint.ConstraintLayout>



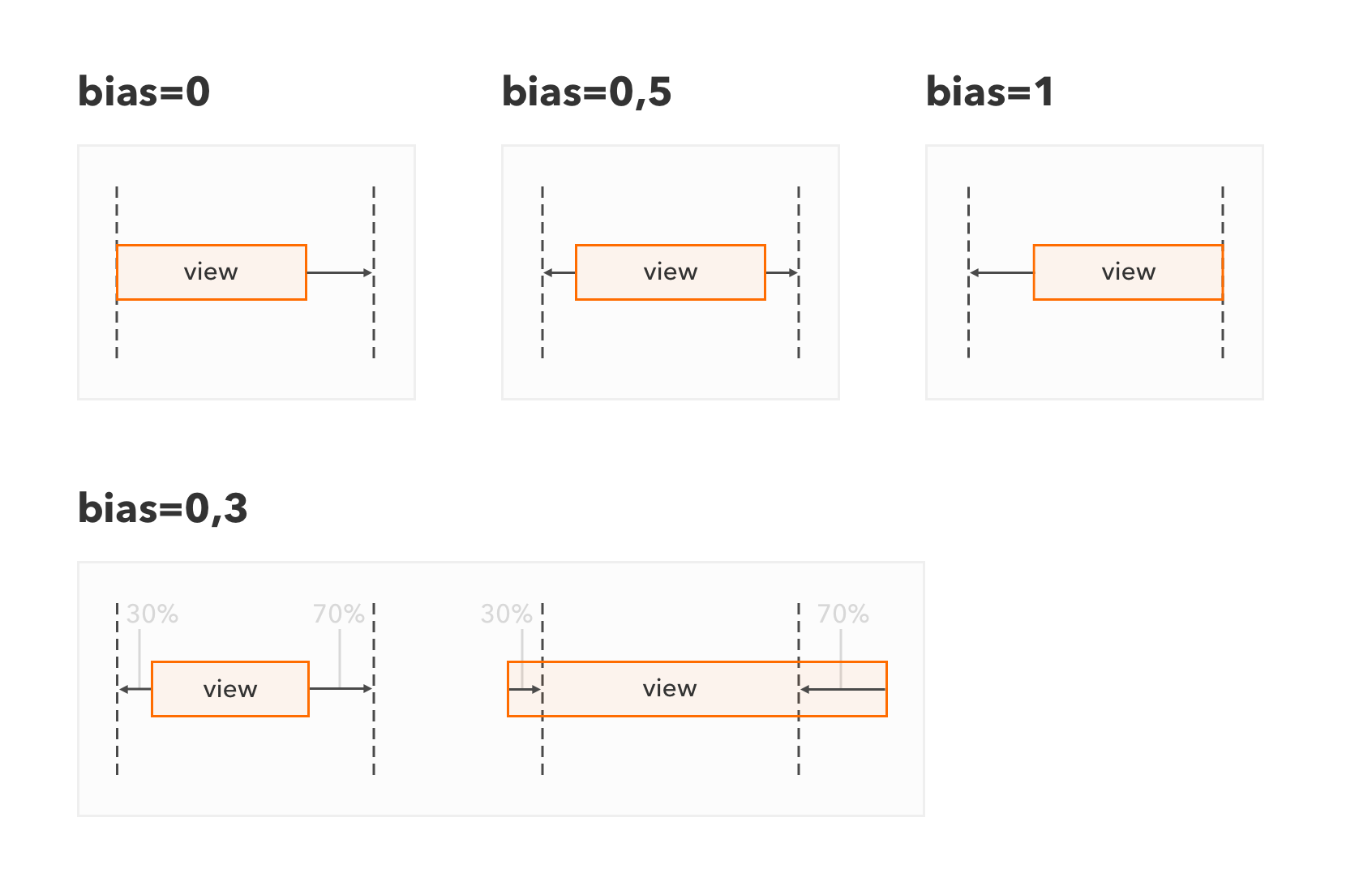
Выставляем относительное расположение View внутри ConstraintLayout

Если у view привязать два горизонтальных constraints, то ей можно выставить горизонтальное относительное расположение. То же применимо и для вертикальных constraints.

За горизонтальное расположение отвечает атрибут layout\_constraintHorizontal\_bias, за вертикальное — layout\_constraintVertical\_bias. Указывается относительное расположение значением от 0 до 1.

По сути, это более гибкая замена атрибута layout\_gravity. Например, для горизонтального расположения **0** будет означать расположение **крайне слева**, **0.5** — **по центру**, **1** — **крайне справа**. По умолчанию — **0.5**.

Теперь, например, выставим значение **0.3**. Это будет означать, что **30%** не заполненного view места будет слева от view, а **70%** — справа. Если же размер view больше размера расстояния между constraints, то **30%** выходящего за constraints размера будет слева от ограничений, а **70%** — справа.



Небольшое важное замечание: если в манифесте выставлена поддержка RTL языков, то layout\_constraintHorizontal\_bias вместо "слева" будет располагать элементы "от начала", а вместо "справа" — "от конца". То есть тем, кто поддерживает RTL языки стоит учитывать, что явно выставить расположение "слева" и "справа" не выйдет. По крайней мере, я такой возможности не нашел.

