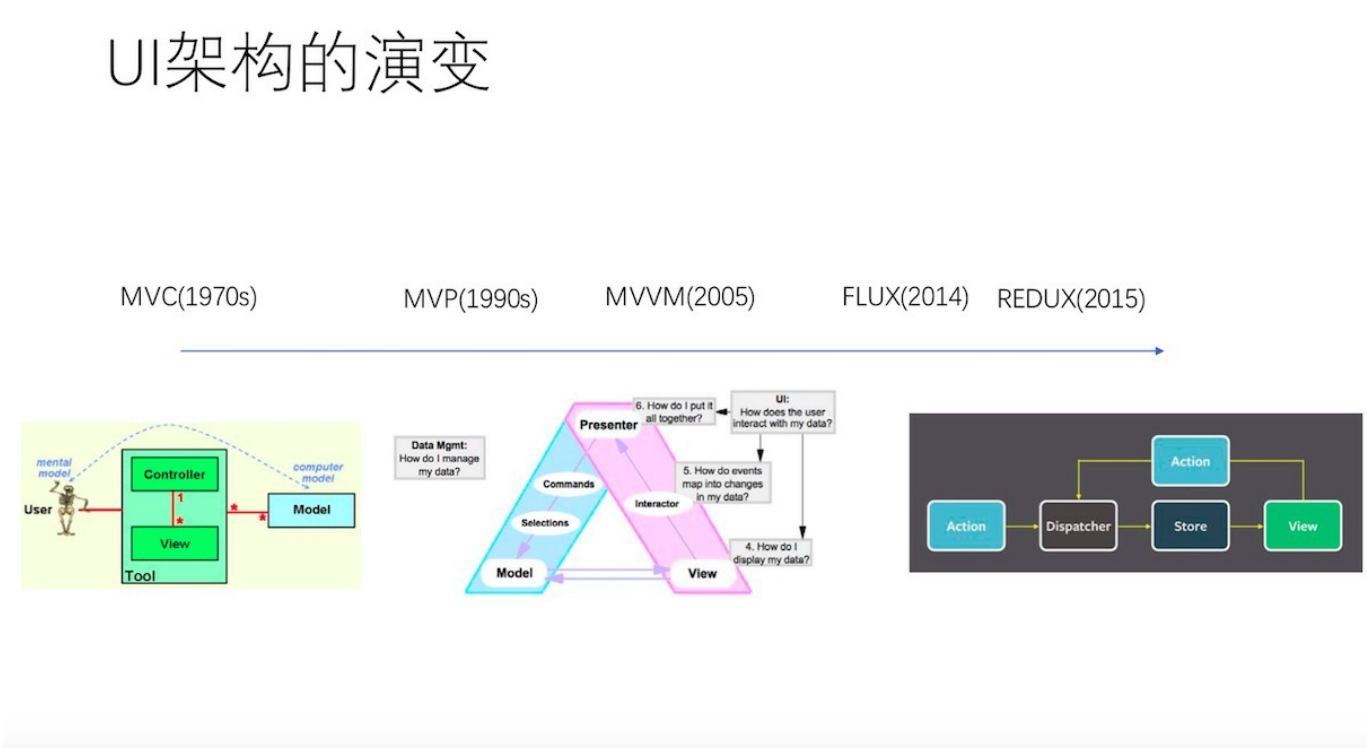
分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。



首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

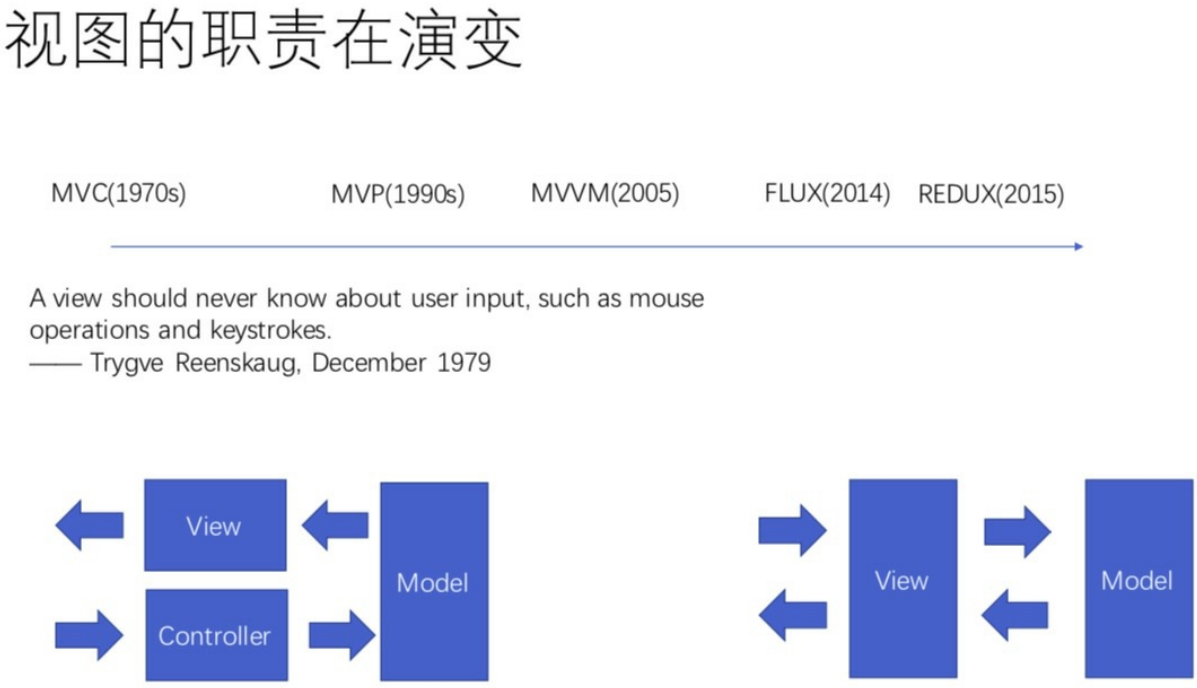
这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

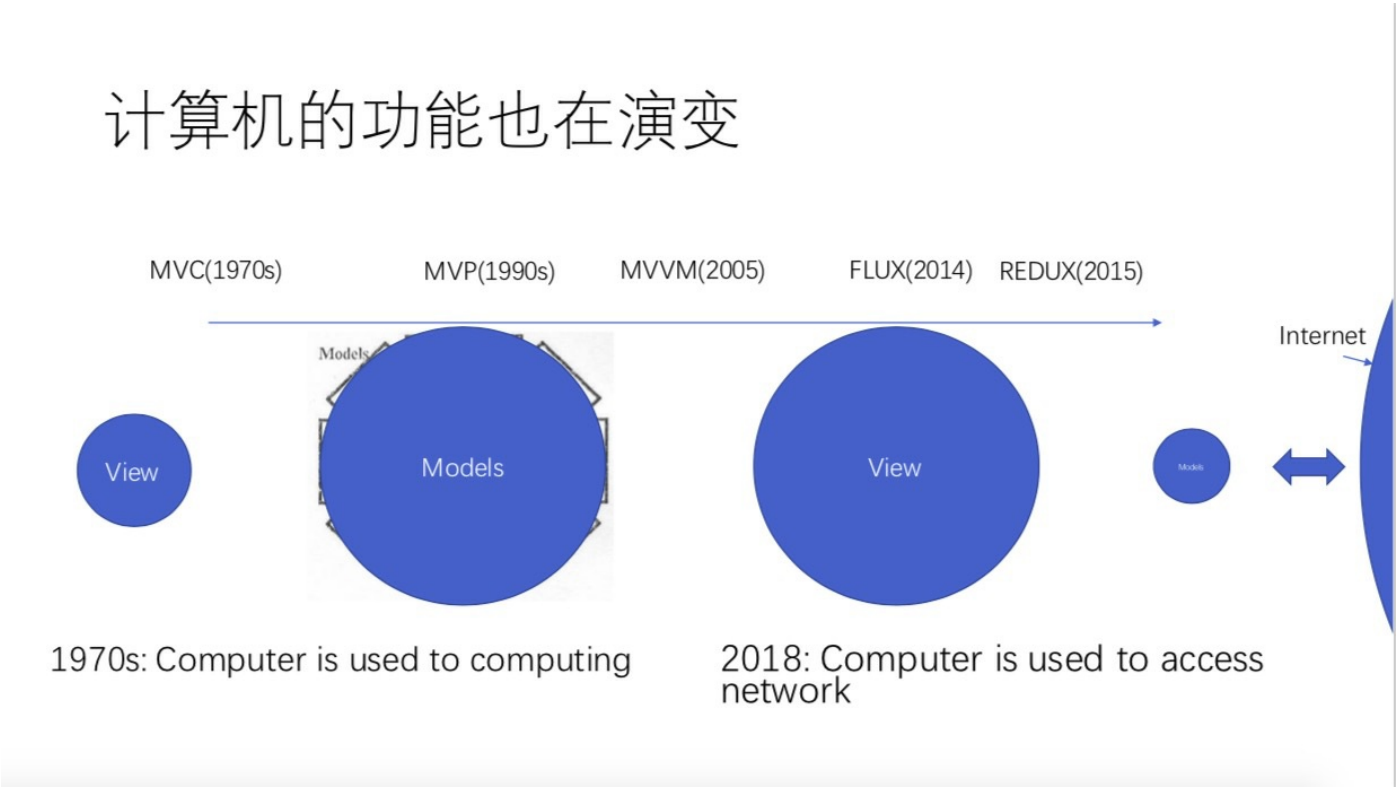
再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相

似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

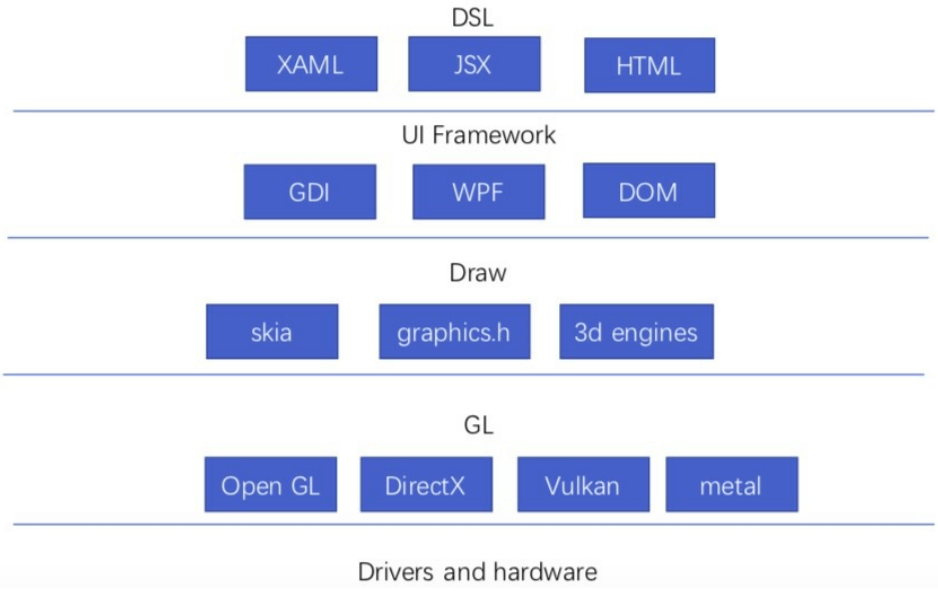


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

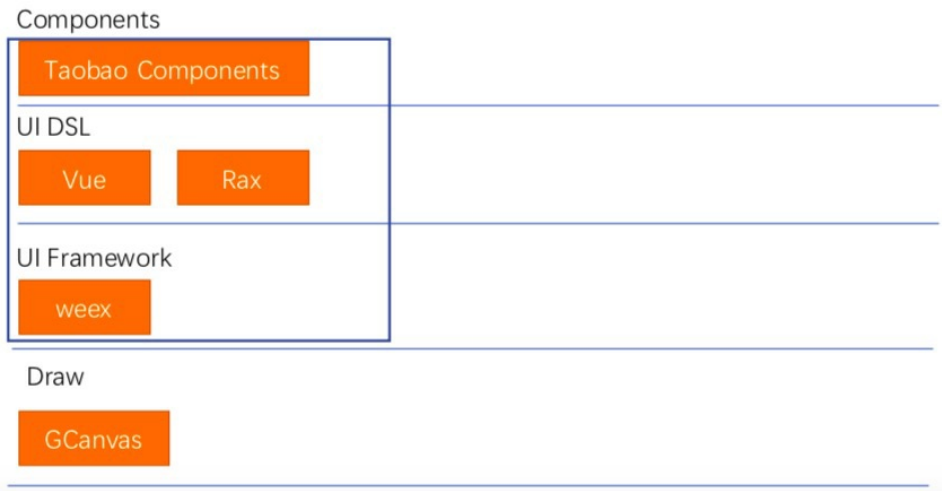
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

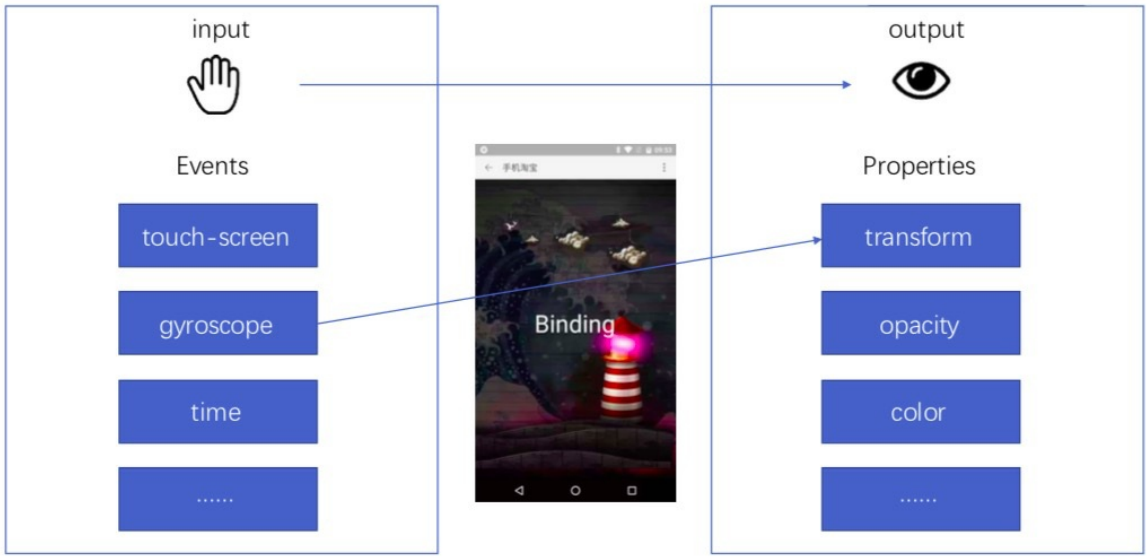
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

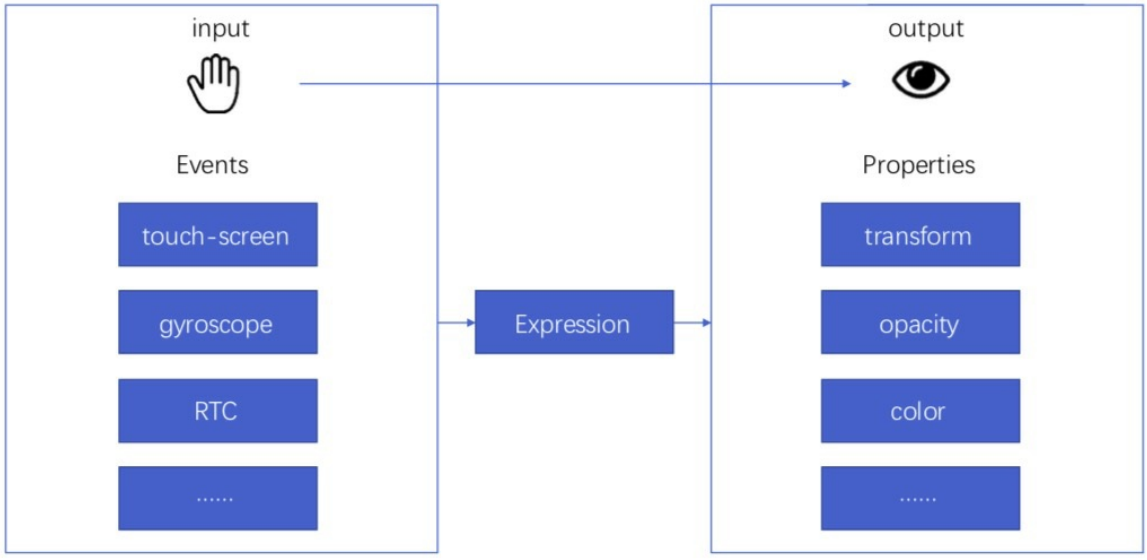


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

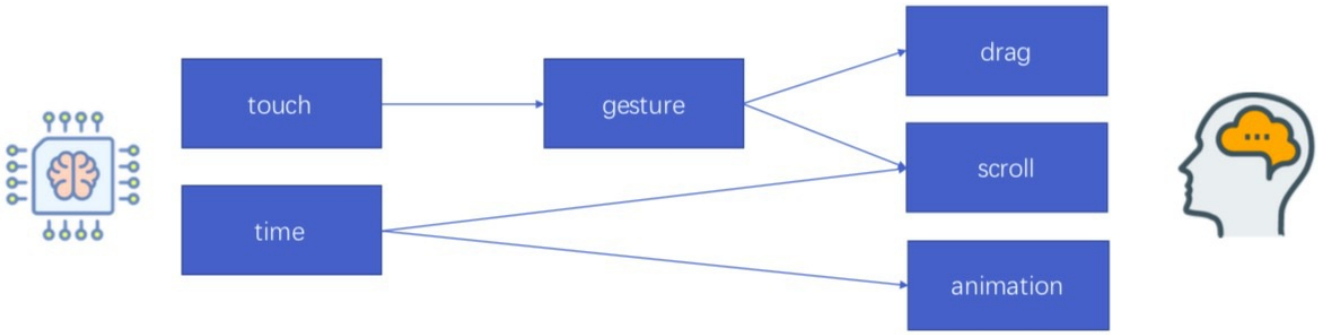
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

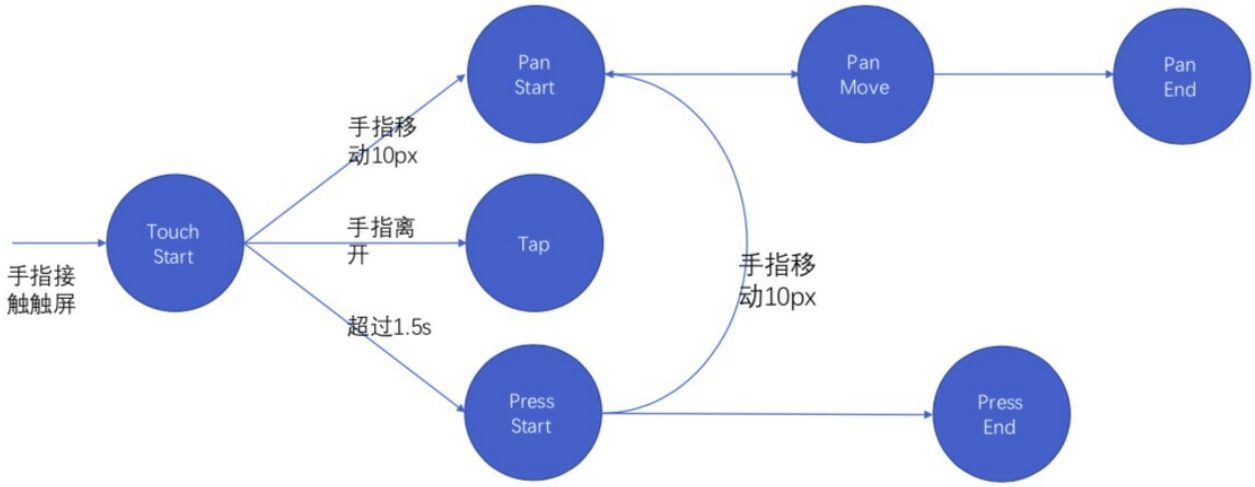
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

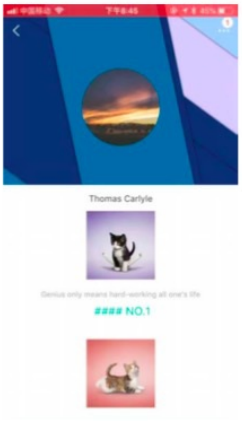
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

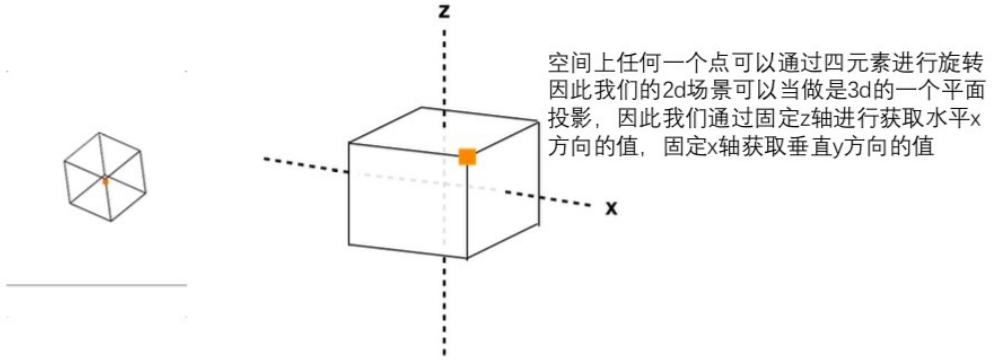
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

