# View

**MotionEvent是什么？包含几种事件？什么条件下会产生？**

- 技术点：View触控
- 参考回答：MotionEvent是手指触摸屏幕锁产生的一系列事件。
  包含的事件有：
    - ACTION_DOWN：手指刚接触屏幕
    - ACTION_MOVE：手指在屏幕上滑动
    - ACTION_UP：手指在屏幕上松开的一瞬间
    - ACTION_CANCEL：手指保持按下操作，并从当前控件转移到外层控件时会触发

**scrollTo()和scrollBy()的区别?**

- 技术点：View滑动
- 参考回答：scrollBy内部调用了scrollTo，它是基于当前位置的相对滑动；而scrollTo是绝对滑动，因此如果利用相同输入参数多次调用scrollTo()方法，由于View初始位置是不变只会出现一次View滚动的效果而不是多次。
- 引申：两者都只能对view内容进行滑动，而不能使view本身滑动，且非平滑，可使用Scroller有过渡滑动的效果

**Scroller中最重要的两个方法是什么？主要目的是?**

- 技术点：View滑动
- 思路：从Scroller实现滑动的具体过程出发，
- 参考回答：Scroller实现滑动的具体过程：
    - 在MotionEvent.ACTION_UP事件触发时调用startScroll()方法，该方法并没有进行实际的滑动操作，而是记录滑动相关量
    - 马上调用invalidate/postInvalidate()方法，请求View重绘，导致View.draw方法被执行
    - 紧接着会调用View.computeScroll()方法，此方法是空实现，需要自己处理逻辑。具体逻辑是：先判断computeScrollOffset()，若为true（表示滚动未结束），则执行scrollTo()方法，它会再次调用postInvalidate()，如此反复执行，直到返回值为false。
    - 其中，最重要的两个方法是startScroll()和computeScroll()，computeScrollOffset()也是相当重要的。

**谈一谈View的事件分发机制?**

- 技术点：View事件分发
- 思路：从分发本质、传递顺序、核心方法展开

- 参考回答：
    - 事件分发本质：就是对MotionEvent事件分发的过程。即当一个MotionEvent产生了以后，系统需要将这个点击事件传递到一个具体的View上。
    - 点击事件的传递顺序：Activity（Window） -> ViewGroup -> View
    - 三个主要方法：
        - dispatchTouchEvent：进行事件的分发（传递）。返回值是 boolean 类型，受当前onTouchEvent和下级view的dispatchTouchEvent影响
        - onInterceptTouchEvent：对事件进行拦截。该方法只在ViewGroup中有，View（不包含ViewGroup）是没有的。一旦拦截，则执行ViewGroup的onTouchEvent，在ViewGroup中处理事件，而不接着分发给View。且只调用一次，所以后面的事件都会交给ViewGroup处理。
        - onTouchEvent：进行事件处理。

**如何解决View的滑动冲突?**

- 技术点：View滑动冲突
- 思路：从处理规则和具体实现方法展开讨论
- 参考回答：
    - （1）处理规则：
      对于由于外部滑动和内部滑动方向不一致导致的滑动冲突，可以根据滑动的方向判断谁来拦截事件。 对于由于外部滑动方向和内部滑动方向一致导致的滑动冲突，可以根据业务需求，规定何时让外部View拦截事件何时由内部View拦截事件。 对于上面两种情况的嵌套，相对复杂，可同样根据需求在业务上找到突破点。
    - （2）实现方法：
        - 外部拦截法：指点击事件都先经过父容器的拦截处理，如果父容器需要此事件就拦截，否则就不拦截。具体方法：需要重写父容器的onInterceptTouchEvent方法，在内部做出相应的拦截。
        - 内部拦截法：指父容器不拦截任何事件，而将所有的事件都传递给子容器，如果子容器需要此事件就直接消耗，否则就交由父容器进行处理。具体方法：需要配合requestDisallowInterceptTouchEvent方法。

**谈一谈View的工作原理?**

- 技术点：View工作流程
- 思路：围绕三大流程展开
- 参考回答：View工作流程简单来说就是，先measure测量，用于确定View的测量宽高，再 layout布局，用于确定View的最终宽高和四个顶点的位置，最后 draw绘制，用于将View 绘制到屏幕上。

- ViewRoot对应于ViewRootImpl类，它是连接WindowManager和DecorView的纽带。
- View的绘制流程是从ViewRoot和performTraversals开始。
- performTraversals()依次调用performMeasure()、performLayout()和performDraw()三个方法，分别完成顶级 View的绘制。
- 其中，performMeasure()会调用measure()，measure()中又调用onMeasure()，实现对其所有子元素的measure过程，这样就完成了一次measure过程；接着子元素会重复父容器的measure过程，如此反复至完成整个View树的遍历。layout和draw同理。

**onTouch()、onTouchEvent()和onClick()关系?**

- 技术点：View事件分发
- 参考回答：优先度onTouch()>onTouchEvent()>onClick()。因此onTouchListener的onTouch()方法会先触发；如果onTouch()返回false才会接着触发onTouchEvent()，同样的，内置诸如onClick()事件的实现等等都基于onTouchEvent()；如果onTouch()返回true，这些事件将不会被触发。

**SurfaceView和View的区别?**

- 技术点：View、SurfaceView
- 参考回答：SurfaceView是从View基类中派生出来的显示类，他和View的区别有：
    - View需要在UI线程对画面进行刷新，而SurfaceView可在子线程进行页面的刷新
    - View适用于主动更新的情况，而SurfaceView适用于被动更新，如频繁刷新，这是因为如果使用View频繁刷新会阻塞主线程，导致界面卡顿
    - SurfaceView在底层已实现双缓冲机制，而View没有，因此SurfaceView更适用于需要频繁刷新、刷新时数据处理量很大的页面

**invalidate()和postInvalidate()和RequestLayout的区别?**

- 技术点：View刷新
- 参考回答：invalidate()与postInvalidate()都用于刷新View，主要区别是invalidate()在主线程中调用，若在子线程中使用需要配合handler；而postInvalidate()可在子线程中直接调用。
- requestLayout方法会导致View的onMeasure、onLayout、onDraw方法被调用；invalidate方法则只会导致View的onDraw方法被调用