Пример:

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<!-- Горизонтальный bias выставлен в 0.3 (30% отступ слева/от начала).

Вертикальный bias — в 1 (снизу).

Constraints привязаны к сторонам контейнера.

Ширина — match\_constraint\_wrap, высота — match\_constraint\_wrap. -->

<TextView

android:id="@+id/view\_1"

android:text="view\_1"

android:layout\_width="0dp"

android:layout\_height="0dp"

app:layout\_constraintWidth\_default="wrap"

app:layout\_constraintHeight\_default="wrap"

android:scaleType="centerCrop"

app:layout\_constraintHorizontal\_bias="0.3"

app:layout\_constraintVertical\_bias="1"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="parent"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintBottom\_toBottomOf="parent"/>

</android.support.constraint.ConstraintLayout>

Особенности привязки линий

После презентации ConstraintLayout его часто сравнивали с RelativeLayout. Но, на самом деле, у них принципиально разные расчеты расположения элементов. В RelativeLayout у view просто указывается, с какой стороны другой view ей нужно находиться — "слева от", "справа от" и т.д. В ConstraintLayout constraints привязываются к сторонам других views и расположение view зависит от того, как ее constraints будут рассчитаны.

Для расположения constraint сперва рассчитывается расположение view, к которой этот constraint привязан. А для расположения view сперва рассчитываются все указанные для нее constraints. Циклические зависимости view и constraints при этом запрещены, так что, фактически, внутри ConstraintLayout строится направленный ациклический граф зависимостей constraints от view и view от constraints. Все расчеты производятся последовательно, начиная от не зависимых элементов графа.

Для простоты советую разделять вертикальные и горизонтальные зависимости на две независимые группы, как-будто внутри ConstraintLayout строится отдельно граф вертикальных зависимостей и отдельно — горизонтальных.

Но, вообще говоря, стоит понимать, что вертикальные параметры view, косвенно, могут зависеть от горизонтальных параметров другой view. Пример — расчет размера на основе соотношения сторон: при изменении ширины view меняется и ее высота. То есть, если у view изменится ширина из-за изменений ее горизонтальных constraints, то высота view тоже изменится.

Теперь рассмотрим любопытный пример расчета вертикальных constraints:

<android.support.constraint.ConstraintLayout

xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:app="http://schemas.android.com/apk/res-auto"

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

<TextView

android:id="@+id/view\_A"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="View A"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintTop\_toTopOf="parent"/>

<TextView

android:id="@+id/view\_B"

android:text="View B"

android:layout\_width="wrap\_content"

android:layout\_height="0dp"

app:layout\_constraintTop\_toBottomOf="@id/view\_A"

app:layout\_constraintBottom\_toTopOf="@id/view\_A"/>

<TextView

android:id="@+id/view\_C"

android:text="View C"

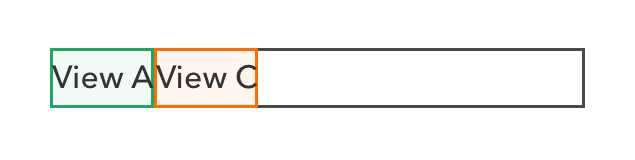
android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

app:layout\_constraintLeft\_toRightOf="@id/view\_A"

app:layout\_constraintTop\_toBottomOf="@id/view\_B"/>

</android.support.constraint.ConstraintLayout>

Результат:  


View A просто привязан к левой и верхней сторонам ConstraintLayout, то есть находится слева сверху.

View B привязан странным образом — Top(B)->bottom(A) и Bottom(B)->top(A), — расстояние между его вертикальными constraints, фактически, отрицательное. Сама высота B выставлена в match\_constraint\_spread.

View C находится справа от A — Left(C)toRight(A) — и (вроде как) снизу от B — Top(C)toBottom(B).

По горизонтальному расположению вопросов возникнуть не должно. Теперь объясню вертикальное расположение.

Последовательность вертикальных расчетов:

1. Для расчета C необходимо рассчитать ее нижний constraint;
2. Для расчета нижнего constraint C необходимо рассчитать верхнюю сторону B;
3. Для расчета верхней стороны B необходимо рассчитать ее нижний и верхний constraints;
4. Для расчета нижнего и верхнего constraints B необходимо рассчитать верхнюю и нижнюю стороны A;
5. A просто располагается слева-сверху, размеры рассчитывает сам.

Результаты вертикальных расчетов:

1. Верхняя сторона A находится на уровне верхней стороны ConstraintLayout, нижняя рассчитывается по размеру текста A, так как у A высота wrap\_content;
2. Верхний constraint B на уровне нижней стороны A, нижний привязан к верхней стороне A, то есть он на уровне верхней стороны ConstraintLayout;
3. Так как высота B — match\_constraint\_spread, то верхняя сторона B — на уровне нижней стороны A, а нижняя — на уровне верхней стороны ConstraintLayout. Это странно, но, фактически, высота B — отрицательная.
4. Верхний constraint C привязан к нижней стороне B, то есть он на уровне верхней стороны ConstraintLayout;
5. В итоге, верхняя сторона C на уровне верхней стороны ConstraintLayout, нижняя рассчитывается по размеру текста C, так как у A высота wrap\_content.