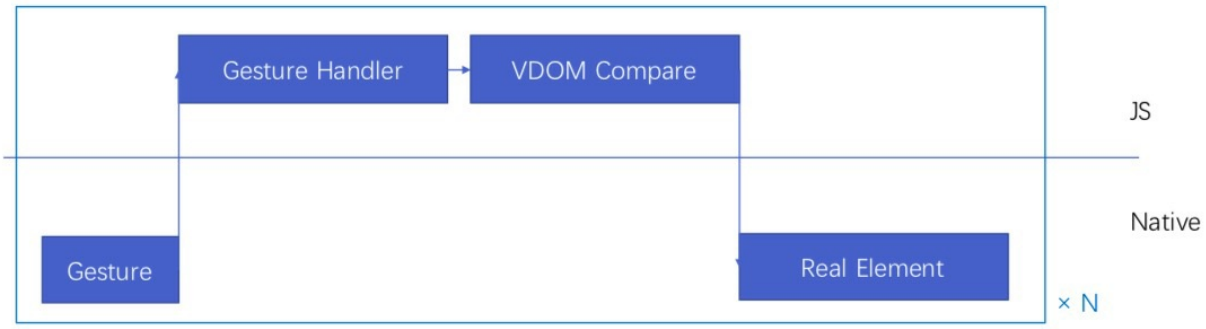
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

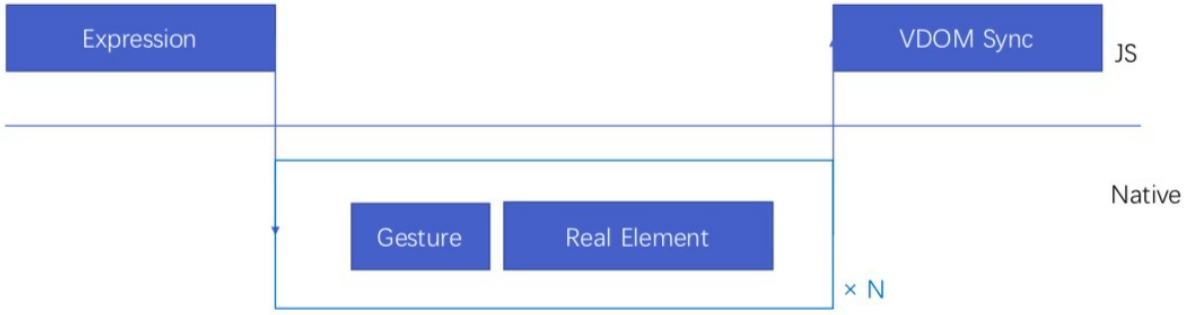
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

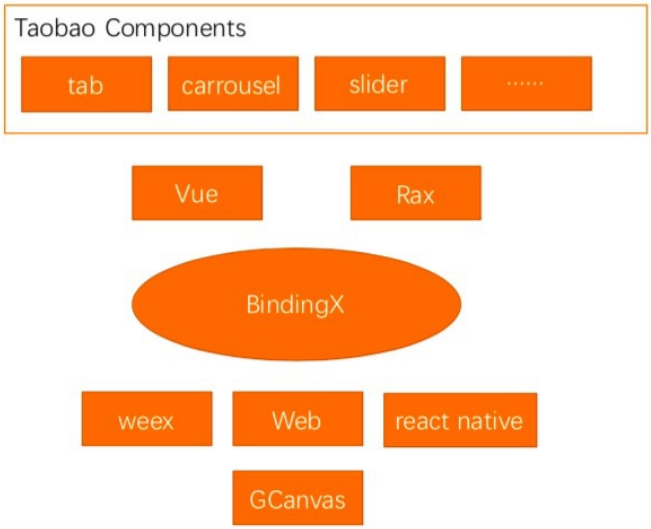
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

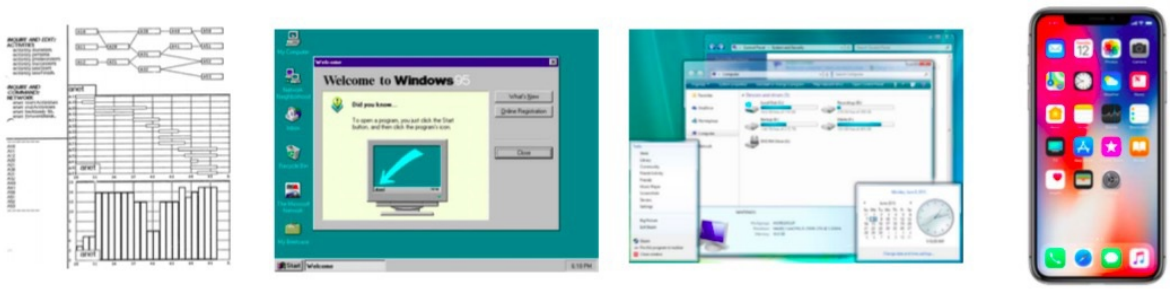
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

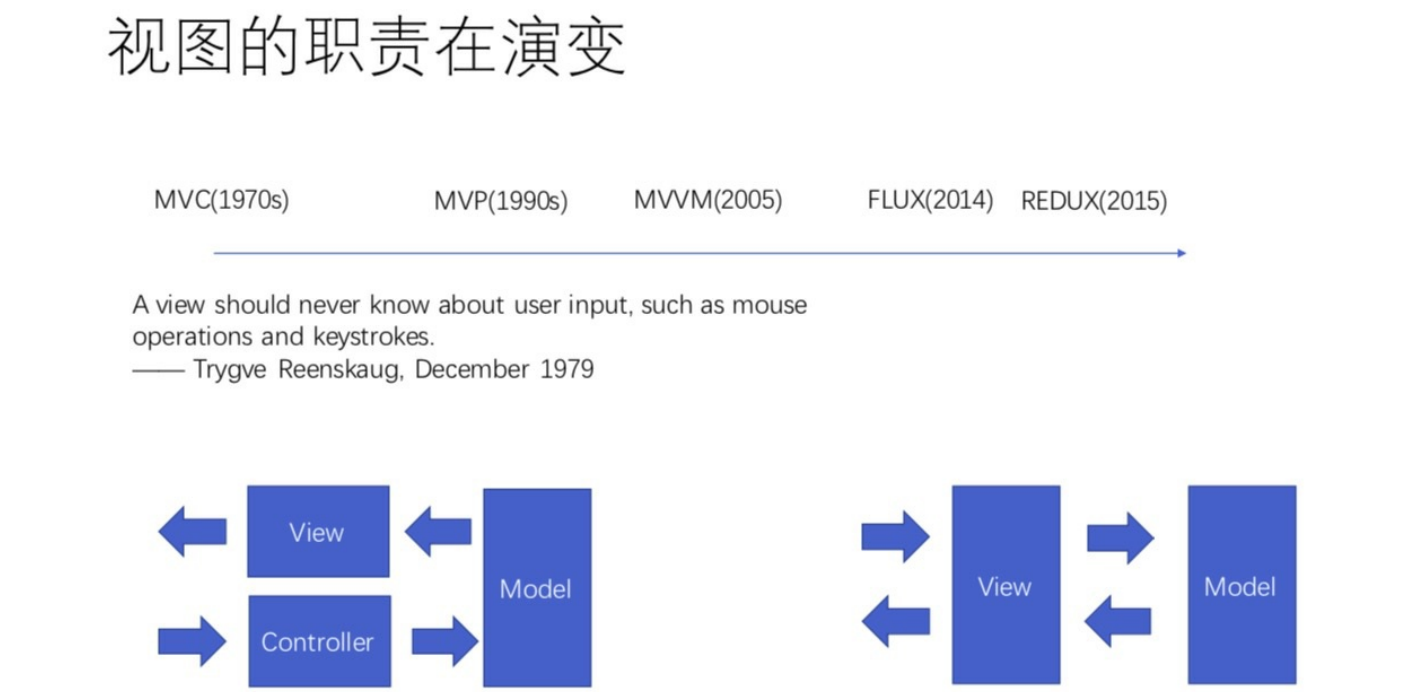


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

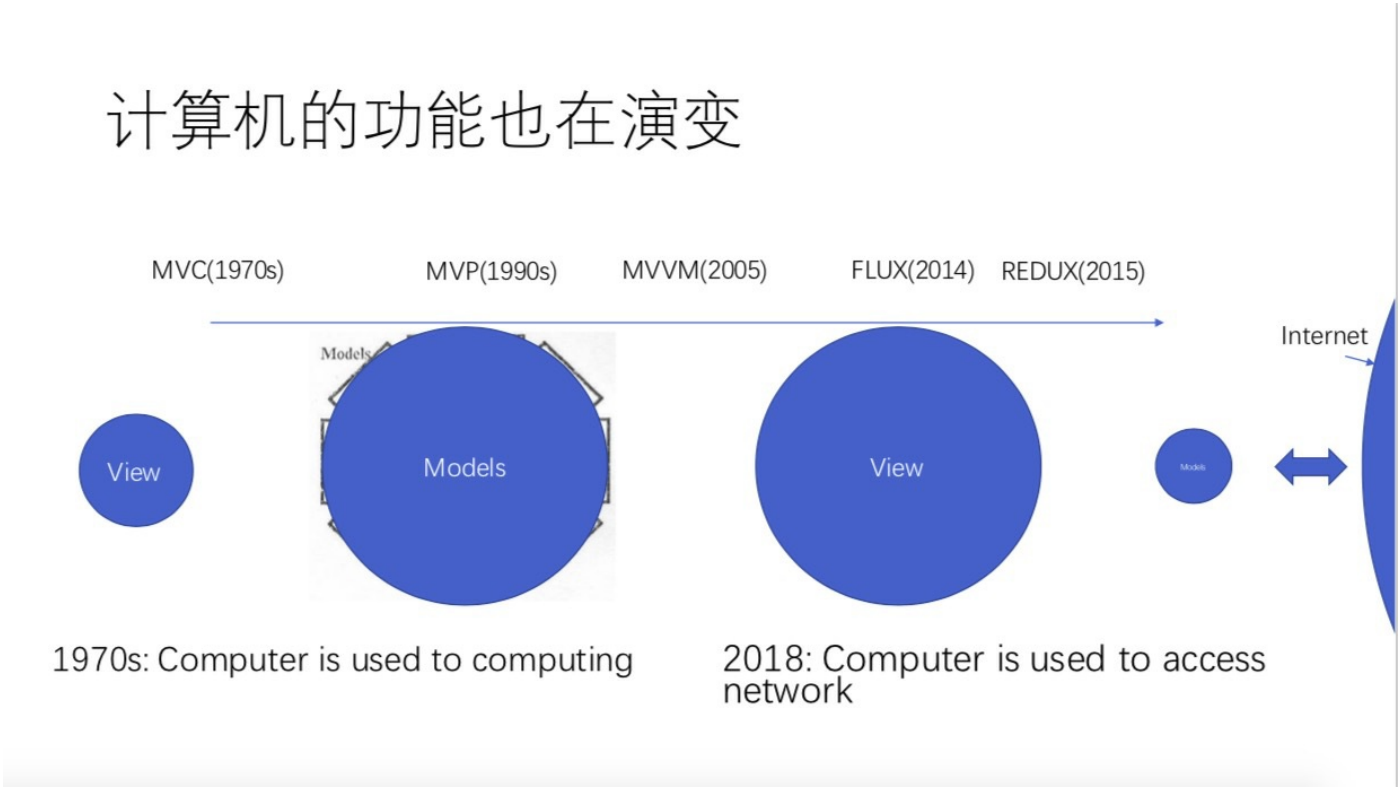
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

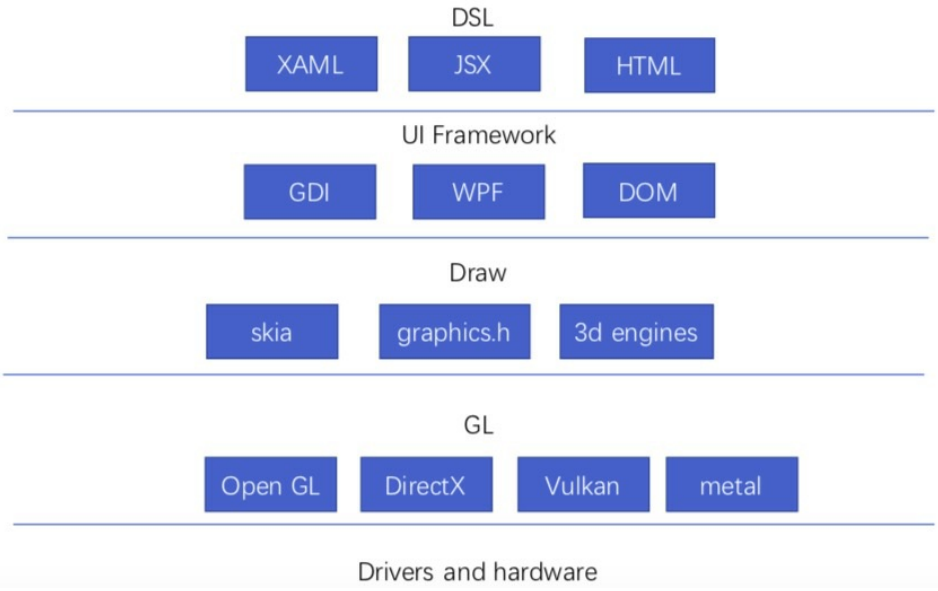


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

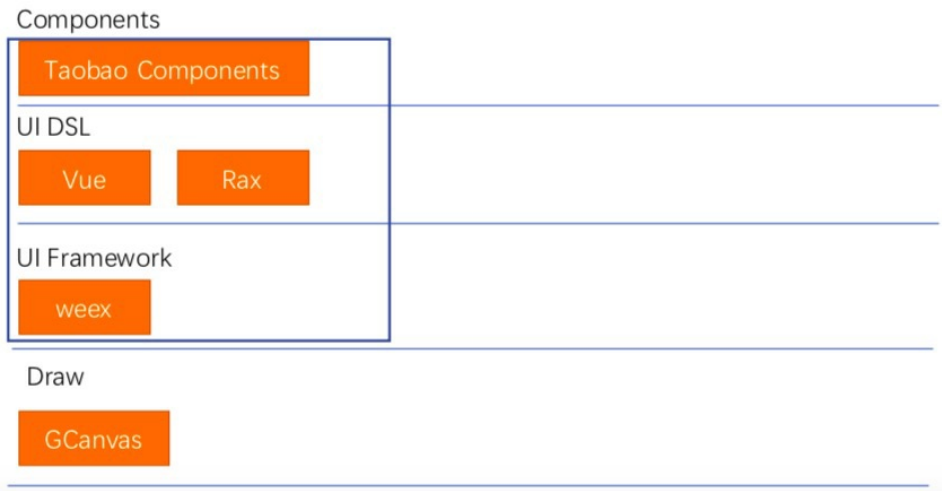
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

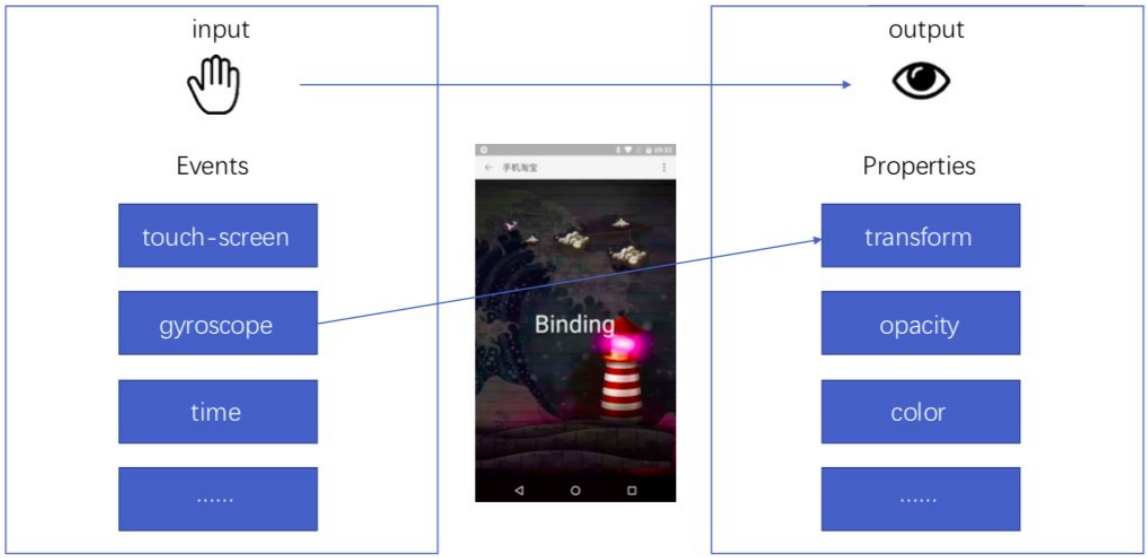
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

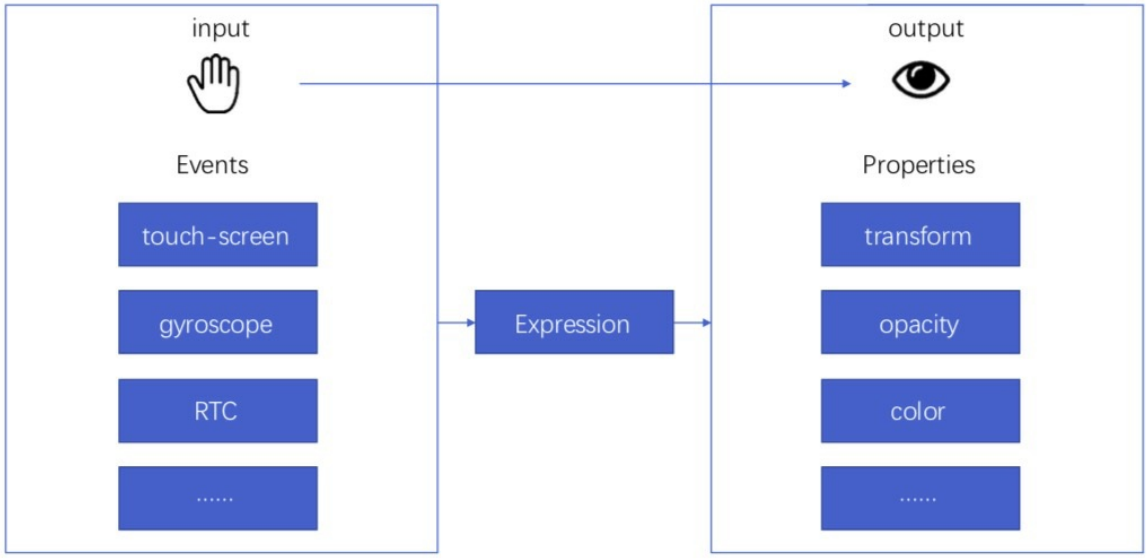


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

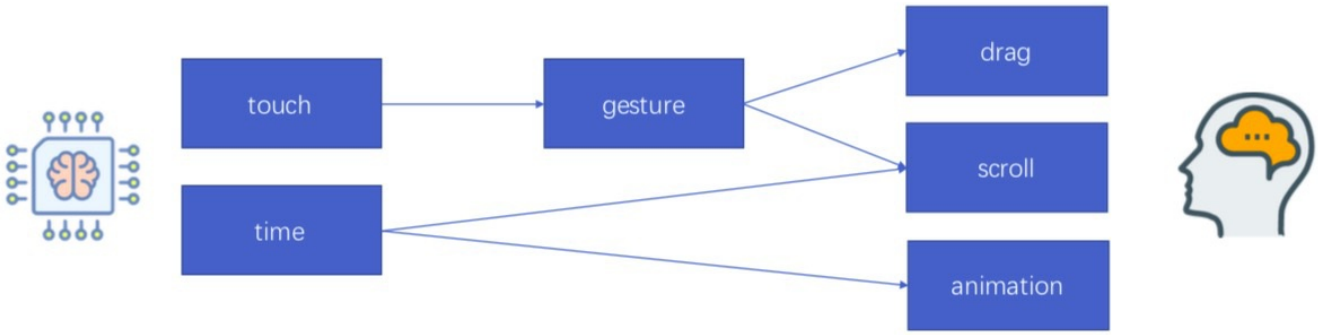
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

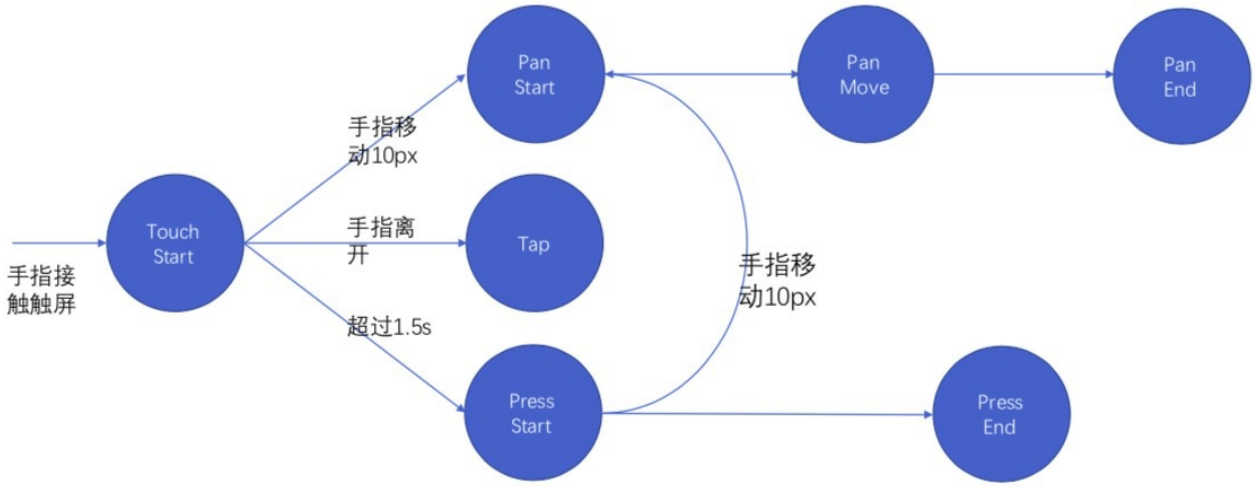
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

