# 6.1 可滚动组件简介

当组件内容超过当前显示视口(ViewPort)时，如果没有特殊处理，Flutter则会提示Overflow错误。为此，Flutter提供了多种可滚动组件（Scrollable Widget）用于显示列表和长布局。在本章中，我们先介绍一下常用的可滚动组件（如ListView、GridView等），然后介绍一下ScrollController。可滚动组件都直接或间接包含一个Scrollable组件，因此它们包括一些共同的属性，为了避免重复介绍，我们在此统一介绍一下：

Scrollable({  
 ...  
 this.axisDirection = AxisDirection.down,  
 this.controller,  
 this.physics,  
 @required this.viewportBuilder, //后面介绍  
})

* axisDirection滚动方向。
* physics：此属性接受一个ScrollPhysics类型的对象，它决定可滚动组件如何响应用户操作，比如用户滑动完抬起手指后，继续执行动画；或者滑动到边界时，如何显示。默认情况下，Flutter会根据具体平台分别使用不同的ScrollPhysics对象，应用不同的显示效果，如当滑动到边界时，继续拖动的话，在iOS上会出现弹性效果，而在Android上会出现微光效果。如果你想在所有平台下使用同一种效果，可以显式指定一个固定的ScrollPhysics，Flutter SDK中包含了两个ScrollPhysics的子类，他们可以直接使用：
  + ClampingScrollPhysics：Android下微光效果。
  + BouncingScrollPhysics：iOS下弹性效果。
* controller：此属性接受一个ScrollController对象。ScrollController的主要作用是控制滚动位置和监听滚动事件。默认情况下，Widget树中会有一个默认的PrimaryScrollController，如果子树中的可滚动组件没有显式的指定controller，并且primary属性值为true时（默认就为true），可滚动组件会使用这个默认的PrimaryScrollController。这种机制带来的好处是父组件可以控制子树中可滚动组件的滚动行为，例如，Scaffold正是使用这种机制在iOS中实现了点击导航栏回到顶部的功能。我们将在本章后面“滚动控制”一节详细介绍ScrollController。

### Scrollbar

Scrollbar是一个Material风格的滚动指示器（滚动条），如果要给可滚动组件添加滚动条，只需将Scrollbar作为可滚动组件的任意一个父级组件即可，如：

Scrollbar(  
 child: SingleChildScrollView(  
 ...  
 ),  
);

Scrollbar和CupertinoScrollbar都是通过ScrollController来监听滚动事件来确定滚动条位置的。关于ScrollController的详细内容我们将在本章最后一节中专门介绍。

#### CupertinoScrollbar

CupertinoScrollbar是iOS风格的滚动条，如果你使用的是Scrollbar，那么在iOS平台它会自动切换为CupertinoScrollbar。

### ViewPort视口

在很多布局系统中都有ViewPort的概念，在Flutter中，术语ViewPort（视口），如无特别说明，则是指一个Widget的实际显示区域。例如，一个ListView的显示区域高度是800像素，虽然其列表项总高度可能远远超过800像素，但是其ViewPort仍然是800像素。

### 基于Sliver的延迟构建

通常可滚动组件的子组件可能会非常多、占用的总高度也会非常大；如果要一次性将子组件全部构建出将会非常昂贵！为此，Flutter中提出一个Sliver（中文为“薄片”的意思）概念，如果一个可滚动组件支持Sliver模型，那么该滚动可以将子组件分成好多个“薄片”（Sliver），只有当Sliver出现在视口中时才会去构建它，这种模型也称为“基于Sliver的延迟构建模型”。可滚动组件中有很多都支持基于Sliver的延迟构建模型，如ListView、GridView，但是也有不支持该模型的，如SingleChildScrollView。

### 主轴和纵轴

在可滚动组件的坐标描述中，通常将滚动方向称为主轴，非滚动方向称为纵轴。由于可滚动组件的默认方向一般都是沿垂直方向，所以默认情况下主轴就是指垂直方向，水平方向同理。