Activity Fragment横竖屏生命周期变化

Activity

1、新建一个Activity，并把各个生命周期打印出来

2、运行Activity，得到如下信息

onCreate-->  
onStart-->  
onResume-->

3、按crtl+f12切换成横屏时

onSaveInstanceState-->  
onPause-->  
onStop-->  
onDestroy-->  
onCreate-->  
onStart-->  
onRestoreInstanceState-->  
onResume-->

4、再按crtl+f12切换成竖屏时，发现打印了两次相同的log

onSaveInstanceState-->  
onPause-->  
onStop-->  
onDestroy-->  
onCreate-->  
onStart-->  
onRestoreInstanceState-->  
onResume-->  
onSaveInstanceState-->  
onPause-->  
onStop-->  
onDestroy-->  
onCreate-->  
onStart-->  
onRestoreInstanceState-->  
onResume-->

5、修改AndroidManifest.xml，把该Activity添加 [**Android**](http://lib.csdn.net/base/android):configChanges="orientation"，执行步骤3

onSaveInstanceState-->  
onPause-->  
onStop-->  
onDestroy-->  
onCreate-->  
onStart-->  
onRestoreInstanceState-->  
onResume-->

6、再执行步骤4，发现不会再打印相同信息，但多打印了一行onConfigChanged

onSaveInstanceState-->  
onPause-->  
onStop-->  
onDestroy-->  
onCreate-->  
onStart-->  
onRestoreInstanceState-->  
onResume-->  
onConfigurationChanged-->

7、把步骤5的android:configChanges="orientation" 改成 android:configChanges="orientation|keyboardHidden"，执行步骤3，就只打印onConfigChanged

onConfigurationChanged-->

8、执行步骤4

onConfigurationChanged-->  
onConfigurationChanged-->

 总结：

1、不设置Activity的android:configChanges时，切屏会重新调用各个生命周期，切横屏时会执行一次，切竖屏时会执行两次

2、设置Activity的android:configChanges="orientation"时，切屏还是会重新调用各个生命周期，切横、竖屏时只会执行一次

3、设置Activity的android:configChanges="orientation|keyboardHidden"时，切屏不会重新调用各个生命周期，只会执行onConfigurationChanged方法

总结一下整个Activity的生命周期

补充一点，当前Activity产生事件弹出Toast和AlertDialog的时候Activity的生命周期不会有改变

Activity运行时按下HOME键(跟被完全覆盖是一样的)：onSaveInstanceState --> onPause --> onStop       onRestart -->onStart--->onResume

Activity未被完全覆盖只是失去焦点：onPause--->onResume

Fragment

[**fragment横竖屏切换时生命周期的变化**](http://blog.csdn.net/u012835548/article/details/52299871)

fragment

启动  
08-24 13:48:54.608 11678-11678/I/TAG: onAttach  
08-24 13:48:54.608 11678-11678/I/TAG: onCreate   
08-24 13:48:54.608 11678-11678/I/TAG: onCreateView   
08-24 13:48:54.617 11678-11678/I/TAG: onViewCreated   
08-24 13:48:54.618 11678-11678/I/TAG: onActivityCreated   
08-24 13:48:54.619 11678-11678/I/TAG: onStart   
08-24 13:48:54.619 11678-11678/I/TAG: onResume   
  
竖屏旋转到横屏（横屏旋转回竖屏也是执行同样的生命周期）  
08-24 13:49:44.243 11678-11678/I/TAG: onPause   
08-24 13:49:44.260 11678-11678/I/TAG: onStop   
08-24 13:49:44.266 11678-11678/I/TAG: onDestroyView   
08-24 13:49:44.267 11678-11678/I/TAG: onDestroy   
08-24 13:49:44.267 11678-11678/I/TAG: onDetach   
08-24 13:49:44.325 11678-11678/I/TAG: onAttach   
08-24 13:49:44.326 11678-11678/I/TAG: onCreate   
08-24 13:49:44.326 11678-11678/I/TAG: onCreateView   
08-24 13:49:44.328 11678-11678/I/TAG: onViewCreated   
08-24 13:49:44.330 11678-11678/I/TAG: onActivityCreated   
08-24 13:49:44.330 11678-11678/I/TAG: onStart   
08-24 13:49:44.330 11678-11678/I/TAG: onResume   
  
退出  
08-24 13:51:09.033 11678-11678/I/TAG: onPause   
08-24 13:51:09.721 11678-11678/I/TAG: onStop   
08-24 13:51:09.721 11678-11678/I/TAG: onDestroyView   
08-24 13:51:09.721 11678-11678/I/TAG: onDestroy   
08-24 13:51:09.721 11678-11678/I/TAG: onDetach

封装，继承，多态 ， 抽象， 接口

http://www.jb51.net/article/94829.htm

1. 封装

三大特性之---封装

       封装从字面上来理解就是包装的意思，专业点就是信息隐藏，是指利用抽象数据类型将数据和基于数据的操作封装在一起，使其构成一个不可分割的独立实体，数据被保护在抽象数据类型的内部，尽可能地隐藏内部的细节，只保留一些对外接口使之与外部发生联系。系统的其他对象只能通过包裹在数据外面的已经授权的操作来与这个封装的对象进行交流和交互。也就是说用户是无需知道对象内部的细节（当然也无从知道），但可以通过该对象对外的提供的接口来访问该对象。