В общем, на мой взгляд, такой алгоритм расчета стоит учитывать, чтобы понимать, где будет располагаться view в конечном счете.

Особенности привязки по Baseline

View, привязанная по Baseline, не может быть ограничена сверху и снизу, то есть Top и Bottom constraints будут игнорироваться. Это значит, что для такой view нельзя выставить размер match\_constraint\_spread или match\_constraint\_wrap.

Из этого не совсем очевидно следует, что по Baseline стоит привязывать невысокие view к высоким. Иначе есть шанс, что высокая view выйдет за рамки ConstraintLayout или размер ConstraintLayout будет рассчитан неверно.

Пример некорректной Baseline-привязки:

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="wrap\_content">

*<!-- Размер текста 12sp -->*

<TextView

android:id="@+id/left\_view"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Left view"

android:textSize="12sp"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintTop\_toTopOf="parent"/>

*<!-- Размер текста 20sp,*

*то есть больший по размерам элемент привязан по базовой линии к меньшему -->*

<TextView

android:id="@+id/right\_view"

android:text="Right view"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:textSize="20sp"

app:layout\_constraintBaseline\_toBaselineOf="@id/left\_view"

app:layout\_constraintLeft\_toRightOf="@id/left\_view"/>

</android.support.constraint.ConstraintLayout>

Результат:  


Высота ConstraintLayout (в черной рамке) равна высоте большой TextView (в красной рамке), так как высота TextView выставлена, как wrap\_content.

Базовая линия большой TextView привязана к базовой линии малой TextView (в зеленой рамке), так что текст находится на одной линии.

При этом большая TextView выходит за рамки ConstraintLayout.

Создаем цепи (chains)

При привязке сторон есть одно интересное правило — если привязать две стороны двух элементов друг к другу Left(B)toRight(A) и Right(A)toLeft(B), то элементы будут выделены в цепь и к ним будут применяться особые правила расположения.

**Цепью** считается набор элементов, стороны которых привязаны друг к другу. Цепи определяются **автоматически** на основе привязок элементов внутри ConstraintLayout. Цепь располагается на основе привязок ее крайних элементов, а элементы внутри цепи располагаются по правилам определенного стиля цепи. Стиль цепи задается атрибутом layout\_constraint{X}\_chainStyle, где **X**— Horizontal для горизонтальных цепей или Vertical для вертикальных.

Пример цепи: правило Right(A)toLeft(B) + Left(B)toRight(A) свяжет элементы A и B в цепь, а Left(A)toLeft(parent) + Right(B)toRight(parent) привяжет всю цепь элементов к внешним сторонам ConstraintLayout.

Стиль цепи и его параметры берутся из атрибутов головного элемента цепи — самого левого, начального или самого верхнего.

Стиль spread

Элементы цепи распределяются равномерно, то есть отступы между элементами и от элементов до границ цепи будут одинаковые. Используется по умолчанию;

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- По умолчанию chainStyle — spread, так что значения атрибута можно не указывать -->*

<TextView

android:id="@+id/view\_1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_1"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toLeftOf="@+id/view\_2"/>

<TextView

android:id="@+id/view\_2"

android:layout\_width="100dp"

android:layout\_height="wrap\_content"

android:text="view\_2"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"

app:layout\_constraintRight\_toLeftOf="@+id/view\_3"/>

<TextView

android:id="@+id/view\_3"

android:layout\_width="200dp"

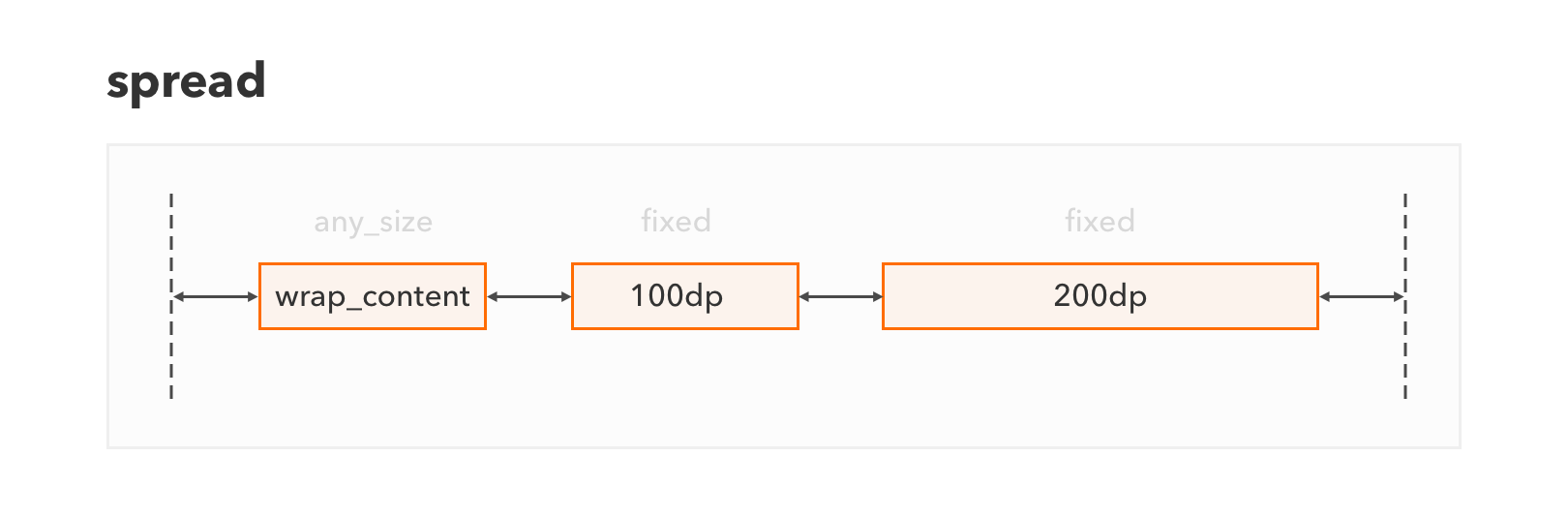
android:layout\_height="wrap\_content"