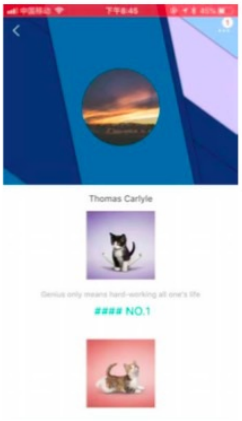
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

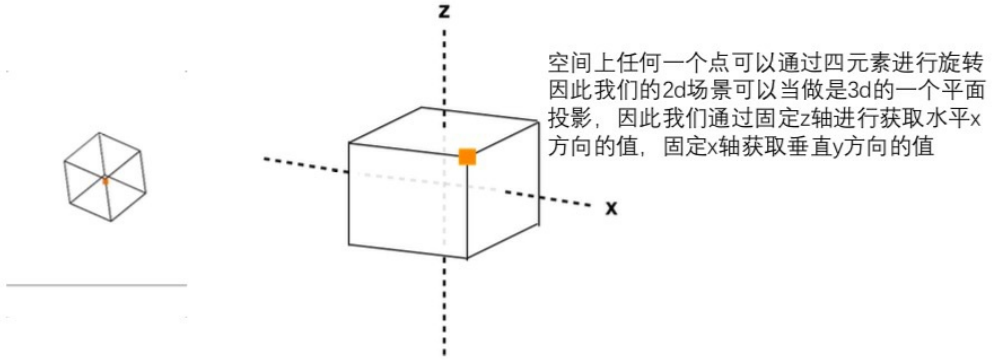
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

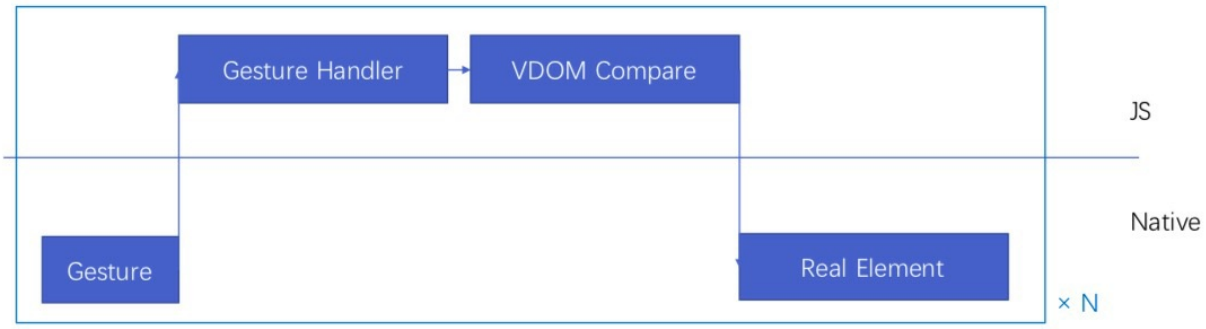
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

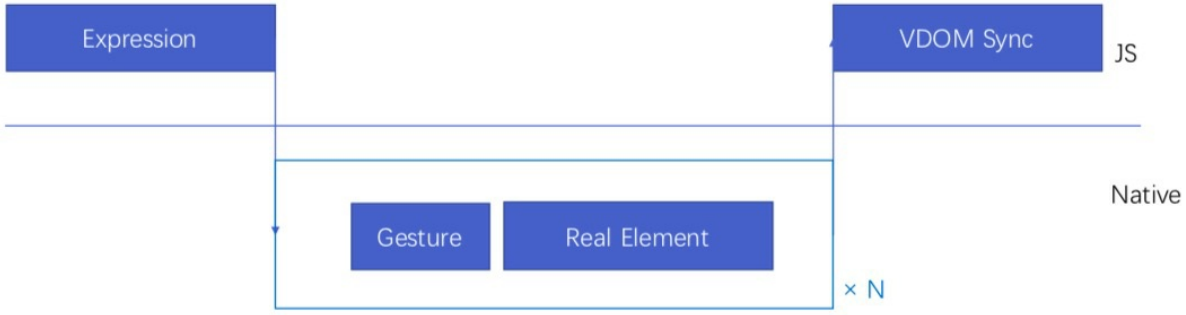
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

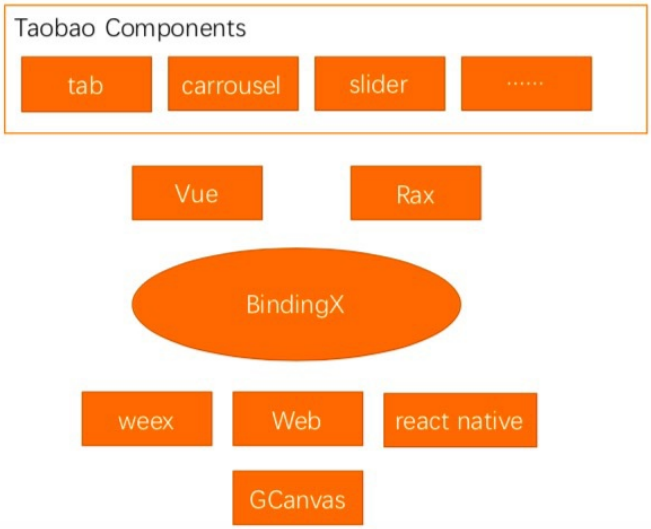
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

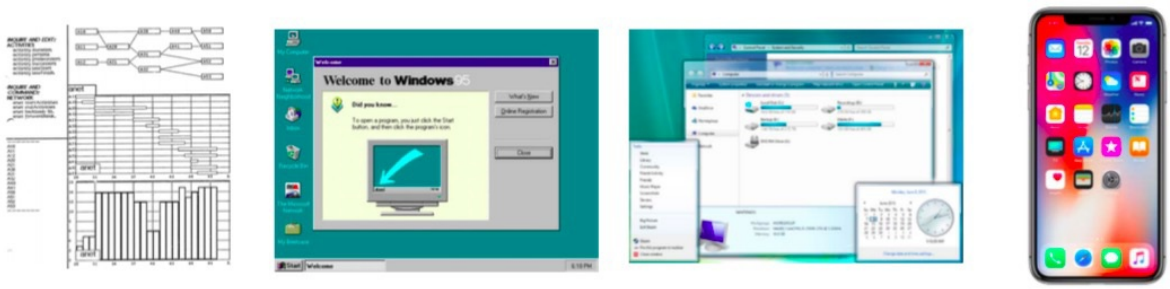
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

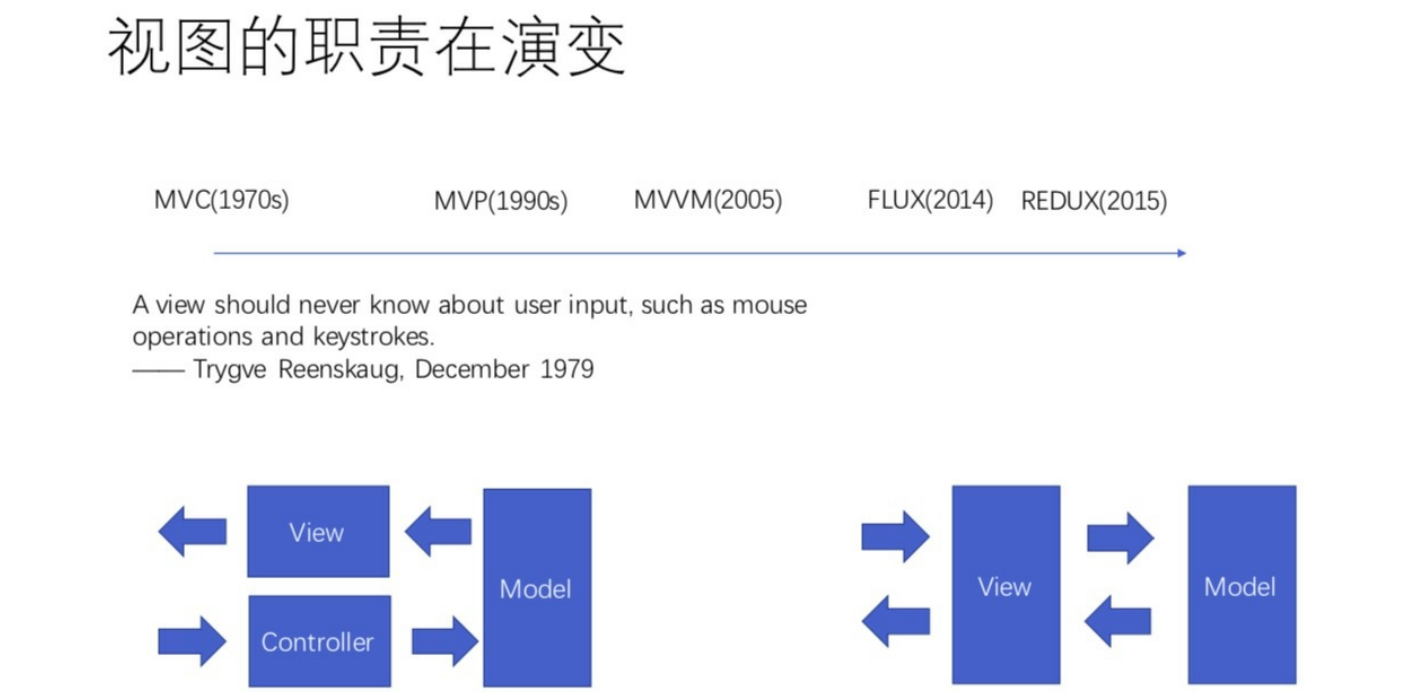


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

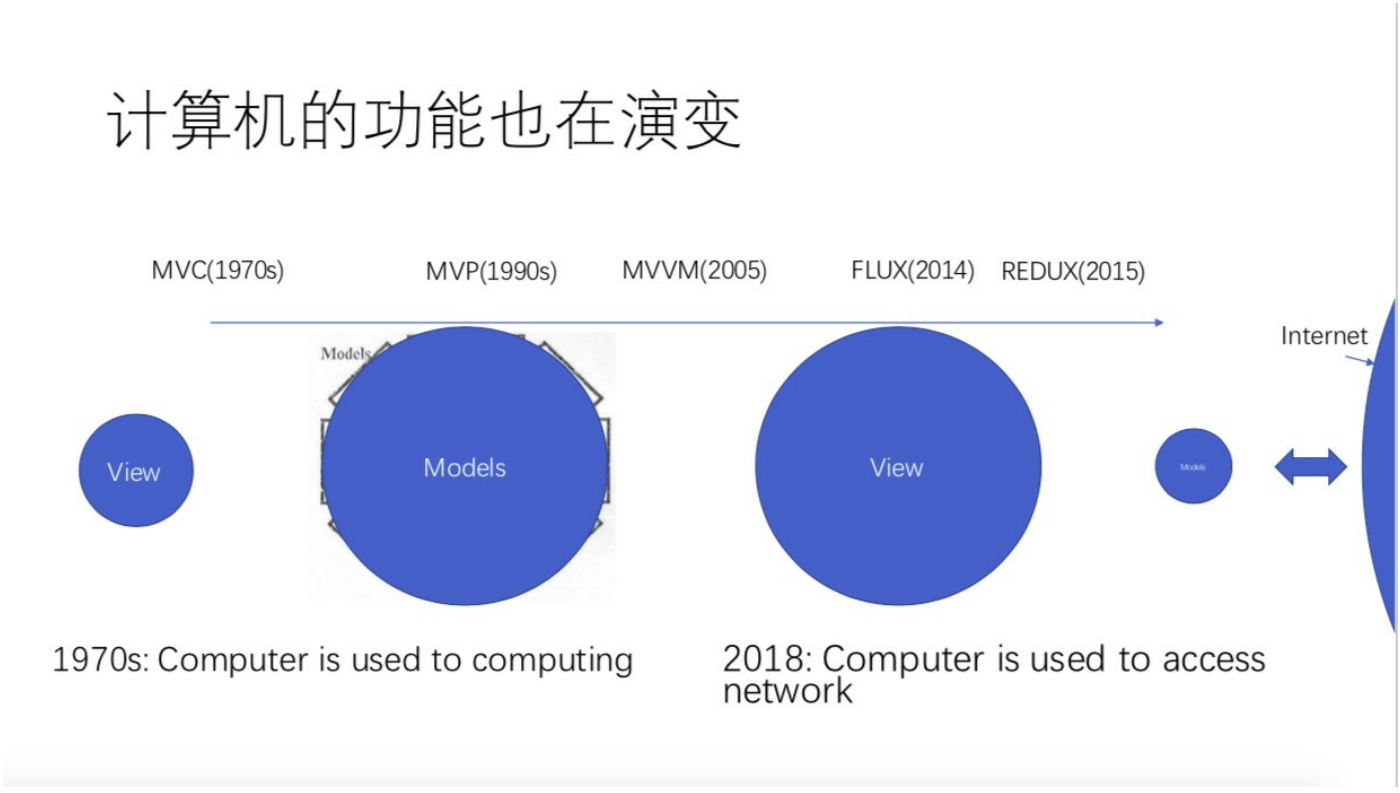
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

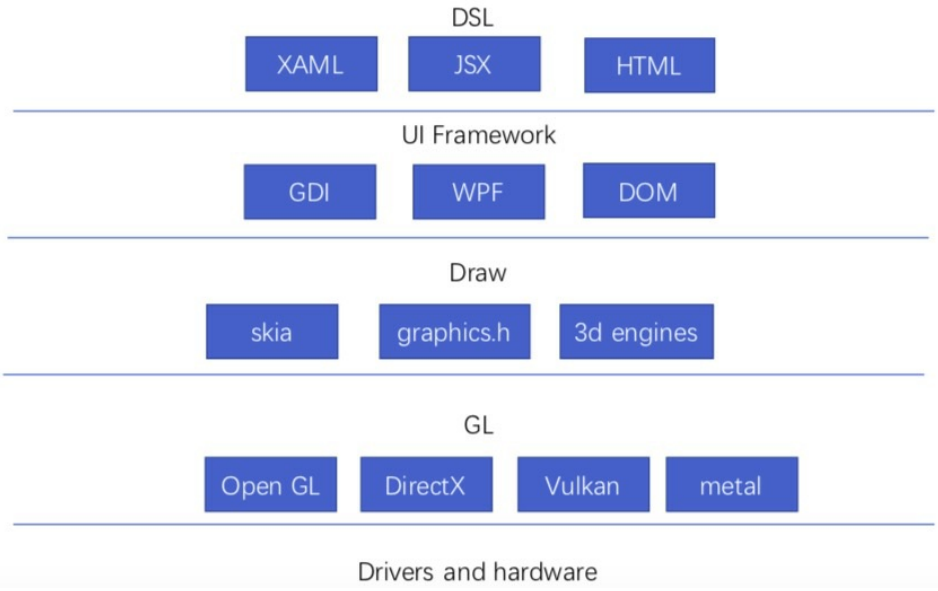


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

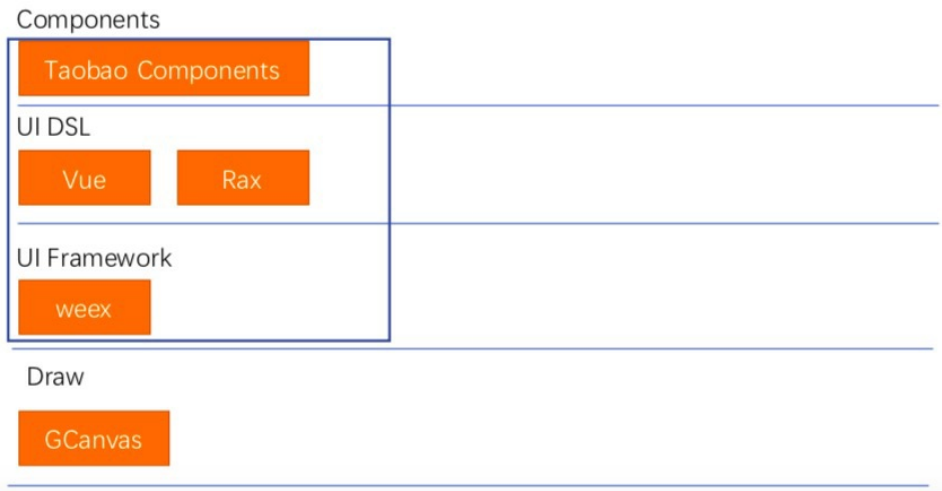
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

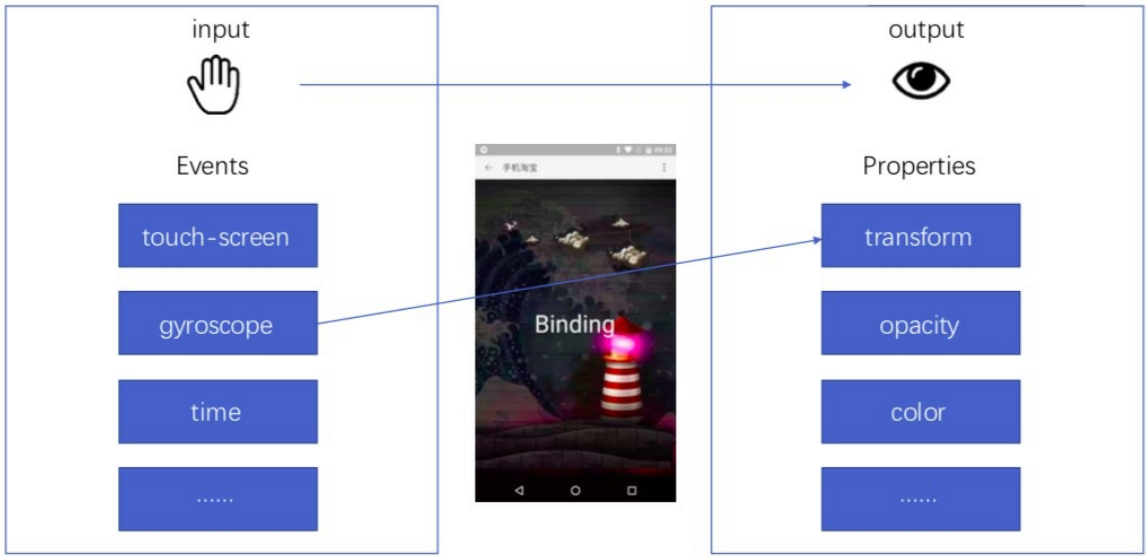
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

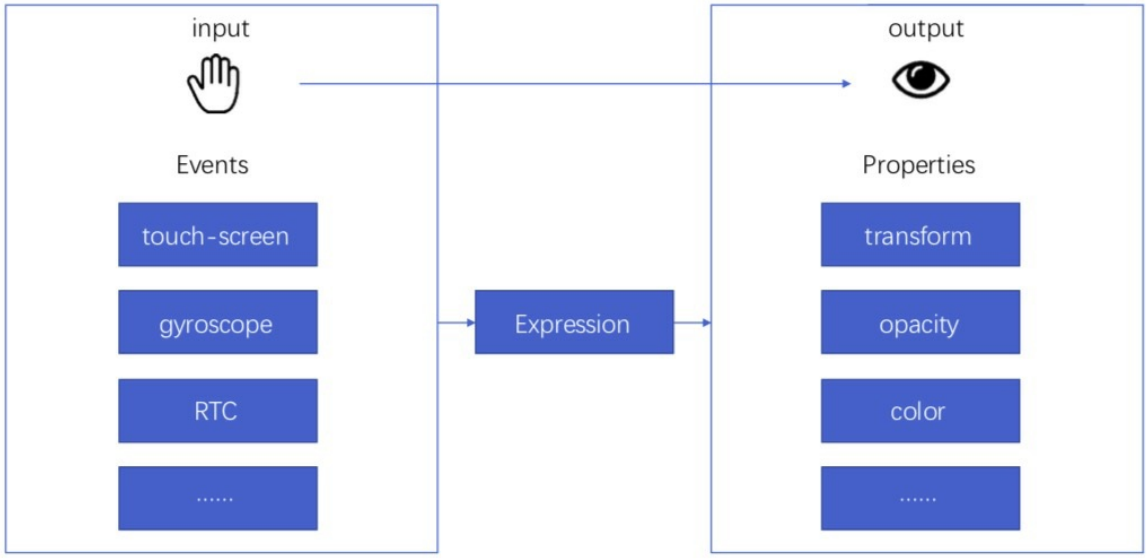


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

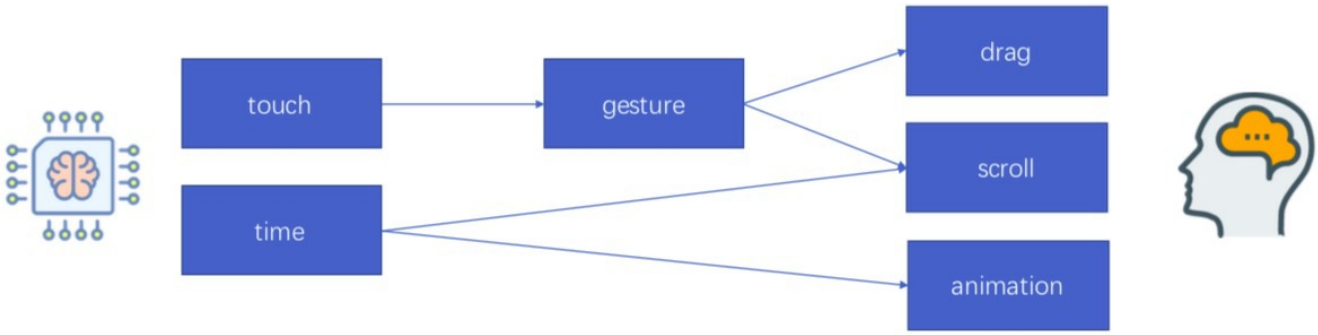
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