       对于封装而言，一个对象它所封装的是自己的属性和方法，所以它是不需要依赖其他对象就可以完成自己的操作。

使用封装有三大好处：

1、良好的封装能够减少耦合。

2、类内部的结构可以自由修改。

3、可以对成员进行更精确的控制。

4、隐藏信息，实现细节。

2.继承

继承是:新定义的类是从已有的类中获取属性和方法的现象。 这个已有的类叫做父类， 从这个父类获取属性和方法的类叫做子类。

在java中使用extends关键字完成类的继承关系，操作格式如下：  
  
class 父类{}  
class 子类 extends 父类{}

继承的好处

1. 提高了代码的重用性

2. 让类和类之间产生了关系， 为多态创造了条件。

3.多态

 **多态的定义**：指允许不同类的对象对同一消息做出响应。即同一消息可以根据发送对象的不同而采用多种不同的行为方式。（发送消息就是函数调用）

**实现多态的技术称为**：动态绑定（dynamic binding），是指在执行期间判断所引用对象的实际类型，根据其实际的类型调用其相应的方法。

**多态的作用**：消除类型之间的耦合关系。

多态存在的三个必要条件  
一、要有继承；  
二、要有重写；  
三、父类引用指向子类对象。

**多态的好处**：

1.可替换性（substitutability）。多态对已存在代码具有可替换性。例如，多态对圆Circle类工作，对其他任何圆形几何体，如圆环，也同样工作。  
2.可扩充性（extensibility）。多态对代码具有可扩充性。增加新的子类不影响已存在类的多态性、继承性，以及其他特性的运行和操作。实际上新加子类更容易获得多态功能。例如，在实现了圆锥、半圆锥以及半球体的多态基础上，很容易增添球体类的多态性。  
3.接口性（interface-ability）。多态是超类通过方法签名，向子类提供了一个共同接口，由子类来完善或者覆盖它而实现的。如图8.3 所示。图中超类Shape规定了两个实现多态的接口方法，computeArea()以及computeVolume()。子类，如Circle和Sphere为了实现多态，完善或者覆盖这两个接口方法。  
4.灵活性（flexibility）。它在应用中体现了灵活多样的操作，提高了使用效率。  
5.简化性（simplicity）。多态简化对应用软件的代码编写和修改过程，尤其在处理大量对象的运算和操作时，这个特点尤为突出和重要。

Java中多态的实现方式：接口实现，继承父类进行方法重写，同一个类中进行方法重载。

4.抽象

抽象类就是为了继承而存在的，如果你定义了一个抽象类，却不去继承它，那么等于白白创建了这个抽象类

抽象类和普通类的主要有三点区别：

　　1）抽象方法必须为public或者protected（因为如果为private，则不能被子类继承，子类便无法实现该方法），缺省情况下默认为public。

　　2）抽象类不能用来创建对象；

　　3）如果一个类继承于一个抽象类，则子类必须实现父类的抽象方法。如果子类没有实现父类的抽象方法，则必须将子类也定义为为abstract类。

　　在其他方面，抽象类和普通的类并没有区别。

5接口

接口中可以含有 变量和方法。但是要注意，接口中的变量会被隐式地指定为public static final变量（并且只能是public static final变量，用private修饰会报编译错误），而方法会被隐式地指定为public abstract方法且只能是public abstract方法（用其他关键字，比如private、protected、static、 final等修饰会报编译错误），并且接口中所有的方法不能有具体的实现，也就是说，接口中的方法必须都是抽象方法。从这里可以隐约看出接口和抽象类的区别，接口是一种极度抽象的类型，它比抽象类更加“抽象”，并且一般情况下不在接口中定义变量。

　　要让一个类遵循某组特地的接口需要使用implements关键字，具体格式如下：

|  |  |
| --- | --- |
| 1  2 | class ClassName implements Interface1,Interface2,[....]{  } |

　　可以看出，允许一个类遵循多个特定的接口。如果一个非抽象类遵循了某个接口，就必须实现该接口中的所有方法。对于遵循某个接口的抽象类，可以不实现该接口中的抽象方法。

抽象接口区别

1）抽象类可以提供成员方法的实现细节，而接口中只能存在public abstract 方法；

　　2）抽象类中的成员变量可以是各种类型的，而接口中的成员变量只能是public static final类型的；

　　3）接口中不能含有静态代码块以及静态方法，而抽象类可以有静态代码块和静态方法；

　　4）一个类只能继承一个抽象类，而一个类却可以实现多个接口。

6依赖注入

依赖注入，全称是“依赖注入到容器”， 容器（IOC容器）是一个设计模式，它也是个对象，

你把某个类（不管有多少依赖关系）放入这个容器中，可以“解析”出这个类的实例。  
  
所以依赖注入就是把有依赖关系的类放入容器（IOC容器）中，然后解析出这个类的实例。

**为什么要有依赖注入（一种设计代码模式），因为我们要控制反转（设计代码的思路）。为什么控制反转。因为我们软件设计需要符合软件设计原则依赖倒置（设计代码原则），单一职责原则。**  
**说通俗点，咱们要解耦啊。**

依赖注入最大的两个好处是:

* 代码松耦合, 易维护
* 易测试