android:text="view\_3"

app:layout\_constraintLeft\_toRightOf="@id/view\_2"

app:layout\_constraintRight\_toRightOf="parent"/>

</android.support.constraint.ConstraintLayout>



Стиль spread\_inside

Элементы цепи распределяются так же, как и при стиле spread, но отступы от границ цепи всегда равны нулю;

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- ChainStyle головного элемента — spread\_inside -->*

<TextView

android:id="@+id/view\_1"

app:layout\_constraintHorizontal\_chainStyle="spread\_inside"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_1"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toLeftOf="@+id/view\_2"/>

<TextView

android:id="@+id/view\_2"

android:layout\_width="100dp"

android:layout\_height="wrap\_content"

android:text="view\_2"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"

app:layout\_constraintRight\_toLeftOf="@+id/view\_3"/>

<TextView

android:id="@+id/view\_3"

android:layout\_width="200dp"

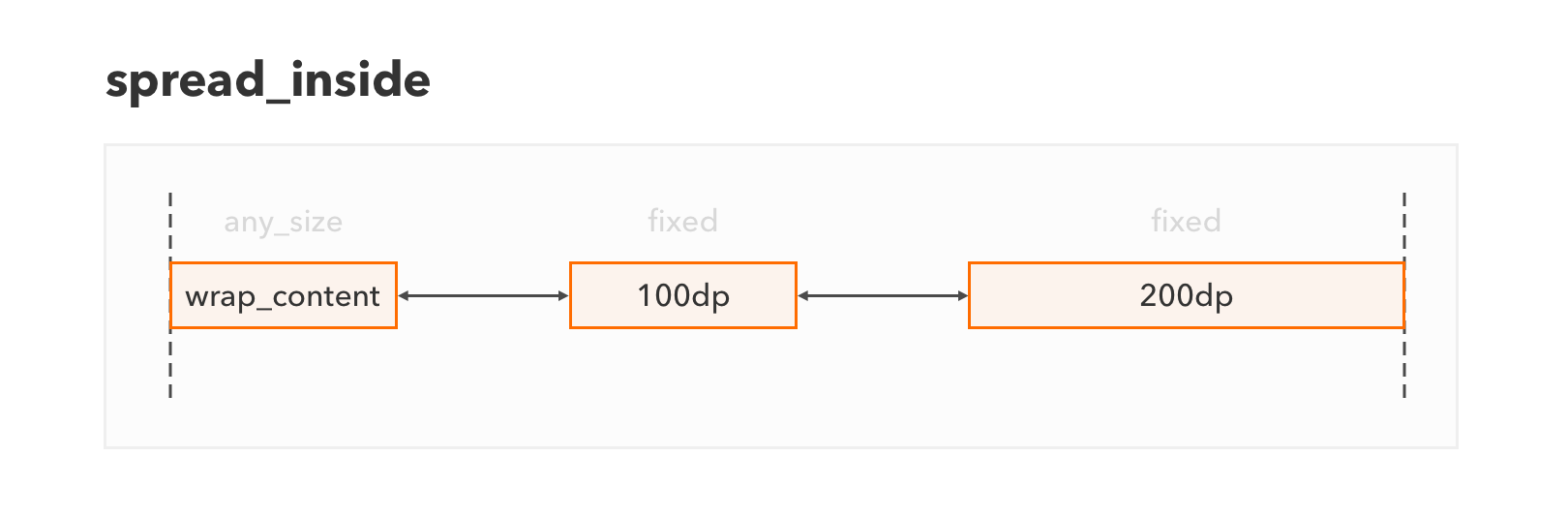
android:layout\_height="wrap\_content"

android:text="view\_3"

app:layout\_constraintLeft\_toRightOf="@id/view\_2"

app:layout\_constraintRight\_toRightOf="parent"/>

</android.support.constraint.ConstraintLayout>



Стиль packed

Элементы располагаются группой друг за другом. Такой стиль позволяет устанавливать относительную позицию группы элементов в доступном цепи пространстве через атрибут layout\_constraint{\*}\_bias. Bias атрибут нужно указывать у головного элемента цепи;

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- ChainStyle головного элемента — packed, bias — 0.3 (30% отступ слева/от начала) -->*