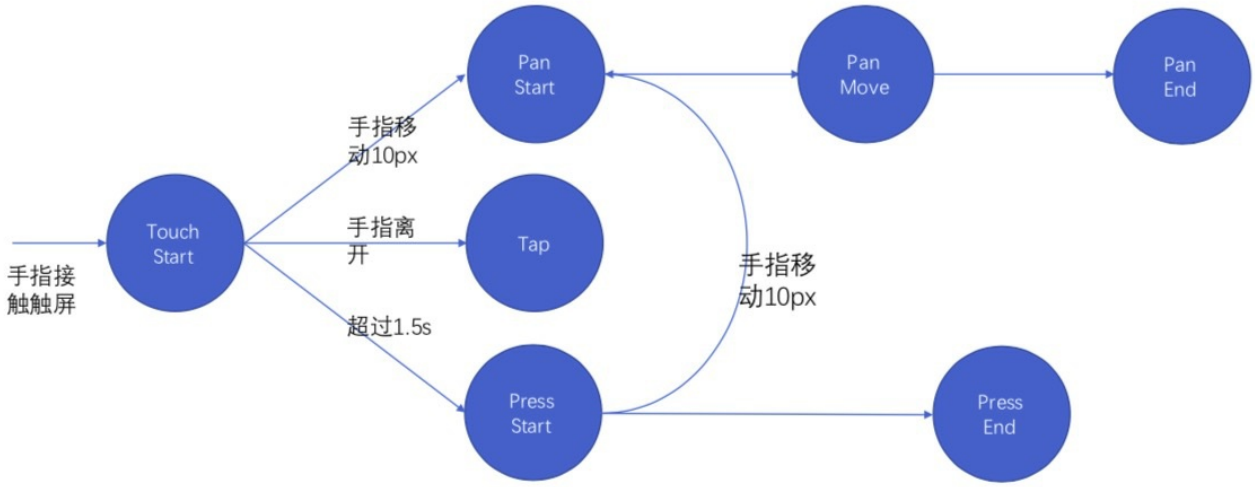
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

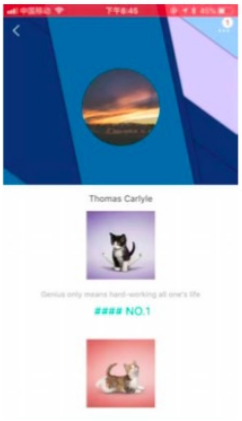
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

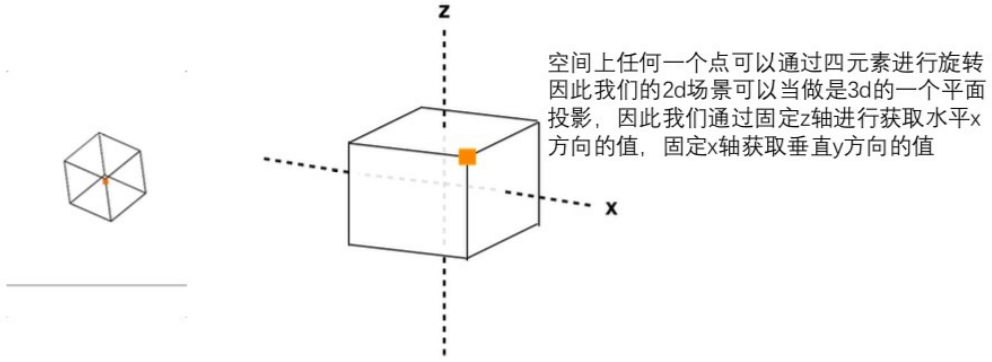
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

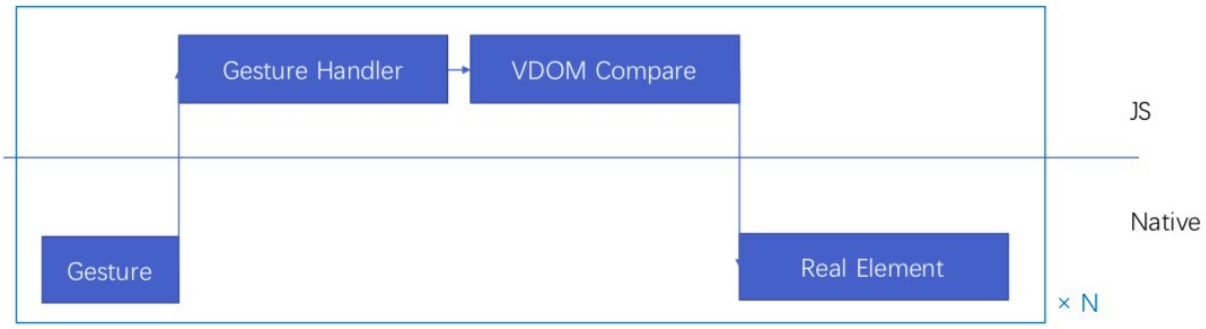
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

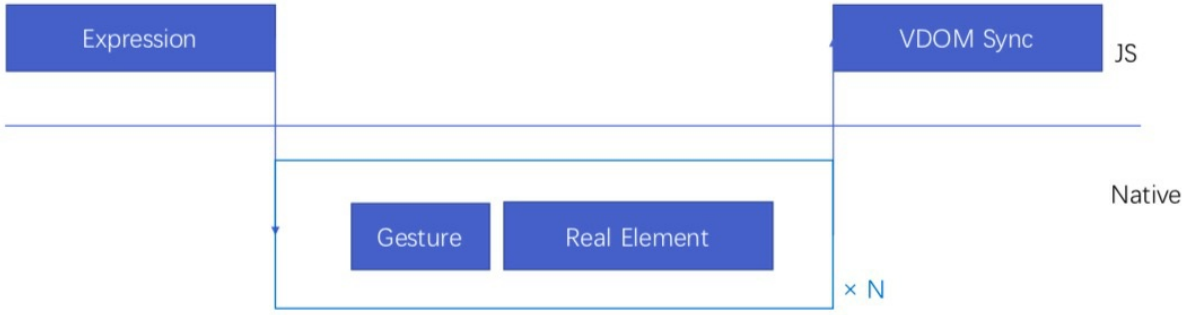
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

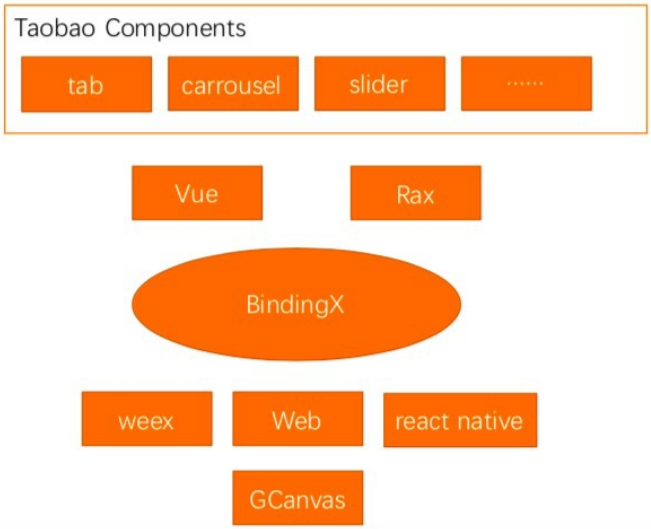
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

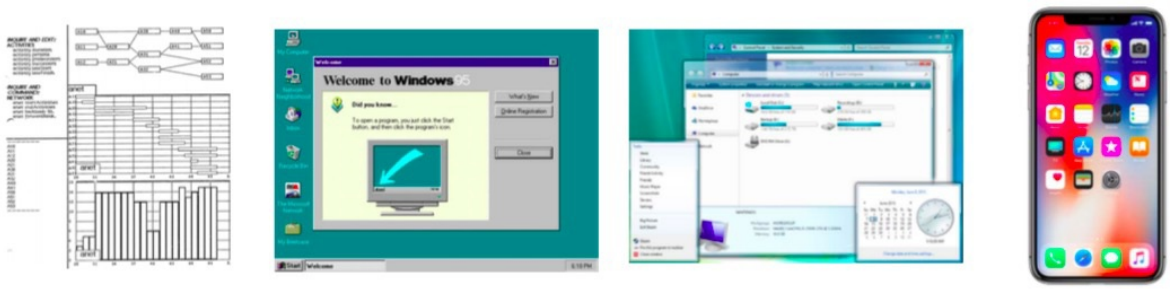
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

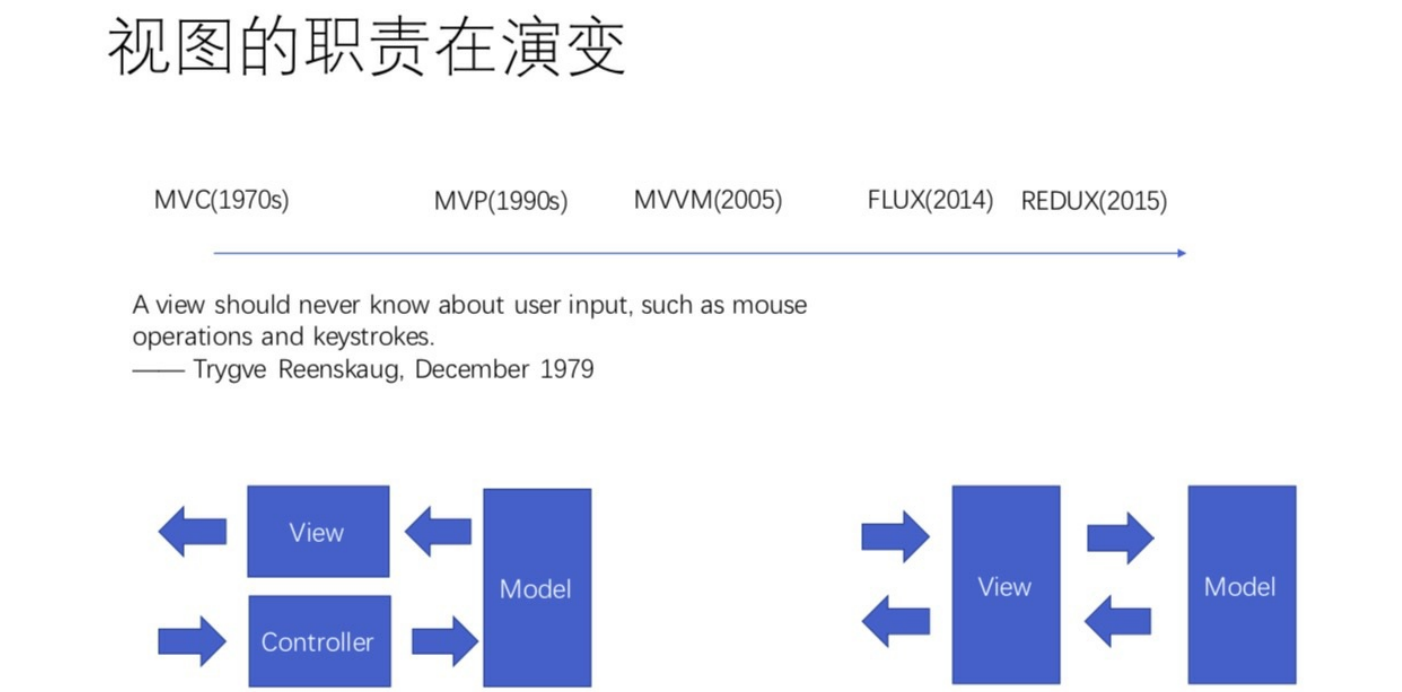


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

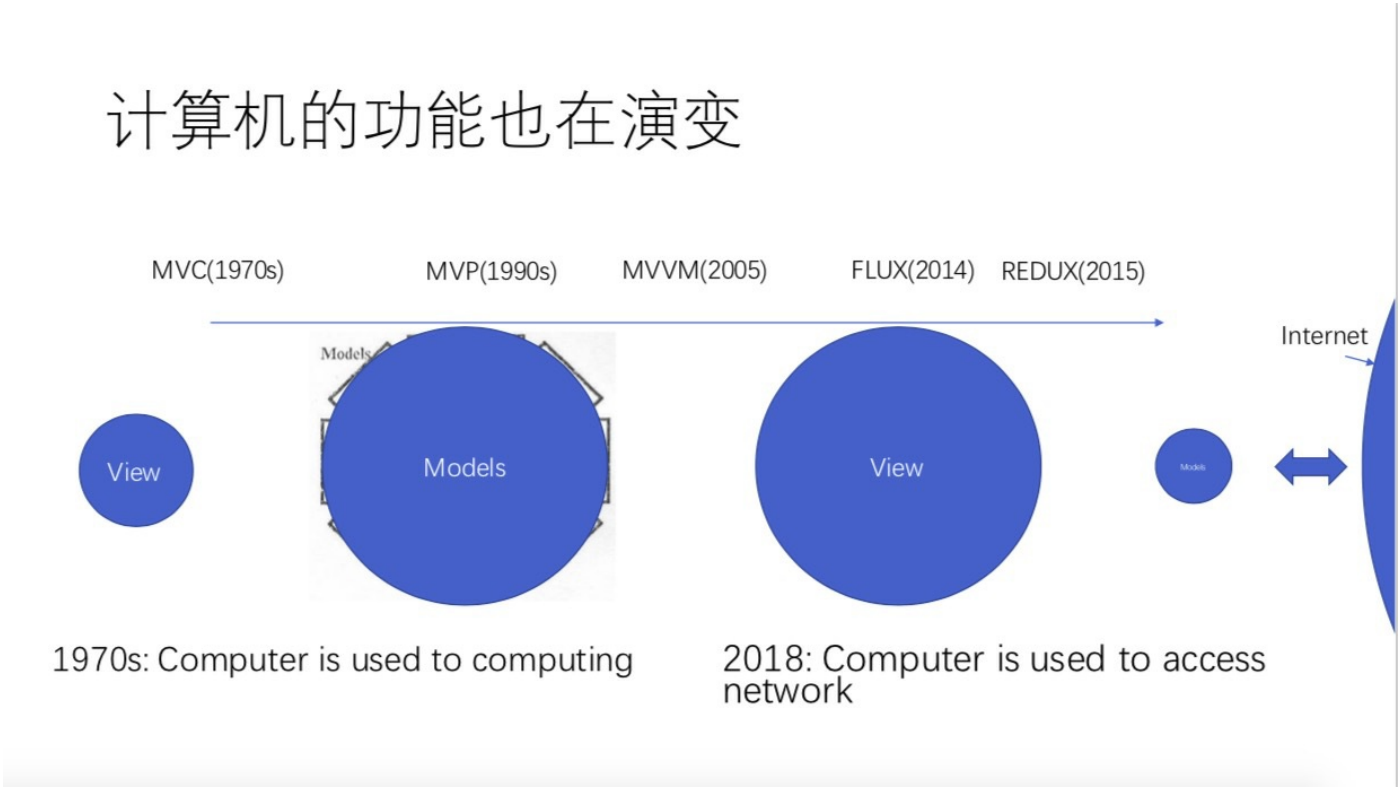
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

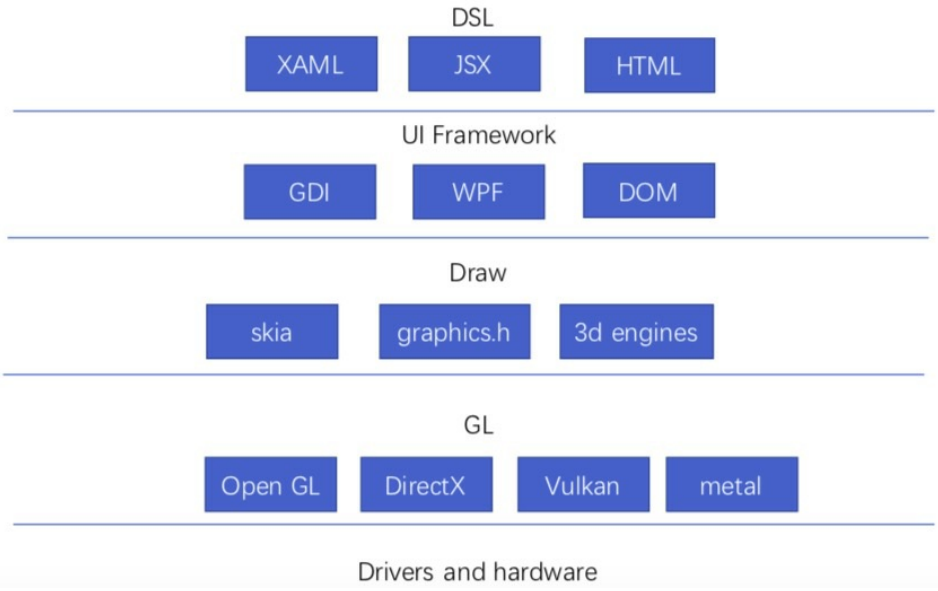


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

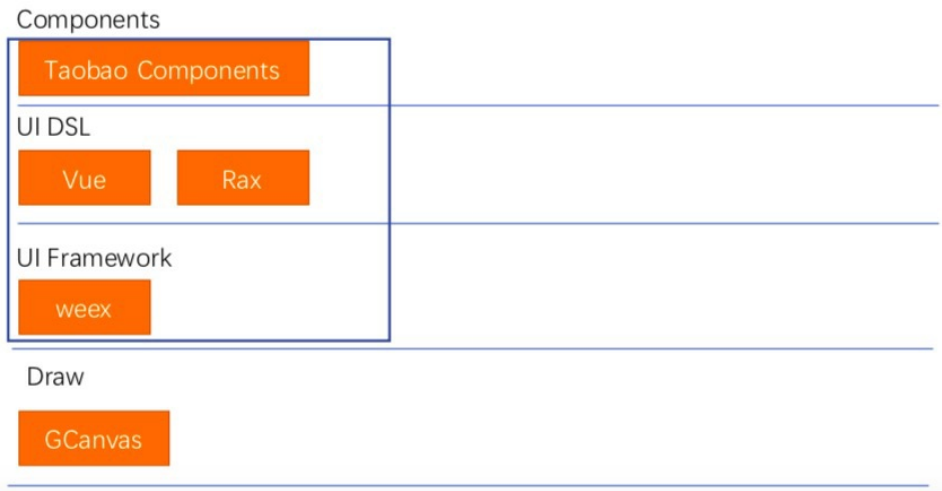
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

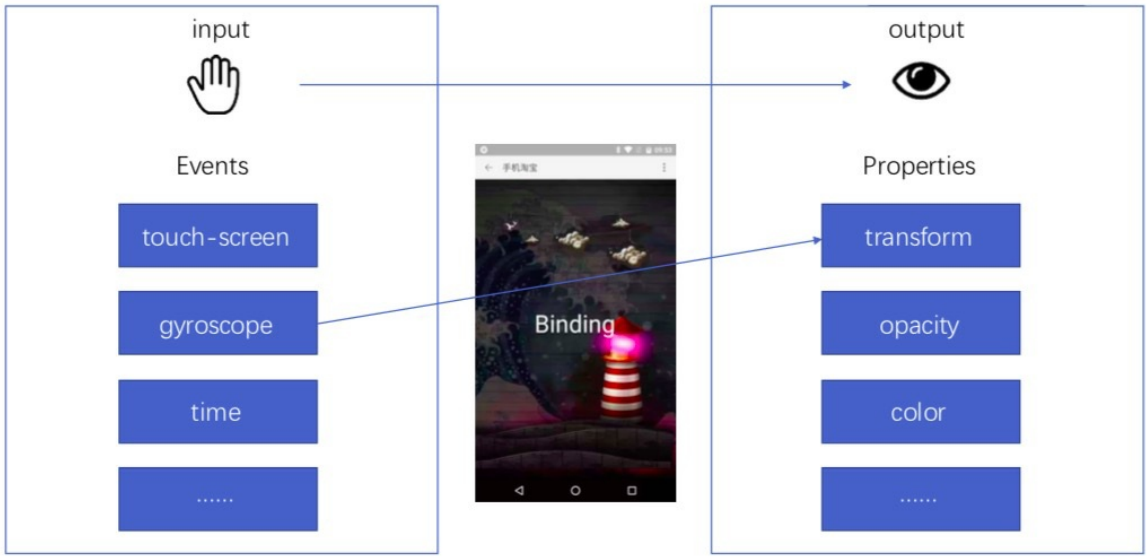
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

