Flutter有生成构造函数、默认构造函数、命名构造函数、重定向构造函数、常量构造函数、工厂构造函数

## 一.生成构造函数 两种形式: this引当前类对象；语法糖

生成构造函数是最常见的构造函数，即生成实体类对象。

```
class Point {
  num x, y; //Dart中int和double是num的子类

  //this引用当前类对象
  Point(num x, num y) {
    this.x = x;
    this.y = y;
  }
//也可以用语法糖
  Point(this.x, this.y);
}
```

## 二.默认构造函数 没有参数

如果未声明构造函数，则会提供默认构造函数。 默认构造函数没有参数，并调用父类无参数构造函数。

```
class Parent{
  Parent(){
    print('In Parent\'s constructor.');
  }
}

class Child extends Parent{
  Child(){
```

```
    print('In Child\'s constructor.');
  }
}

//new Child(); -> 打印 In Parent's constructor.  In Child's constructor.
```

默认情况下，子类中的构造函数调用父类的未命名无参数构造函数。 父类的构造函数在子类构造函数体的开头被调用。 ~~如果还使用初始化子列表，则会在调用父类构造函数之前执行。~~ 执行顺序如下：

构造函数和初始化列表的执行顺序：
1.调用超类的无参构造函数（如果没有显式调用）。
2.执行构造函数体内的代码。
3.执行构造函数的初始化列表，按照它们在构造函数中声明的顺序。

~~初始化列表~~

~~父类的无参数构造函数~~

~~子类的无参数构造函数~~

如果父类没有未命名的无参数构造函数，则必须手动调用父类中的一个构造函数。 在子类的构造函数体之后用冒号 (:) 指定父类构造函数

```
class Parent{
  num x;
  num y;
  Parent(this.x, this.y){
    print('In Parent\'s constructor.');
  }
}

class Child extends Parent{
  Child(num x, num y) : super(x, y){
    print('In Child\'s constructor.');
```

```
  }
}

//new Child(100, 100); -> 打印
//In Parent's constructor.
//In Child's constructor.
```

## 三.命名构造函数 构造函数.XXX来命名构造函数

当需要定义一个有特别含义的构造函数的时候，可以通过命名构造 形式：构造函数.XXX来命名构造函数

```
class Point{
num x;      late num x;
num y;      late num y;

Point(this.x, this.y);

//创建一个坐标原点类
Point.origin(){
  this.x = 0;     x = 0;
  this.y = 0;     y = 0;
}

//创建一个坐标为(100, 100)的类
Point.coordinate100(){
  this.x = 100;     x = 100;
  this.y = 100;     y = 100;
}
@override
String toString() {
  return ("x: $x, y: $y");
}
}
```

类的主构造函数(主构造函数参数列表)

重定向构造函数（参数列表）: this(主构造函数参数列表)

## 四.重定向构造函数

有时构造函数需要重定向到同一个类中的另一个构造函数，在冒号后面用 this：

```
class Point {
  num x, y;

  //类的主构造函数
  Point(this.x, this.y);

  //重定向到主构造函数
  Point.alongXAxis(num x) : this(x, 0);
}
```

## 五.常量构造函数

const Point(this.x, this.y)
//在生成构造函数前加const

如果你的类需要成为永远不会更改的对象，则可以使这些对象成为编译时常量。 定义 const 构造函数要确保所有实例变量都是 final。

const构造函数创建的对象要赋值给const修饰的变量

static const point2origin = Point(0, 0);

```
class Point {
  final num x;
  final num y;
  static final Point2 origin = const Point2(0, 0);
  //const关键字放在构造函数名称之前，且不能有函数体
  const Point2(this.x, this.y);
}
```

const Point(this.x, this.y);

static const point2origin = Point(0, 0);

const Point(this.x, this.y);

## 六.工厂构造函数　　关键字 factory

不用直接创建对象（可以通过调用其他构造函数创建）

```
class CommonPrivacyScreen {
  final String title;
  final String url;

  factory CommonPrivacyScreen.privacy() {
    return CommonPrivacyScreen(title: "title_privacy",url: "url_privacy");
  }

  factory CommonPrivacyScreen.agreement() {
    return CommonPrivacyScreen(title: "title_agreement",url: "title_agreement");
  }

  CommonPrivacyScreen({Key key, this.title, this.url}) : super(key: key);
}
```

```
class CommonPrivacyScreen {
  final String title;
  final String url;
  factory CommonPrivacyScreen.privacy() {
    return CommonPrivacyScreen("title_privacy", "url_privacy");
  }
  factory CommonPrivacyScreen.agreement() {
    return CommonPrivacyScreen(
        "title_agreement", "title_agreement");
  }
  CommonPrivacyScreen(this.title, this.url);
}
```