<TextView

android:id="@+id/view\_1"

app:layout\_constraintHorizontal\_chainStyle="packed"

app:layout\_constraintHorizontal\_bias="0.3"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_1"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toLeftOf="@+id/view\_2"/>

<TextView

android:id="@+id/view\_2"

android:layout\_width="100dp"

android:layout\_height="wrap\_content"

android:text="view\_2"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"

app:layout\_constraintRight\_toLeftOf="@+id/view\_3"/>

<TextView

android:id="@+id/view\_3"

android:layout\_width="200dp"

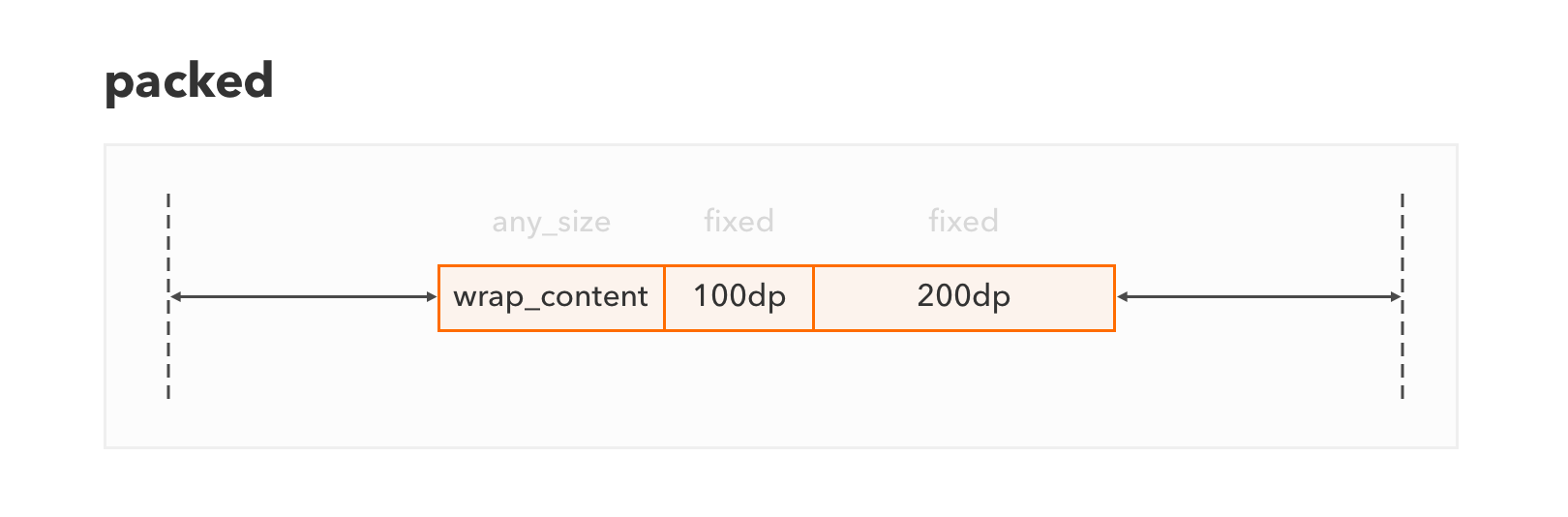
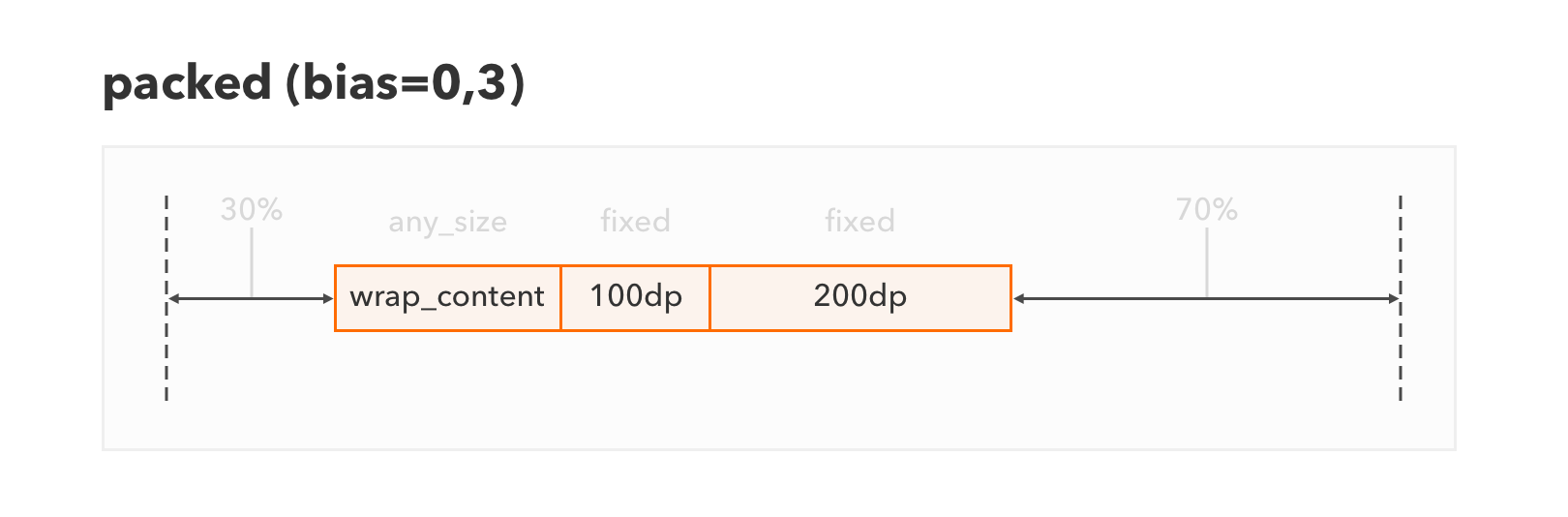
android:layout\_height="wrap\_content"