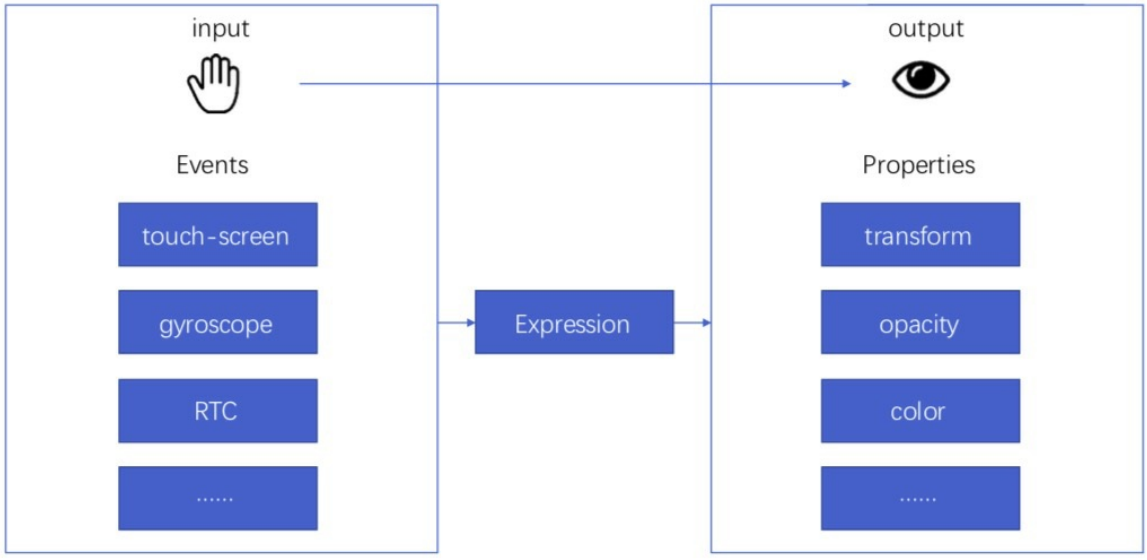


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

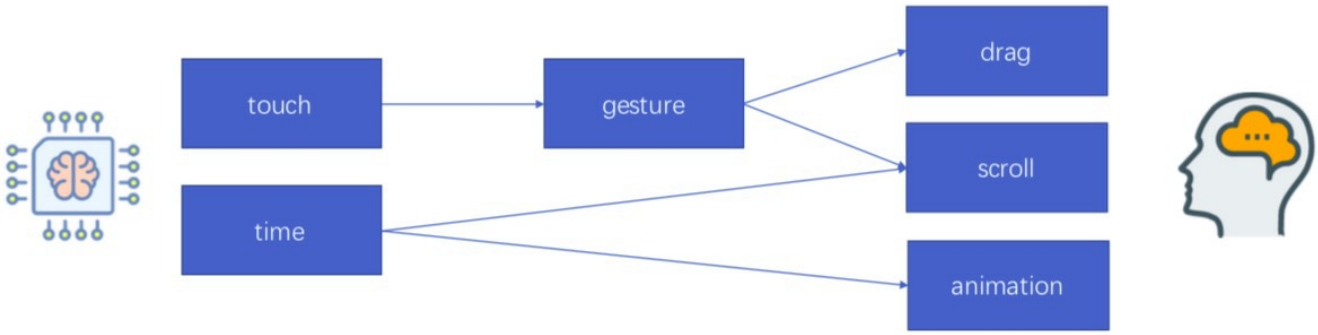
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

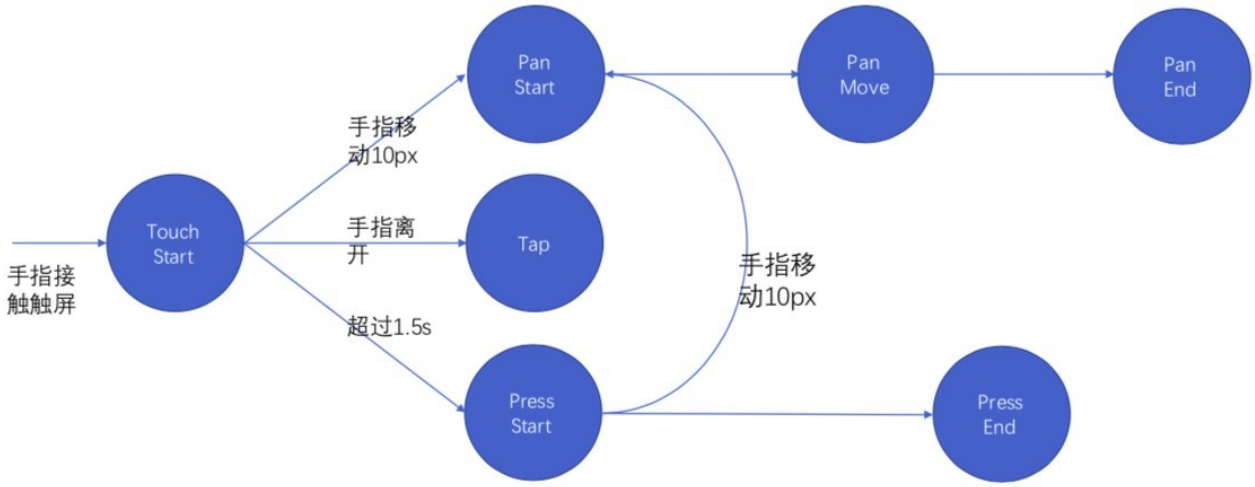
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

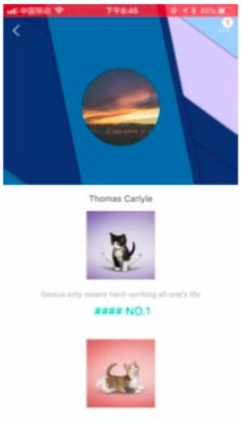
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

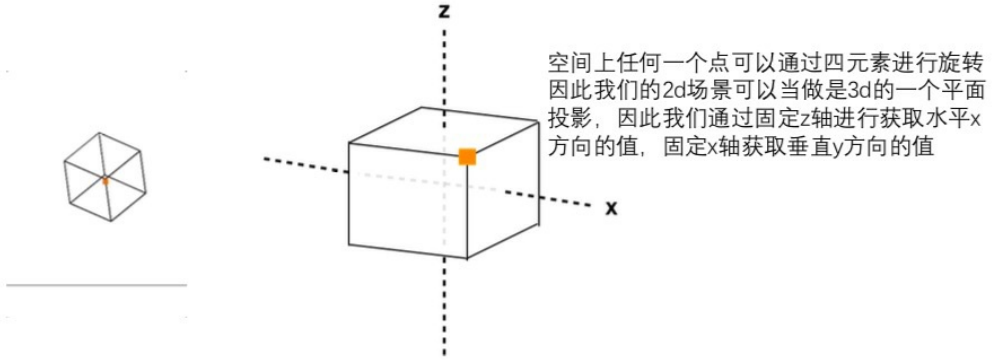
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

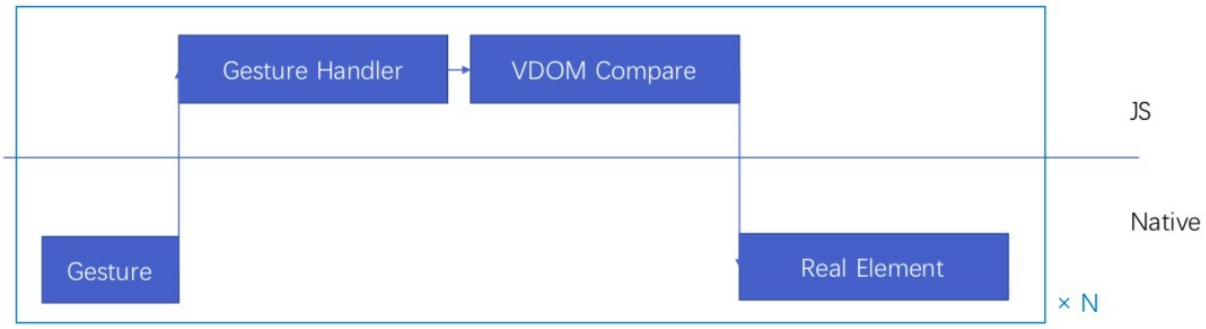
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

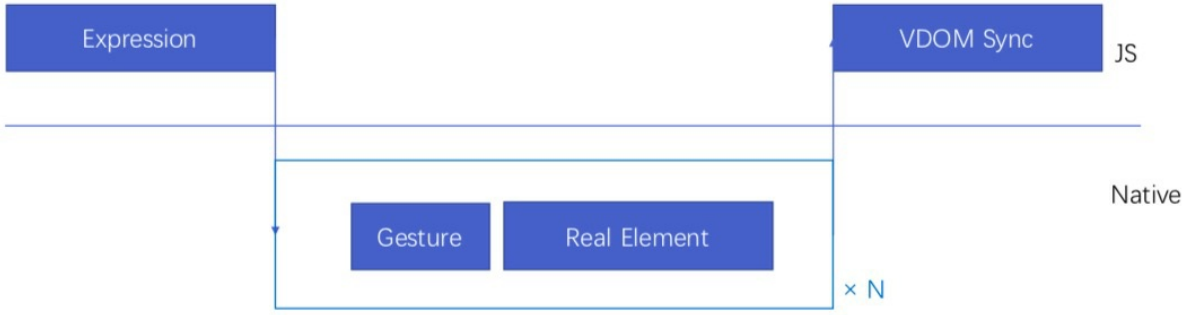
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

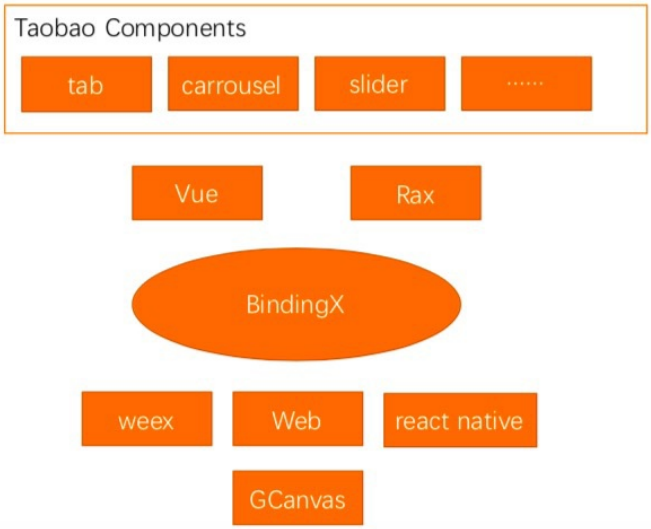
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

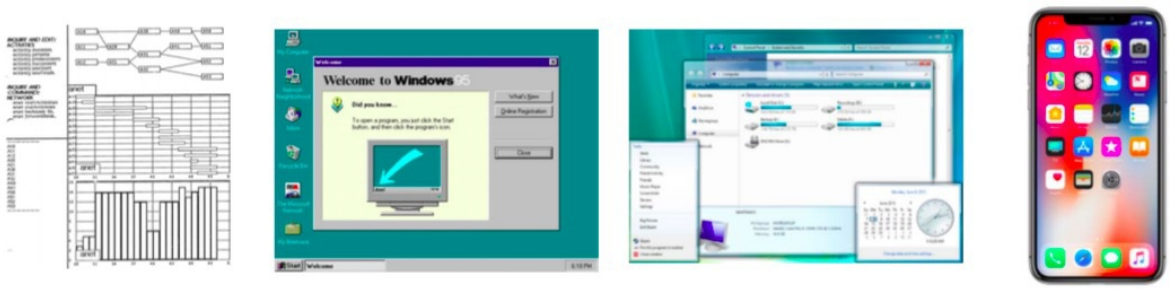
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

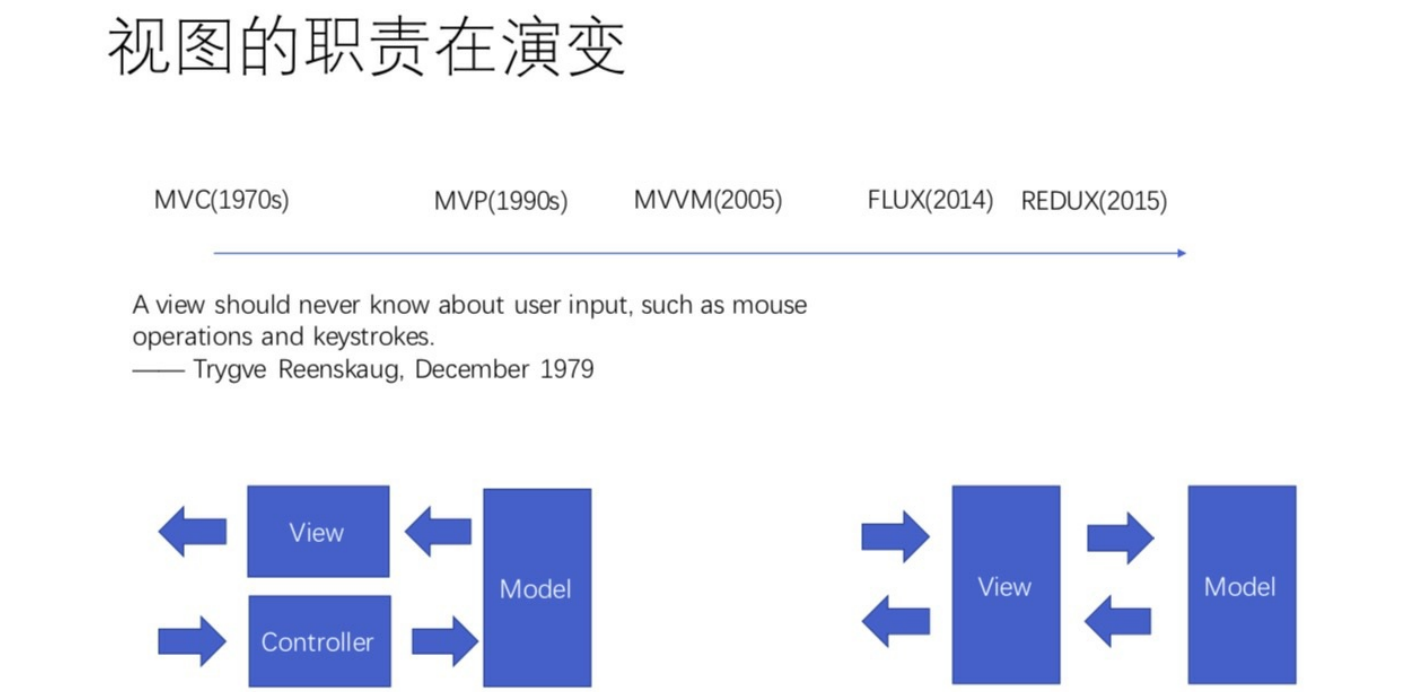


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

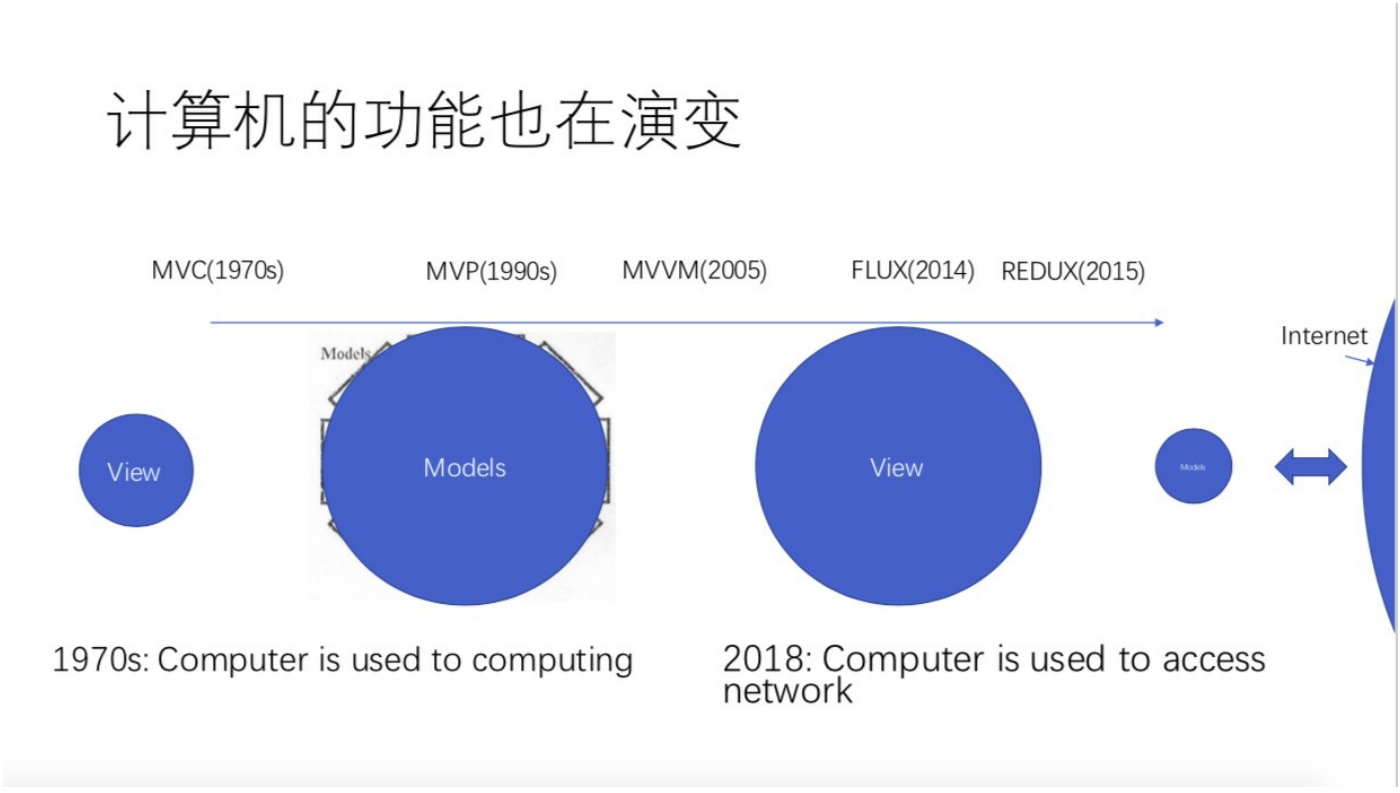
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

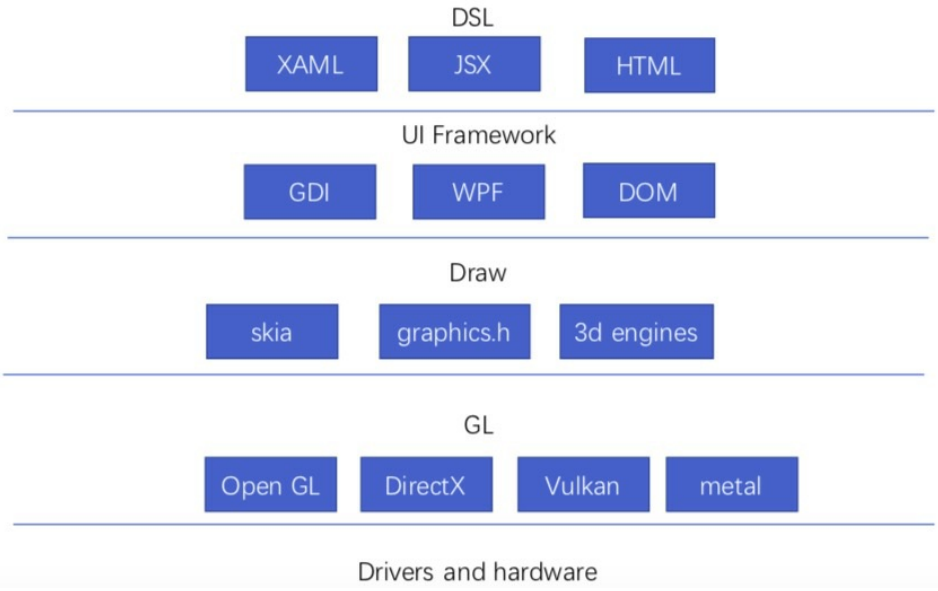


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

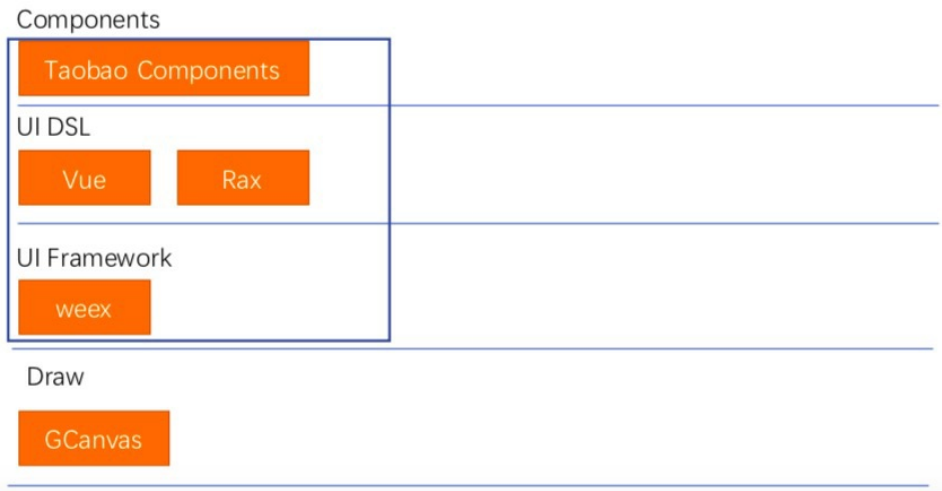
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

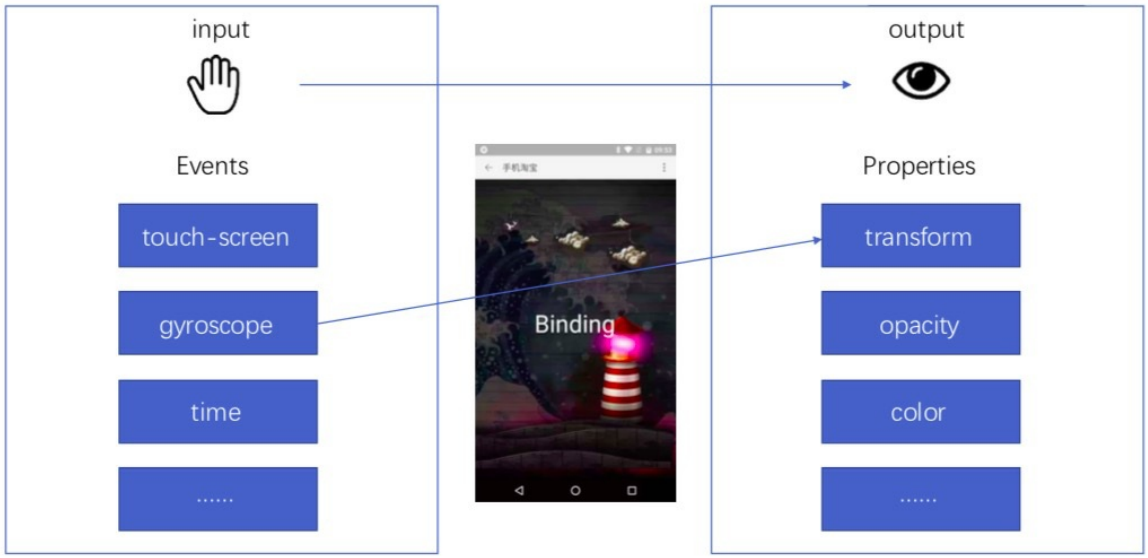
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

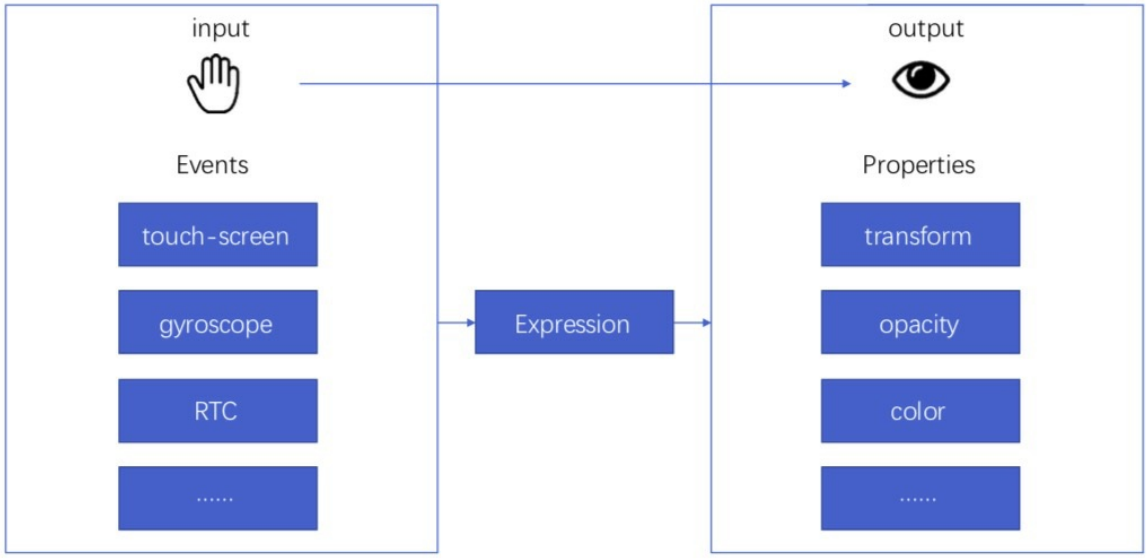


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

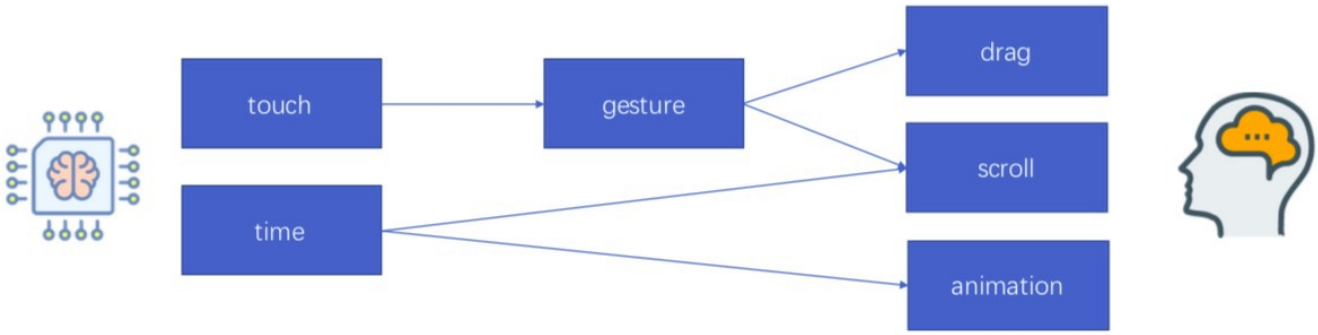
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

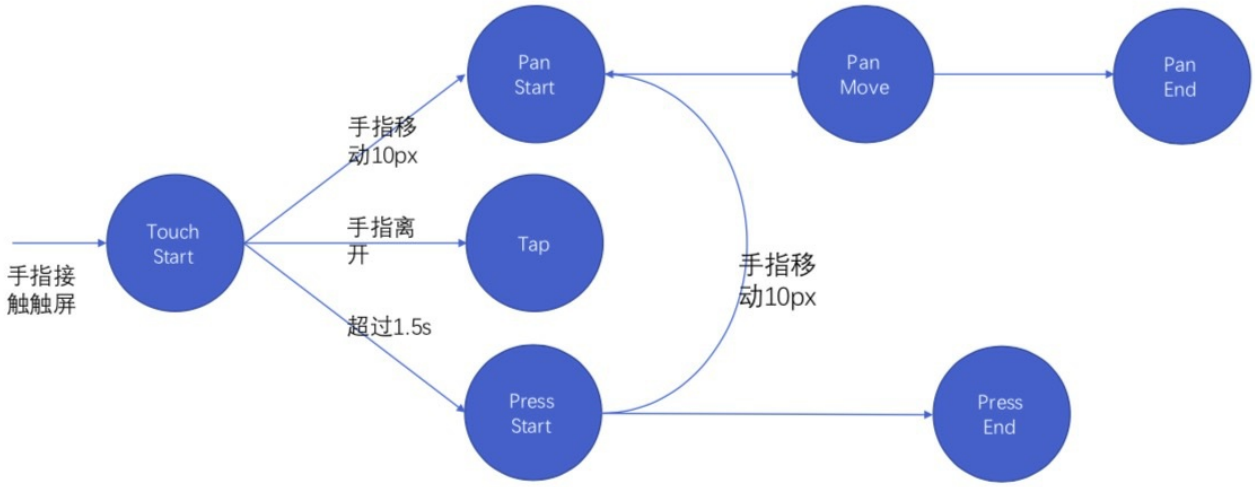
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

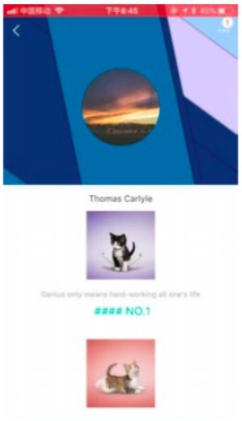
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

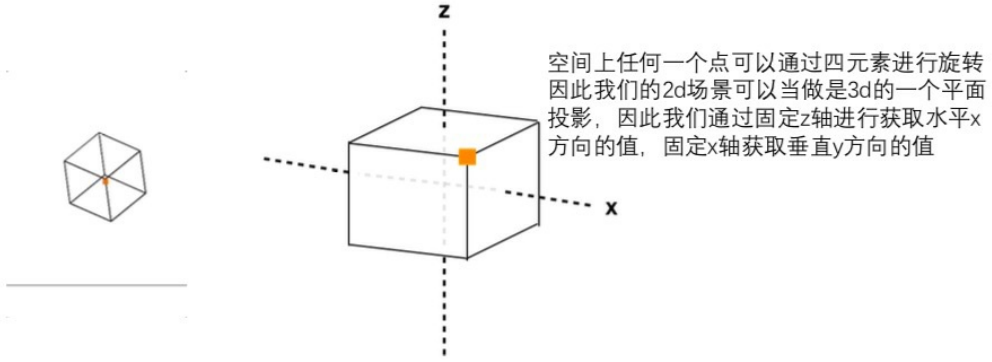
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

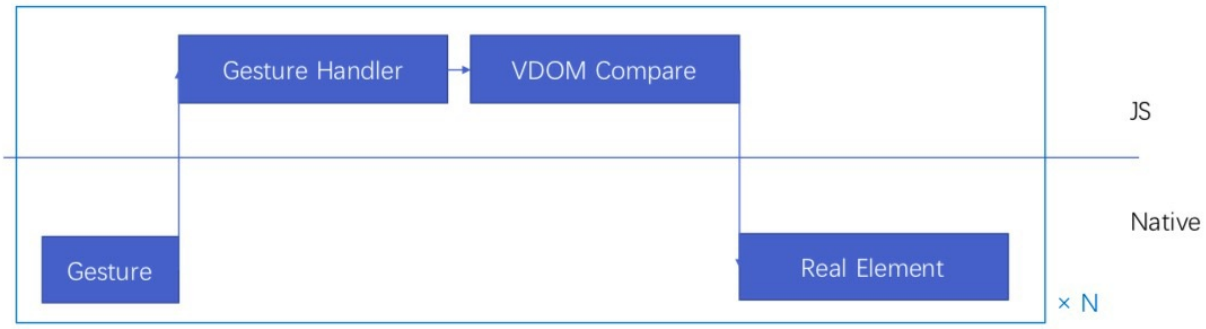
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

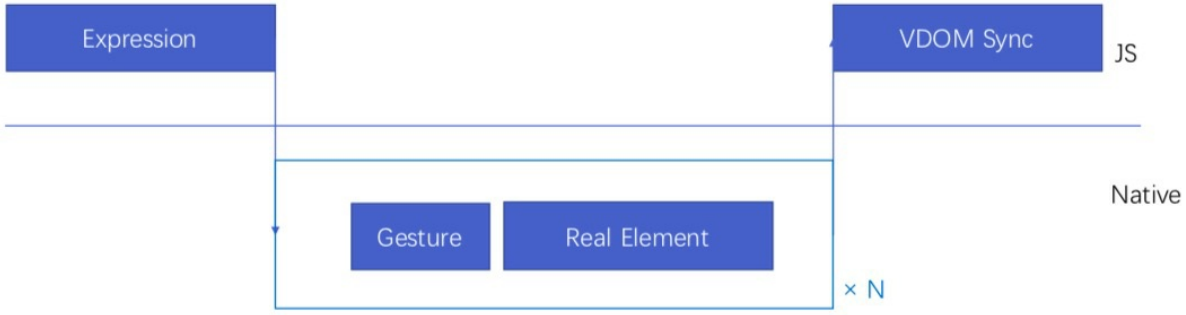
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

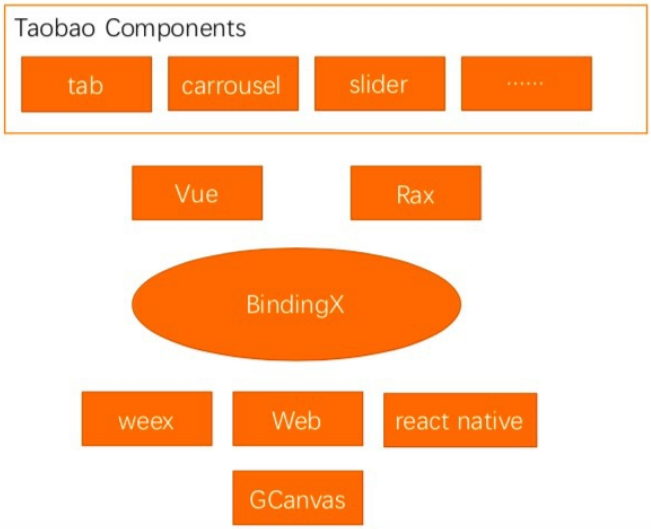
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

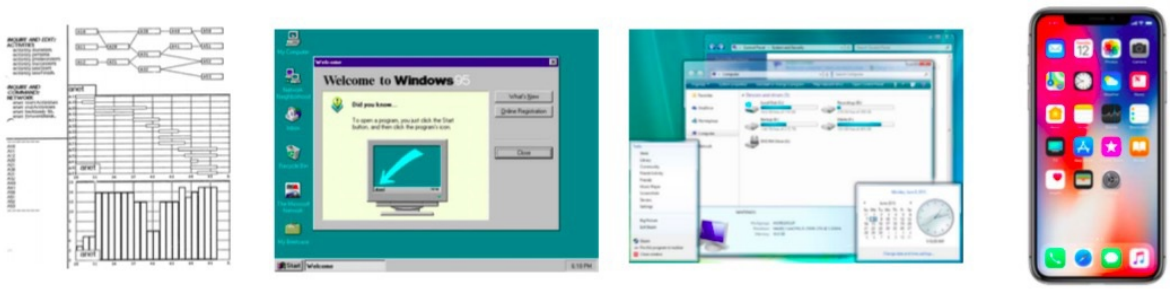
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

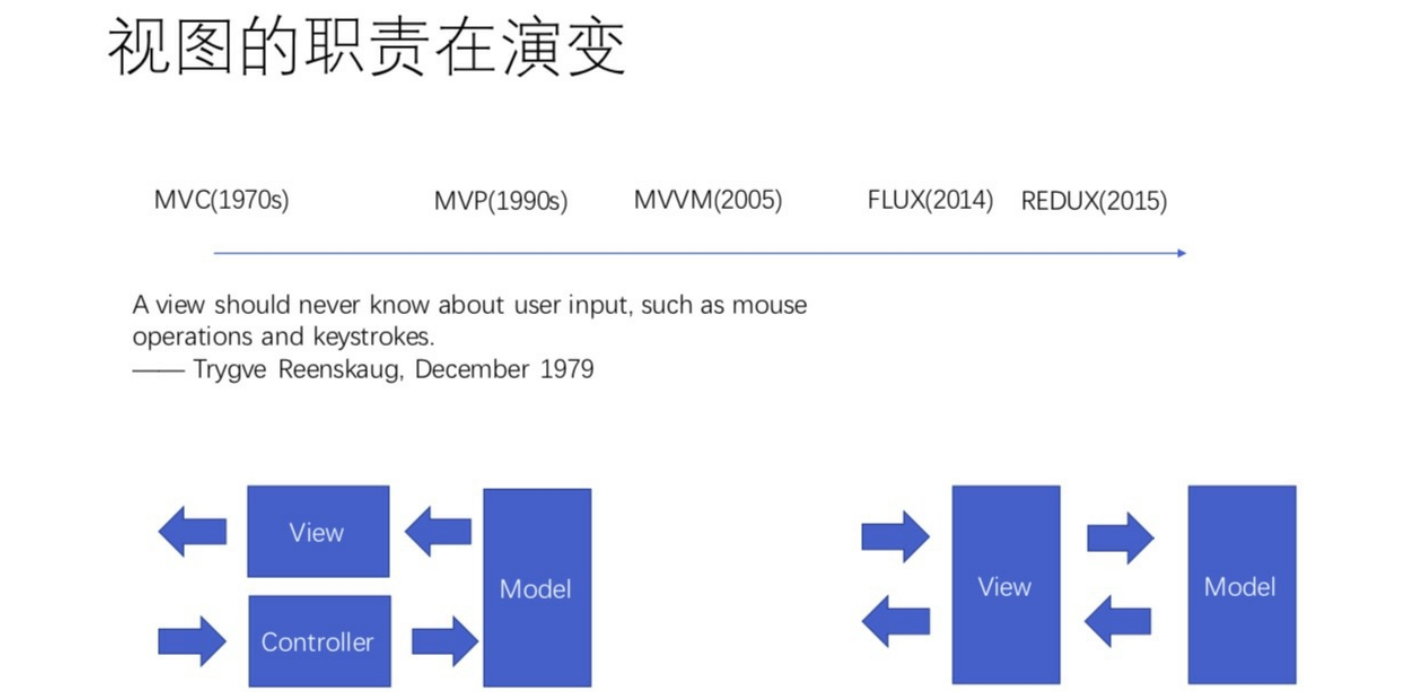


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

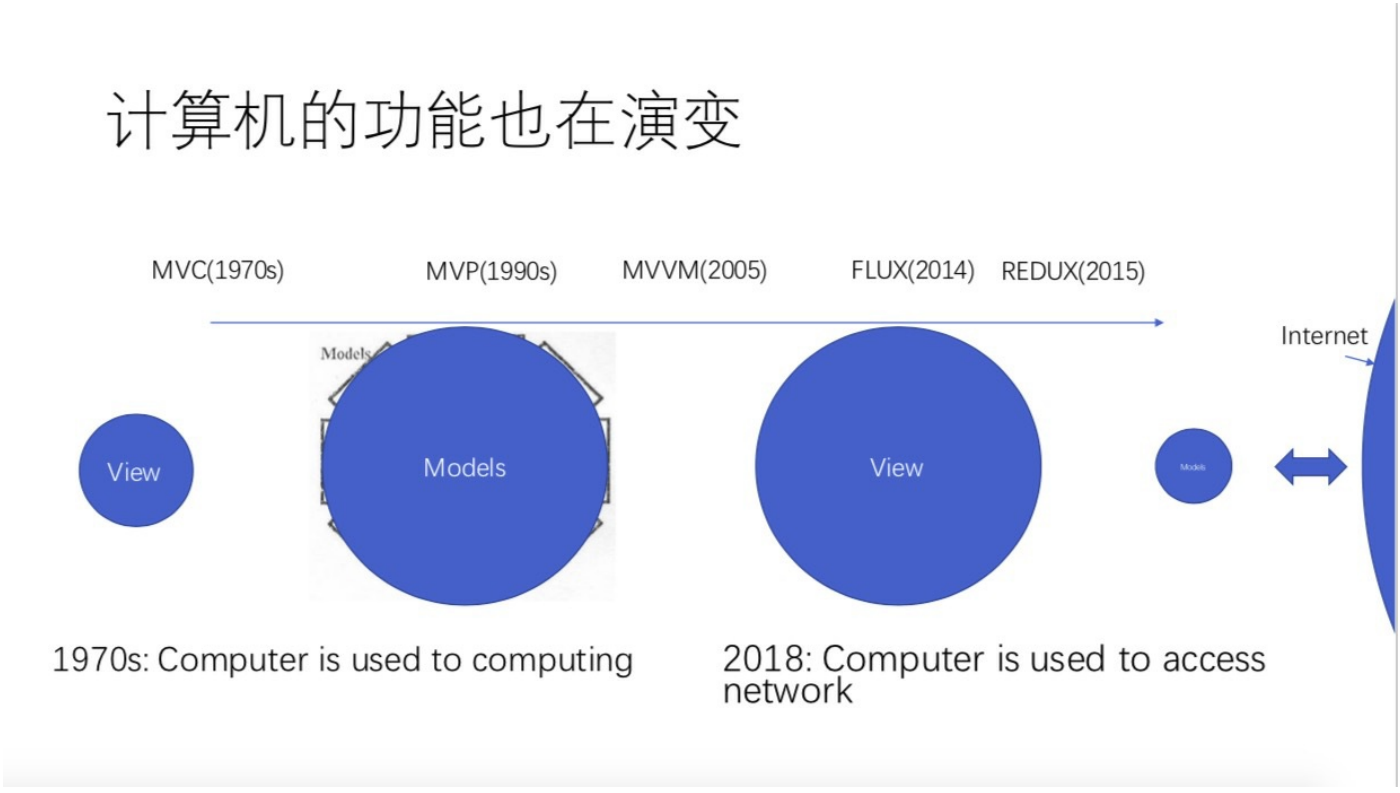
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

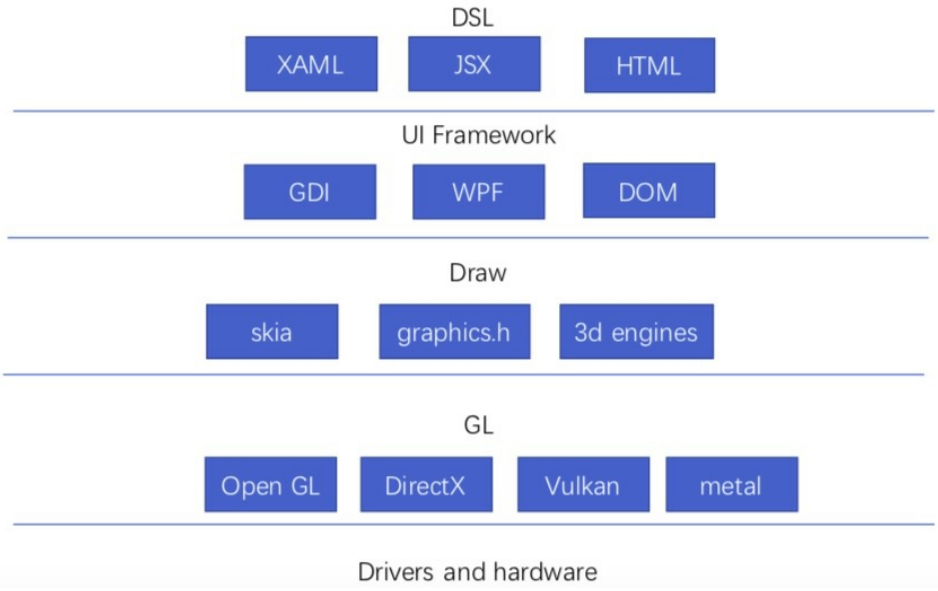


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

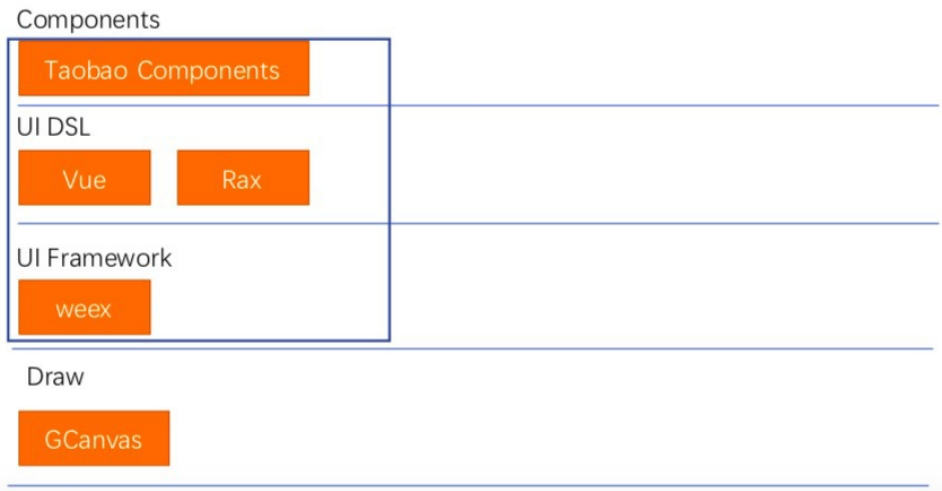
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