android:text="view\_3"

app:layout\_constraintLeft\_toRightOf="@id/view\_2"

app:layout\_constraintRight\_toRightOf="parent"/>

</android.support.constraint.ConstraintLayout>

Стиль weighted

Элементы располагаются в соответствии с их весом по аналогии с тем, как работает LinearLayout. Чтобы такой стиль заработал, одна из view цепи должна иметь размер match\_constraint\_spread. Для указания веса элемента используются атрибуты layout\_constraintHorizontal\_weight и layout\_constraintVertical\_weight.

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- Ширина match\_constraint\_spread, weight=3,*

*по ширине займет 3/4 свободного пространства -->*

<TextView

android:id="@+id/view\_1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

app:layout\_constraintHorizontal\_weight="3"

android:text="view\_1"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toLeftOf="@+id/view\_2"/>

*<!-- Ширина — match\_constraint\_spread, weight=1,*

*по ширине займет 1/4 свободного пространства -->*

<TextView

android:id="@+id/view\_2"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

app:layout\_constraintHorizontal\_weight="1"

android:text="view\_2"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"

app:layout\_constraintRight\_toLeftOf="@+id/view\_3"/>

*<!-- Ширина — any\_size, займет столько пространства, сколько ей нужно для отрисовки -->*

<TextView

android:id="@+id/view\_3"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="view\_3"

app:layout\_constraintLeft\_toRightOf="@id/view\_2"

app:layout\_constraintRight\_toRightOf="parent"/>

</android.support.constraint.ConstraintLayout>



Указываем отступы с учетом скрытых элементов

Отступы указываются стандартными атрибутами layout\_margin{X}, где **X** — сторона отступа (Left/Right/Top/Bottom/Start/End). Эти отступы применяются к линиям привязки элемента, а не к сторонам элемента. Это значит, что линия привязки будет с отступом от стороны, к которой она привязана.

Отдельные правила были введены для скрытых элементов, то есть элементов, у которых Visibilityвыставлено в значение GONE. Когда элемент скрыт, обычные отступы от его сторон игнорируются, но используются специальные gone-отступы. Его размеры при этом при расчетах считаются равными нулю. Gone-отступы представлены атрибутами: layout\_goneMargin{X}, где **X** — сторона отступа.

Другими словами, допустим, у элемента A выставлен отступ слева от элемента B равный 10dp, а gone-отступ слева выставлен 50dp. Если элемент B скрыт (GONE), то отступ элемента A слева будет 50dp, если элемент B — VISIBLE или INVISIBLE, то отступ элемента A слева будет 10dp.

Пример:

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

<TextView

android:id="@+id/view\_1"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:visibility="gone"

android:text="view\_1"

app:layout\_constraintLeft\_toLeftOf="parent"/>

*<!-- Левый constraint view\_2 привязан к правой стороне view\_1.*

*Так как view\_1 скрыта (gone), то используется отступ goneMarginLeft (50dp). -->*

<TextView

android:id="@+id/view\_2"

android:layout\_marginLeft="10dp"

app:layout\_goneMarginLeft="50dp"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