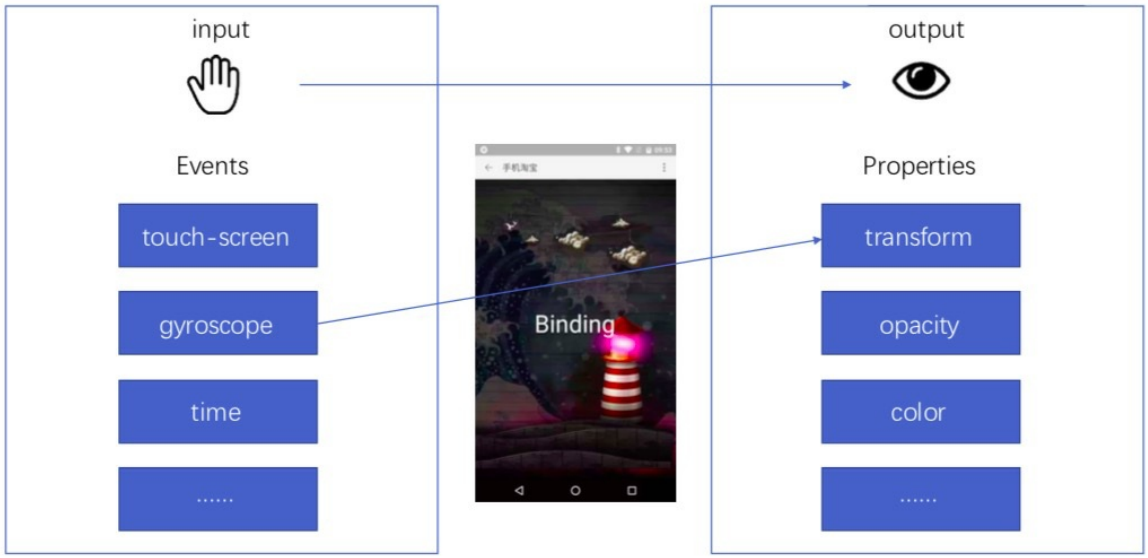
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

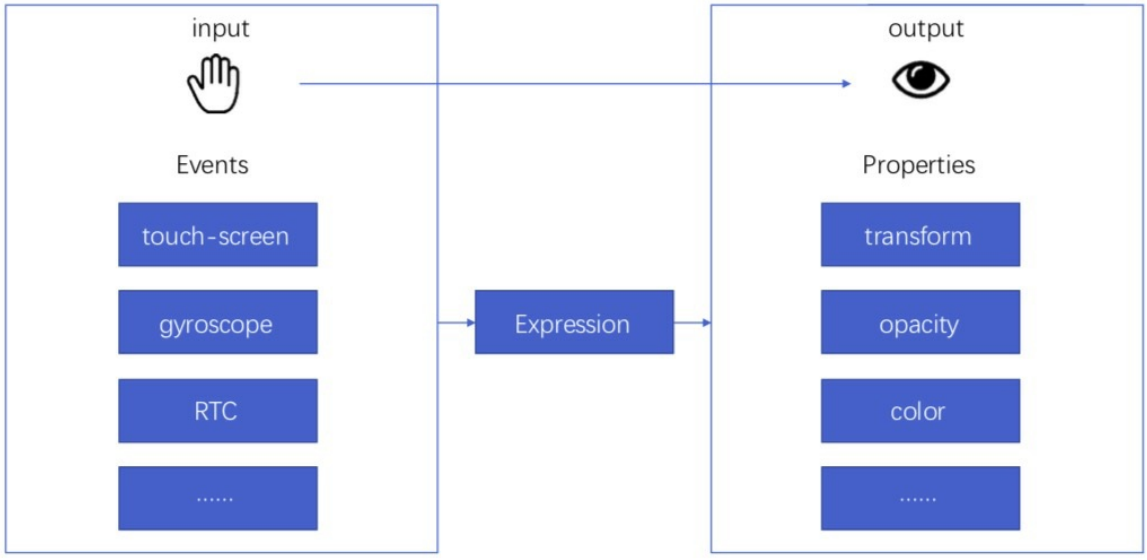


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

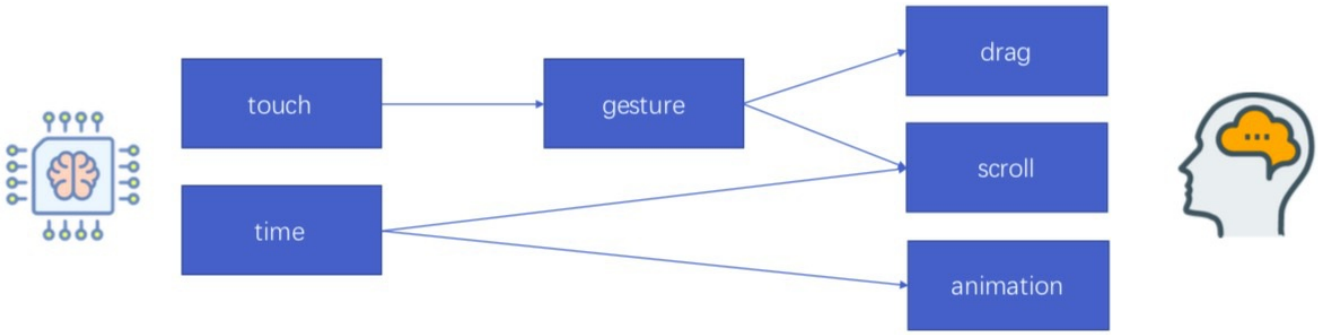
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

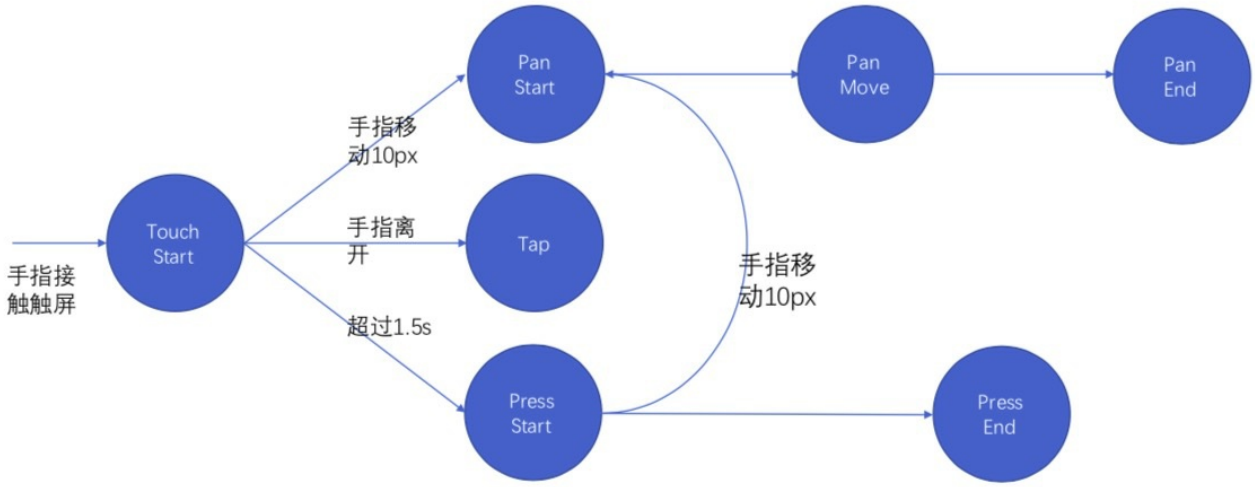
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

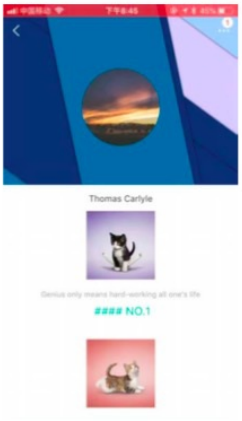
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

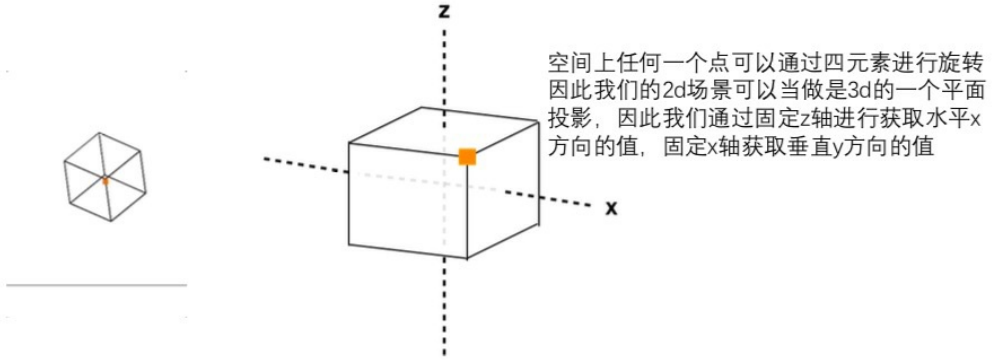
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

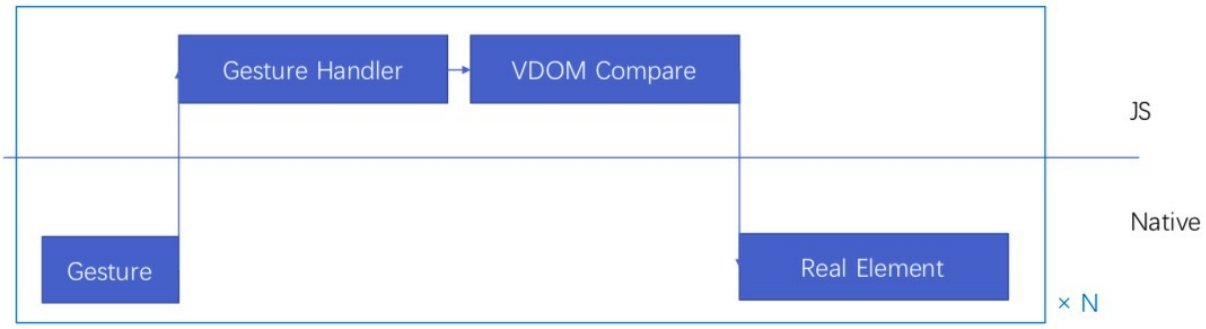
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

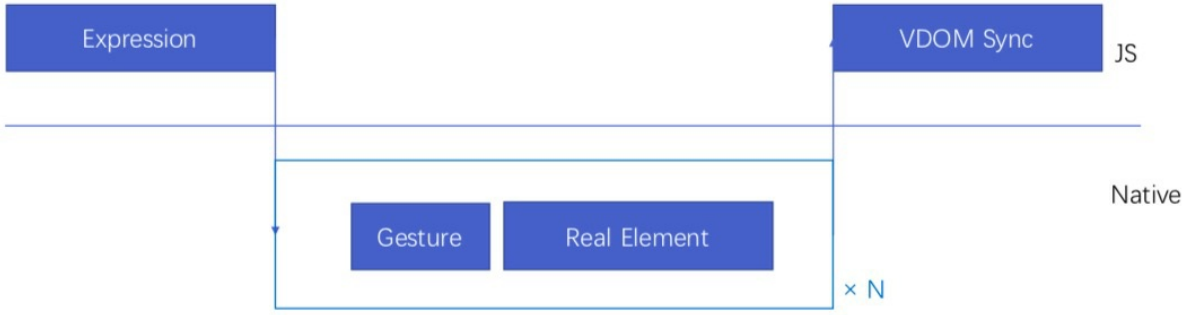
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

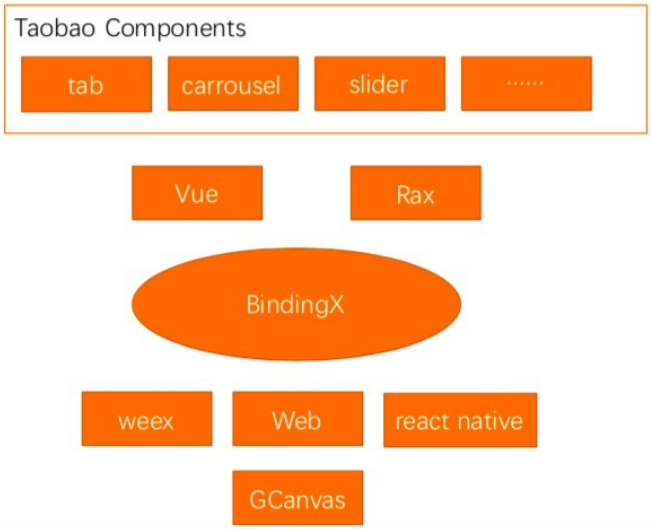
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

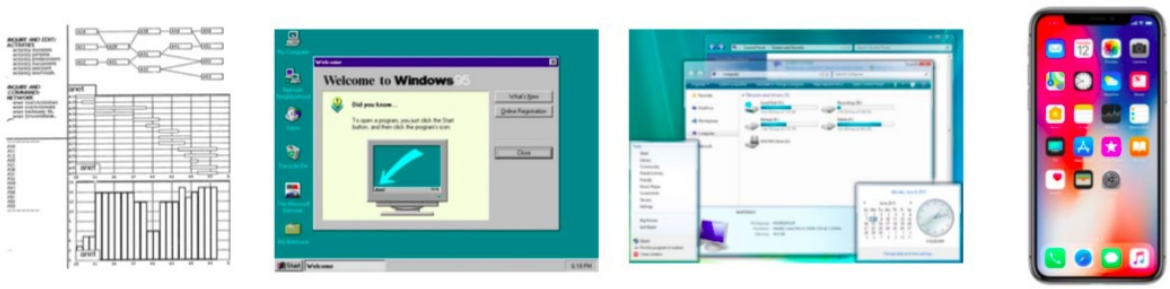
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

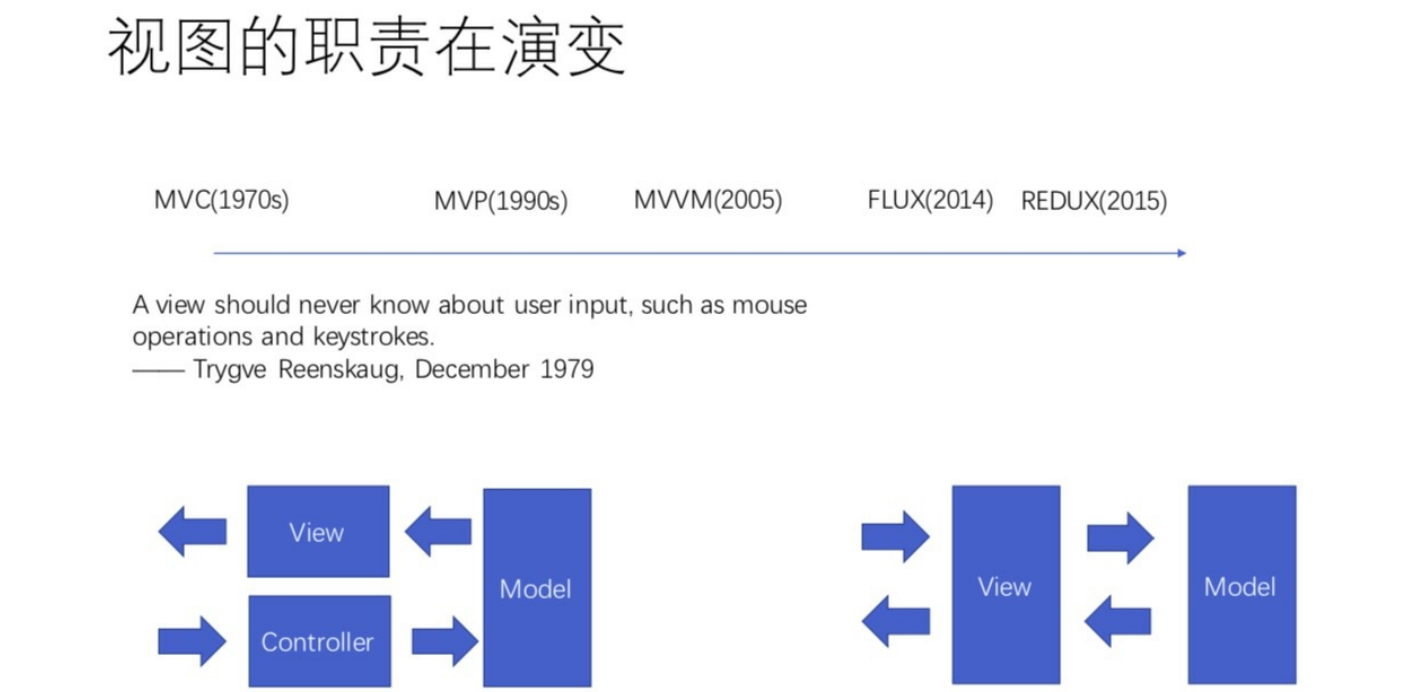


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

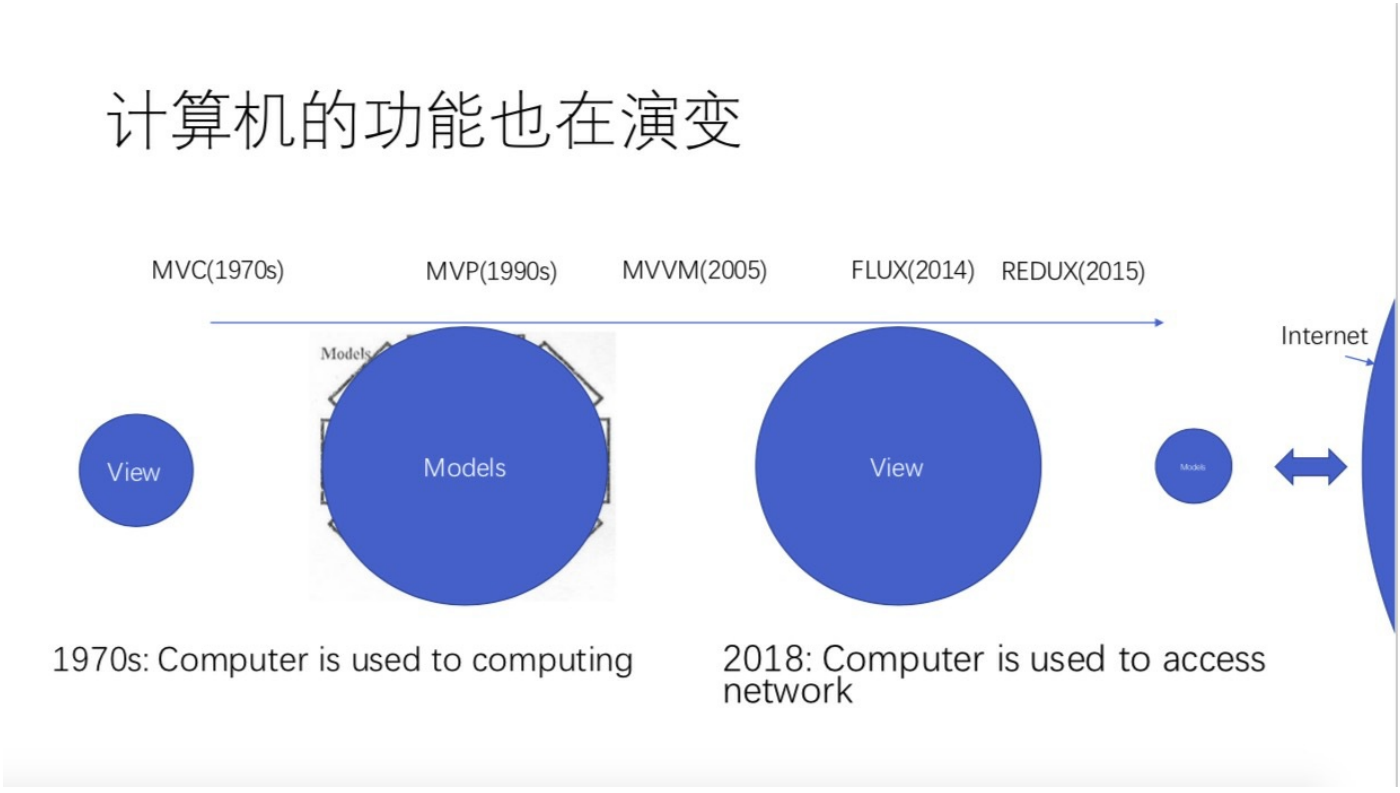
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