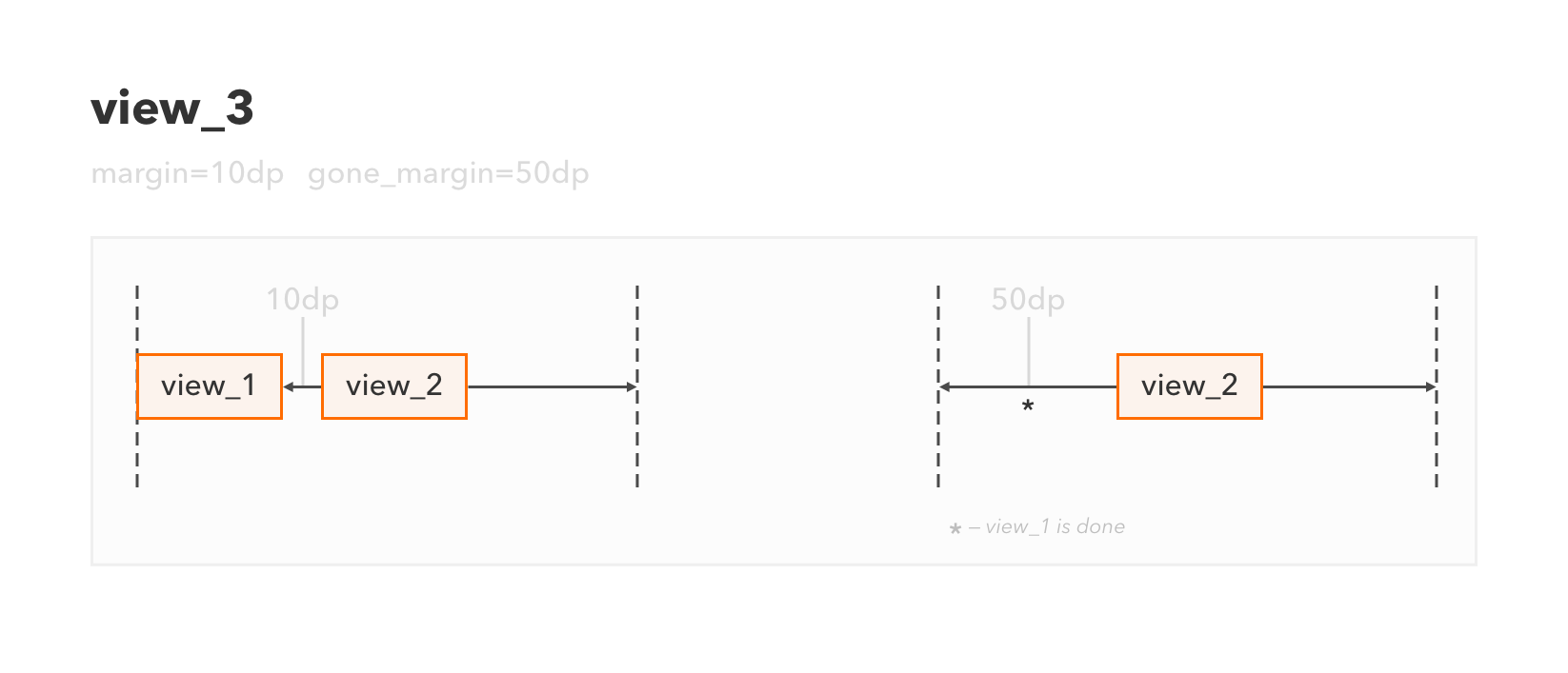
android:text="view\_2"

app:layout\_constraintHorizontal\_bias="0"

app:layout\_constraintLeft\_toRightOf="@id/view\_1"

app:layout\_constraintRight\_toRightOf="parent"/>

</android.support.constraint.ConstraintLayout>



Разбираемся с Guidelines

**Guideline** — это аналог линии, устанавливаемой на макетах в визуальных редакторах, по которой дизайнеры выравнивают элементы. Такую линию представляет view класса android.support.constraint.Guideline. Guideline может быть горизонтальным или вертикальным — это указывается атрибутом android:orientation. Сам guideline нулевого размера, не занимает места в контейнере и всегда привязан только к сторонам ConstraintLayout.

Guideline используется, чтобы привязывать к нему стороны view выравнивая их тем самым по одной линии.

Расположение guideline контролируется тремя атрибутами:

* layout\_constraintGuide\_begin — отступ от левой (Left, а не Start) стороны ConstraintLayout для вертикальных gudeline и от верхней стороны — для горизонтальных;
* layout\_constraintGuide\_end — отступ от правой (Right, а не End) стороны ConstraintLayout для вертикальных guideline и от нижней стороны — для горизонтальных;
* layout\_constraintGuide\_percent — относительный отступ guideline в процентах от левой стороны ConstraintLayout для вертикальных gudeline и от верхней стороны — для горизонтальных. Указывается числом от 0 до 1.

<android.support.constraint.ConstraintLayout

android:layout\_width="match\_parent"

android:layout\_height="match\_parent">

*<!-- Вертикальный guideline с относительным отступом 25% -->*

<android.support.constraint.Guideline

android:id="@+id/line\_1"

android:orientation="vertical"

app:layout\_constraintGuide\_percent="0.25"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

*<!-- Вертикальный guideline с отступом 100dp от левой стороны ConstraintLayout -->*

<android.support.constraint.Guideline

android:id="@+id/line\_2"

android:orientation="vertical"

app:layout\_constraintGuide\_begin="100dp"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

*<!-- Вертикальный guideline с отступом 50dp от правой стороны ConstraintLayout -->*

<android.support.constraint.Guideline

android:id="@+id/line\_3"

android:orientation="vertical"

app:layout\_constraintGuide\_end="50dp"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"/>

<TextView

android:id="@+id/view\_1"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text="view\_1"

app:layout\_constraintLeft\_toLeftOf="@id/line\_1"

app:layout\_constraintRight\_toRightOf="parent"/>

<TextView

android:id="@+id/view\_2"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

android:text="view\_2"

app:layout\_constraintTop\_toBottomOf="@id/view\_1"

app:layout\_constraintLeft\_toLeftOf="@id/line\_1"

app:layout\_constraintRight\_toRightOf="@id/line\_3"/>

<TextView

android:id="@+id/view\_3"

android:layout\_width="0dp"

android:layout\_height="wrap\_content"

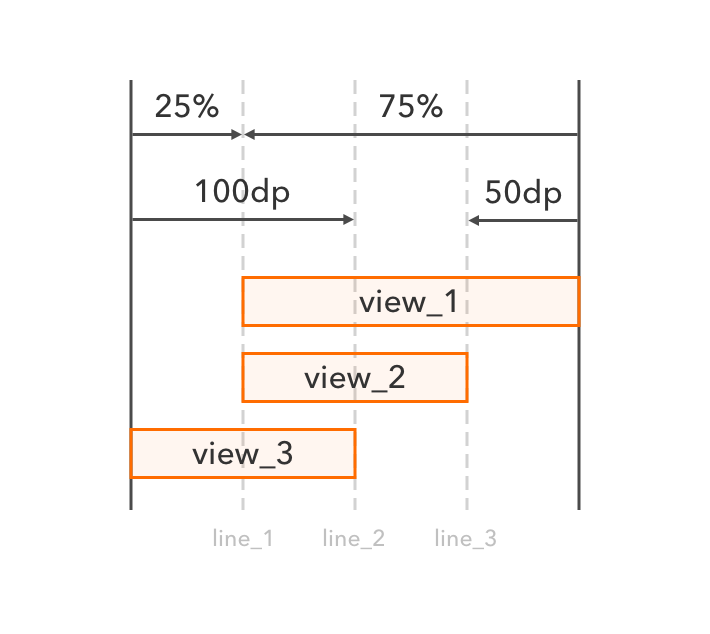
android:text="view\_3"

app:layout\_constraintTop\_toBottomOf="@id/view\_2"

app:layout\_constraintLeft\_toLeftOf="parent"

app:layout\_constraintRight\_toRightOf="@id/line\_2"/>

</android.support.constraint.ConstraintLayout>



Настраиваем параметры ConstraintLayout из кода

За это отвечает отдельный класс — android.support.constraint.ConstraintSet.

Создать этот ConstraintSet можно тремя способами:

* скопировать параметры из существующего ConstraintLayout;
* ConstraintLayout constraintLayout = LayoutInflater.from(context).inflate(R.layout.my\_constraint\_layout, null);
* ConstraintSet constraintSet = new ConstraintSet();

constraintSet.clone(constraintLayout);

* скопировать параметры из \*.xml файла, на основе которого создается ConstraintLayout;
* ConstraintSet constraintSet = new ConstraintSet();

constraintSet.clone(context, R.layout.my\_constraint\_layout);

* скопировать другой ConstraintSet.
* ConstraintSet constraintSet = new ConstraintSet();

constraintSet.clone(constraintSet2);

Описание методов для изменения ConstraintSet можно посмотреть в [документации](https://developer.android.com/reference/android/support/constraint/ConstraintSet.html) (их очень много).

Чтобы применить ConstraintSet к ConstraintLayout используйте метод applyTo.

final ConstraintLayout constraintLayout = (ConstraintLayout) findViewById(R.id.constr);

ConstraintSet constraintSet = new ConstraintSet();

constraintSet.clone(constraintLayout);

constraintSet.setHorizontalBias(R.id.sample\_view, 0.5f);

constraintSet.applyTo(constraintLayout);

Естественно, также можно напрямую изменять LayoutParams конкретного элемента и затем явно вызывать requestLayout. Класс, представляющий все параметры расположения элемента в ConstraintLayout — это android.support.constraint.ConstraintLayout.LayoutParams.

Коротко об анимации

Для анимаций в ConstraintLayout не предусмотрено специальных методов. ConstraintLayoutнаследуется от ViewGroup, так что для анимации расположения элементов используются те же инструменты, что и для обычных контейнеров: ValueAnimator, TransitionManager и так далее.

Стоит ли пользоваться ConstraintLayout?

Основные преимущества:

* обладает широким спектром возможностей, что позволяет реализовывать сложные взаиморасположения элементов на экране;
* в Android Studio для этого контейнера имеется удобный визуальный редактор;
* в большинстве случаев можно избежать излишней вложенности контейнеров друг в друга, что положительно влияет на производительность и читаемость кода;
* заменяет стандартные контейнеры, то есть достаточно выучить один раз ConstraintLayout и не пользоваться в дальнейшем такими контейнерами, как: FrameLayout, LinearLayout, RelativeLayout, GridLayout, PercentRelativeLayout;
* содержится в Support Library, так что баги будут правиться и, возможно, будут появляться новые функции.

Основные недостатки:

* в силу большого количества возможностей уходит больше времени на понимание, как он работает;
* при большом количестве элементов и взаимосвязей между ними сложно проводить код-ревью разметки — иерархическую вложенность контейнеров воспринимать проще, чем большое количество взаимосвязанных элементов в одном контейнере;
* в определенных случаях по производительности может быть хуже иерархии стандартных контейнеров, так как дополнительно требуется время на построение дерева зависимостей;
* несмотря на релизную версию, все еще можно наткнуться на неприятные баги;
* атрибутов у views будет гораздо больше, чем при использовании обычных layouts.

В компании Touch Instinct мы собираемся, как минимум, попробовать этот компонент. Особенно интересно посмотреть, как он поведет себя при использовании в ячейках RecyclerView — там часто изменяются значения элементов, что обычно приводит к пересчету расположения элементов в ячейке и иногда пересчету ее размера.

Разработчики Google проделали большую работу и у ConstraintLayout есть все шансы стать стандартным при разработке элементом UI, какими стали те же RecyclerView и CoordinatorLayout.