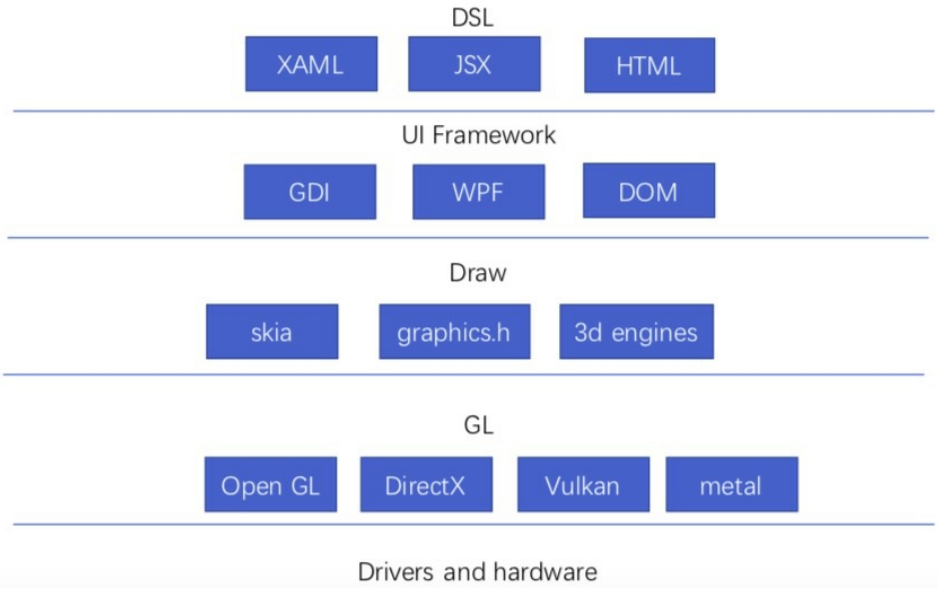


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

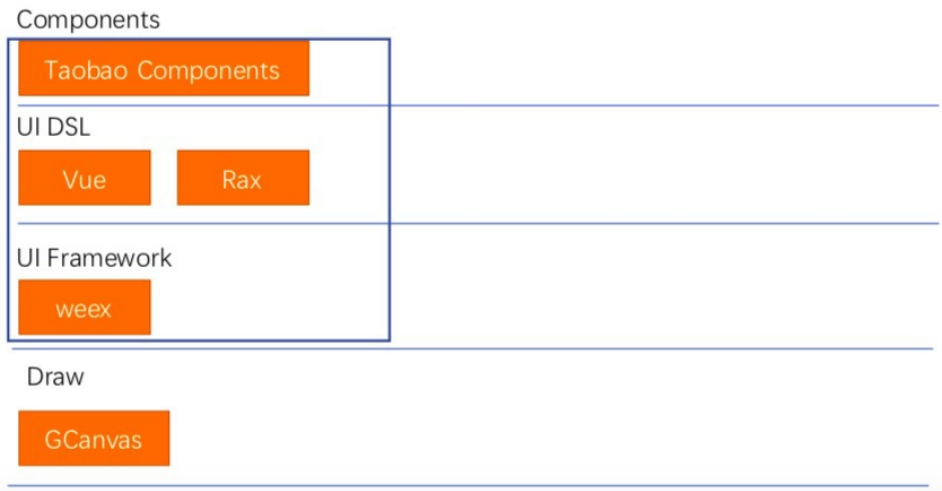
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

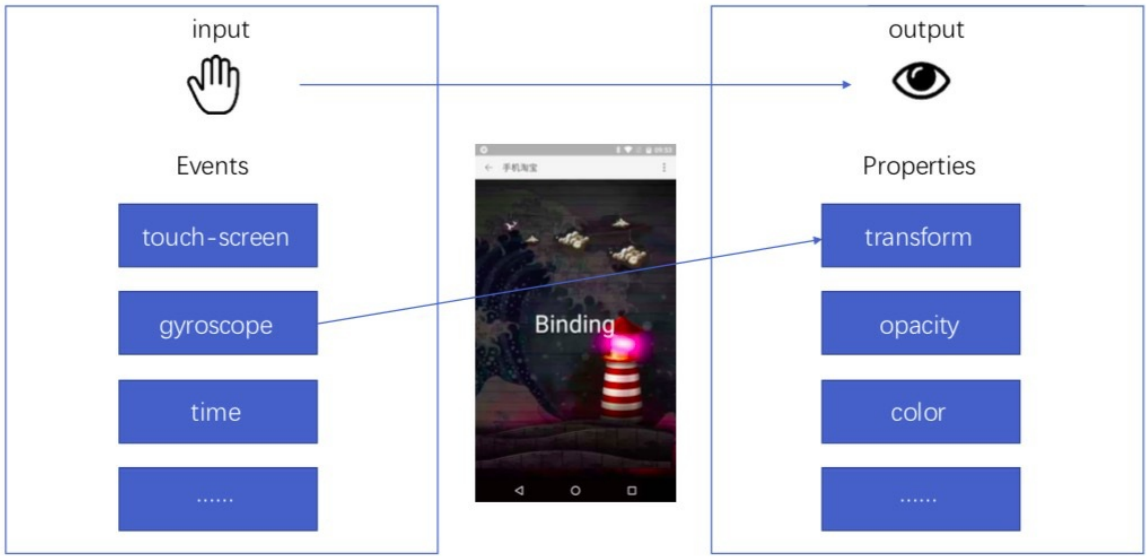
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

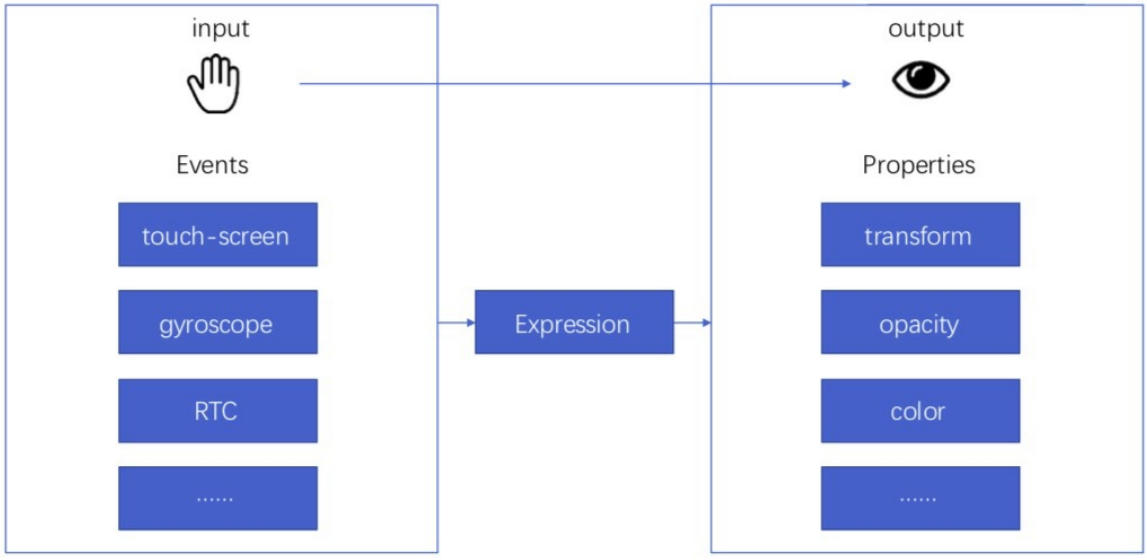


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

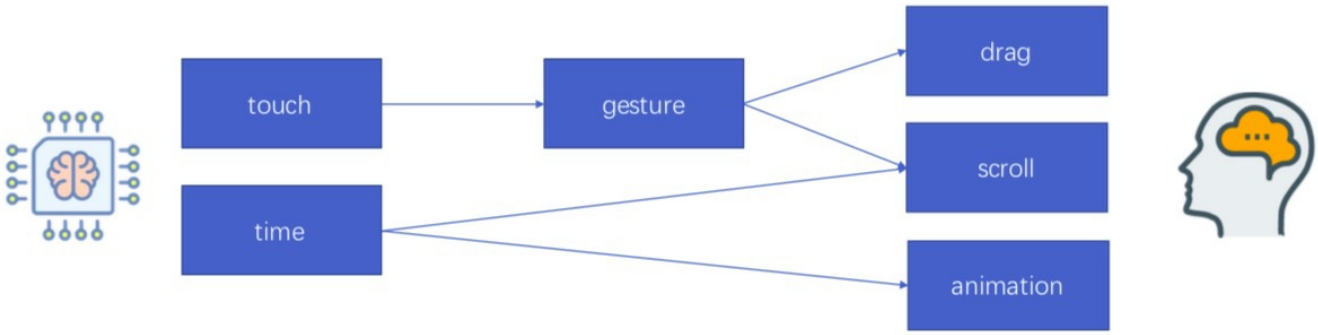
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

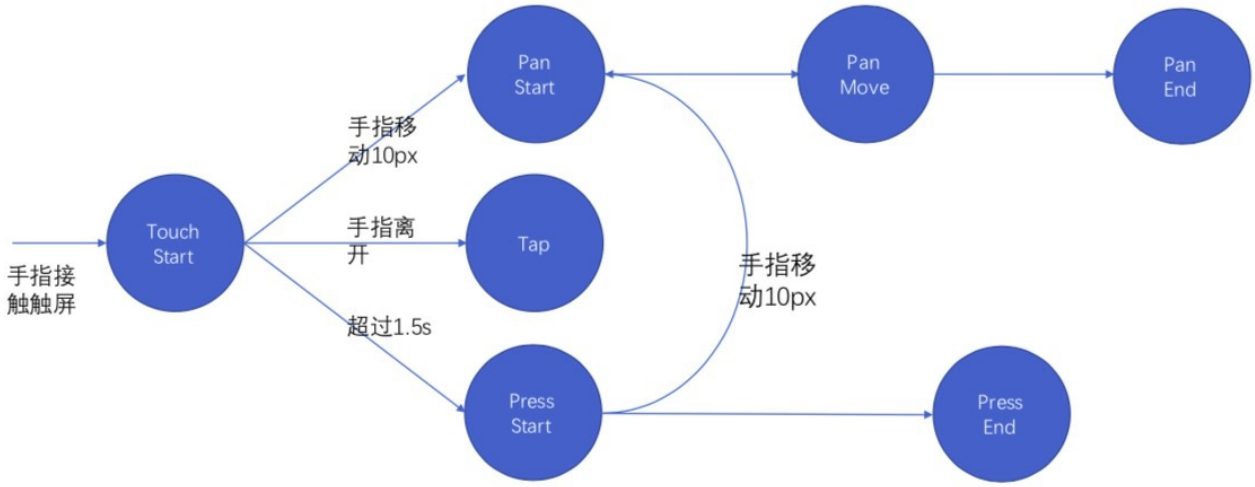
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

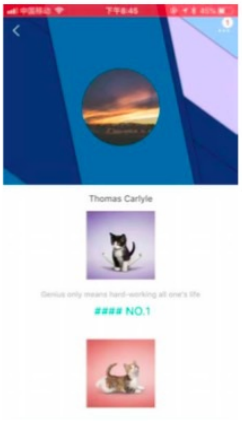
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

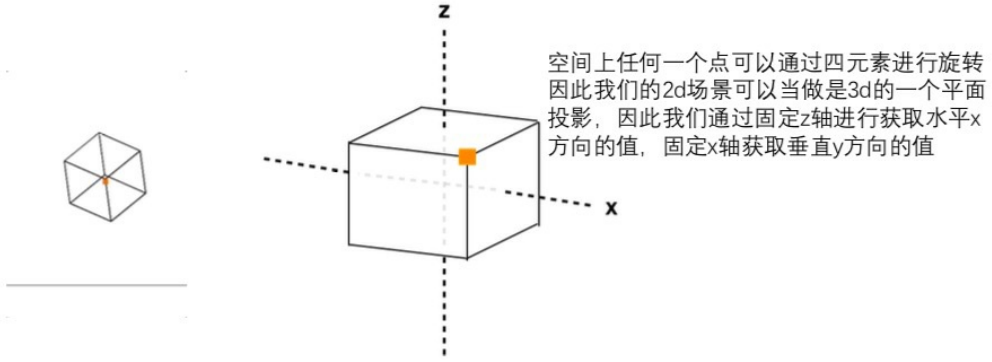
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

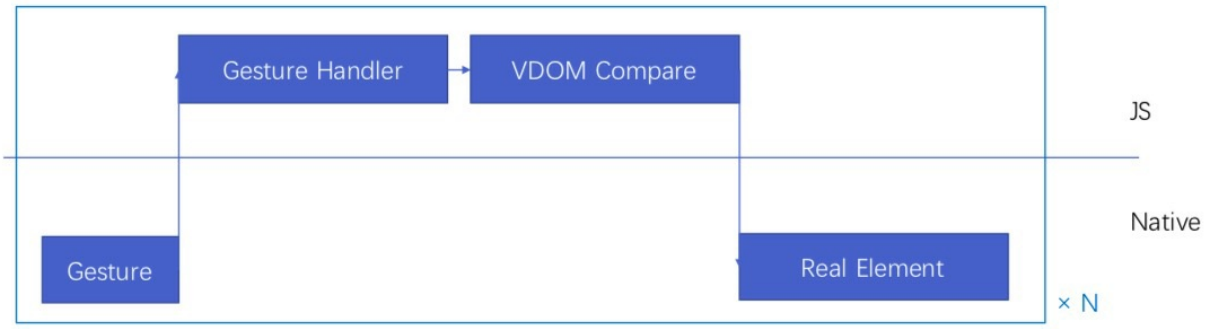
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

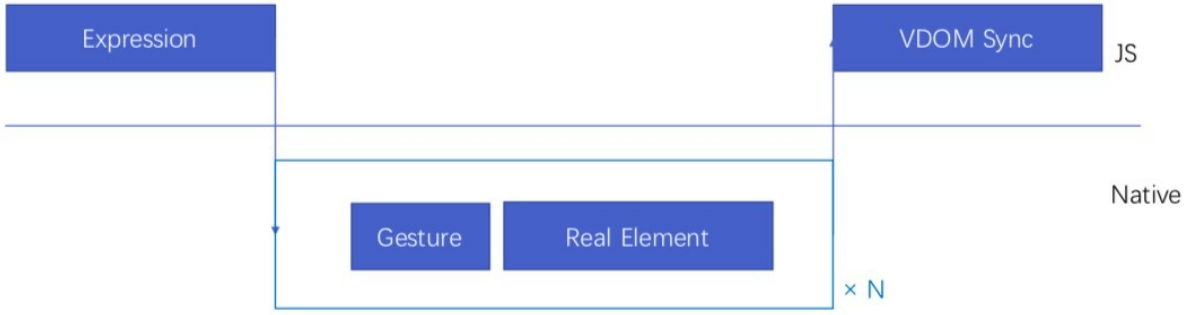
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

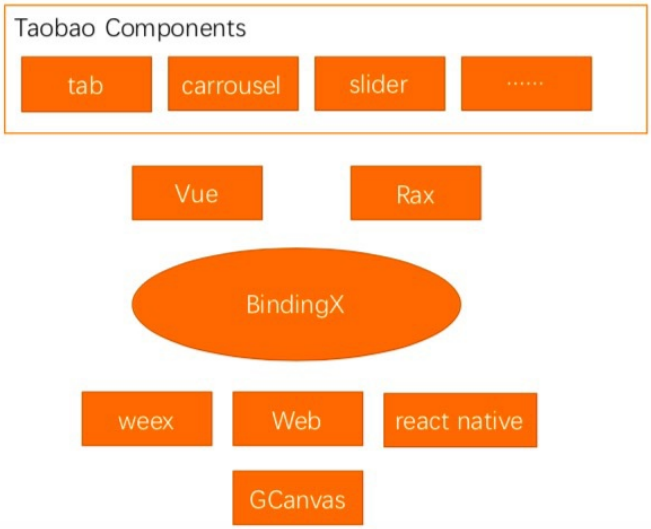
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。

分享内容大纲

Vue、React等现代前端框架很好地解决了组件化和数据视图解耦问题。而对前端来说，新交互永远是花费时间最多的工作，新交互也是前端团队的自然价值和核心竞争力之一。

在这次话题中，我会分享在交互的基础设施的建设上的一些思考和实践，包括图形图像基础、事件机制与视图层架构模式、交互管理框架等内容。

UI架构的演变

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

首先我们要了解一下历史。在70年代，大概是70年代的尾巴，1979年左右，有了特别有名的，MVC架构。

MVC之后，经过了差不多十几年的发展，到了90年代，准确地说应该是95年左右的时候，这个有一个公司的CTO，叫Mike，Mike在MVC的基础上，提出来了MVP。

到了 2005年，2005年微软的一个架构师，做WPF的，提出了MVVM模式。

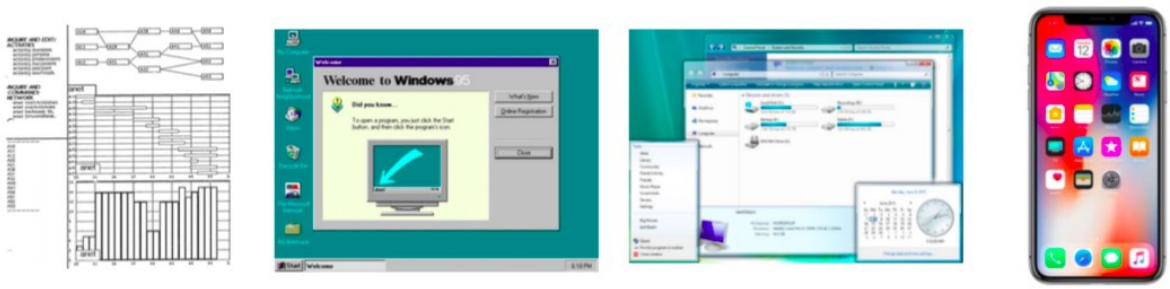
2014年左右的时候，出现了FLUX，这个是Facebook为了它的JSX和React提出的一种模式。

后来隔了短短的一年，2015年，同样是在React社区，出现了REDUX。

对于前端来说，我们为用户创造价值才是特别回答的一个问题，这么多年过去了，前端到底为用户创造了什么价值呢？

用户的界面也在同时发展

MVC(1970s) MVP(1990s) MVVM(2005) FLUX(2014) REDUX(2015)

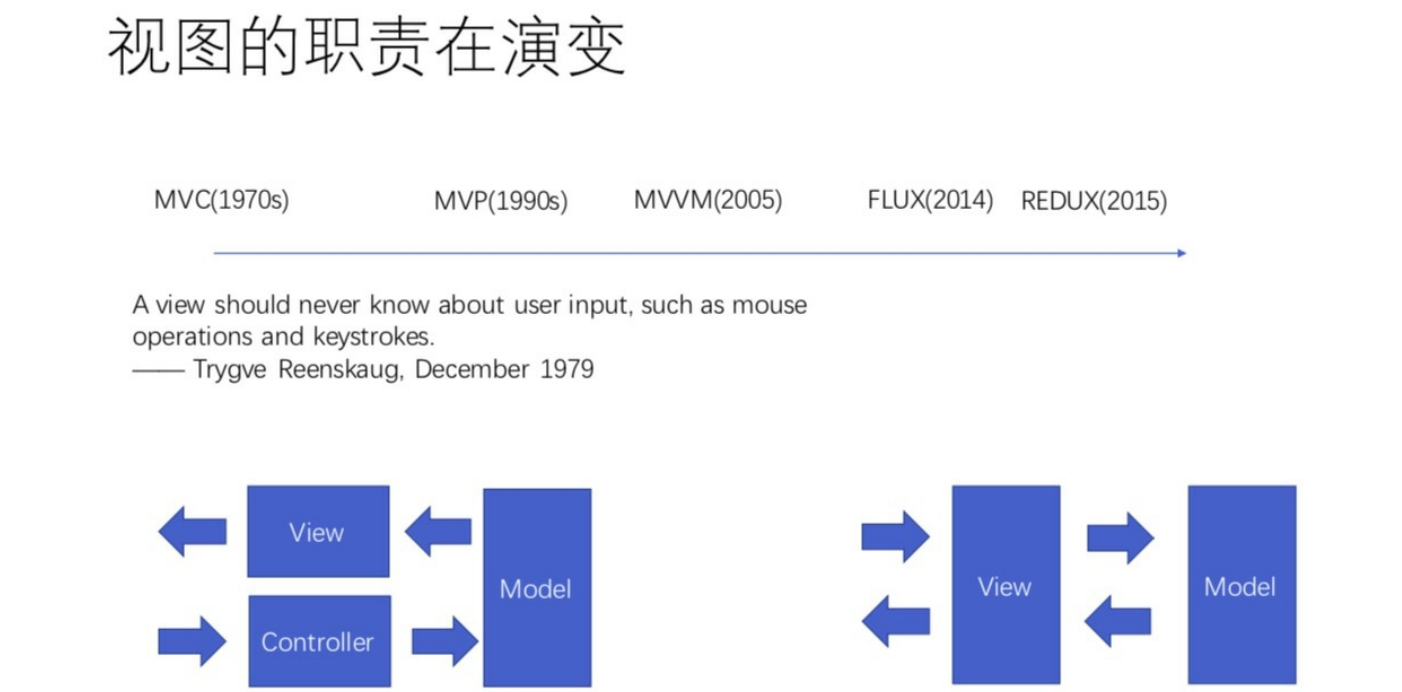


这是70年代，施乐公司做的一个软件管理的流程图软件，那个时代，整个的界面就是这个样子，施乐已经算比较先进的了。

再到90年代，当时这个画面还是很惊艳，按钮键是立体的。现在来看这个东西就有不那么美观了。

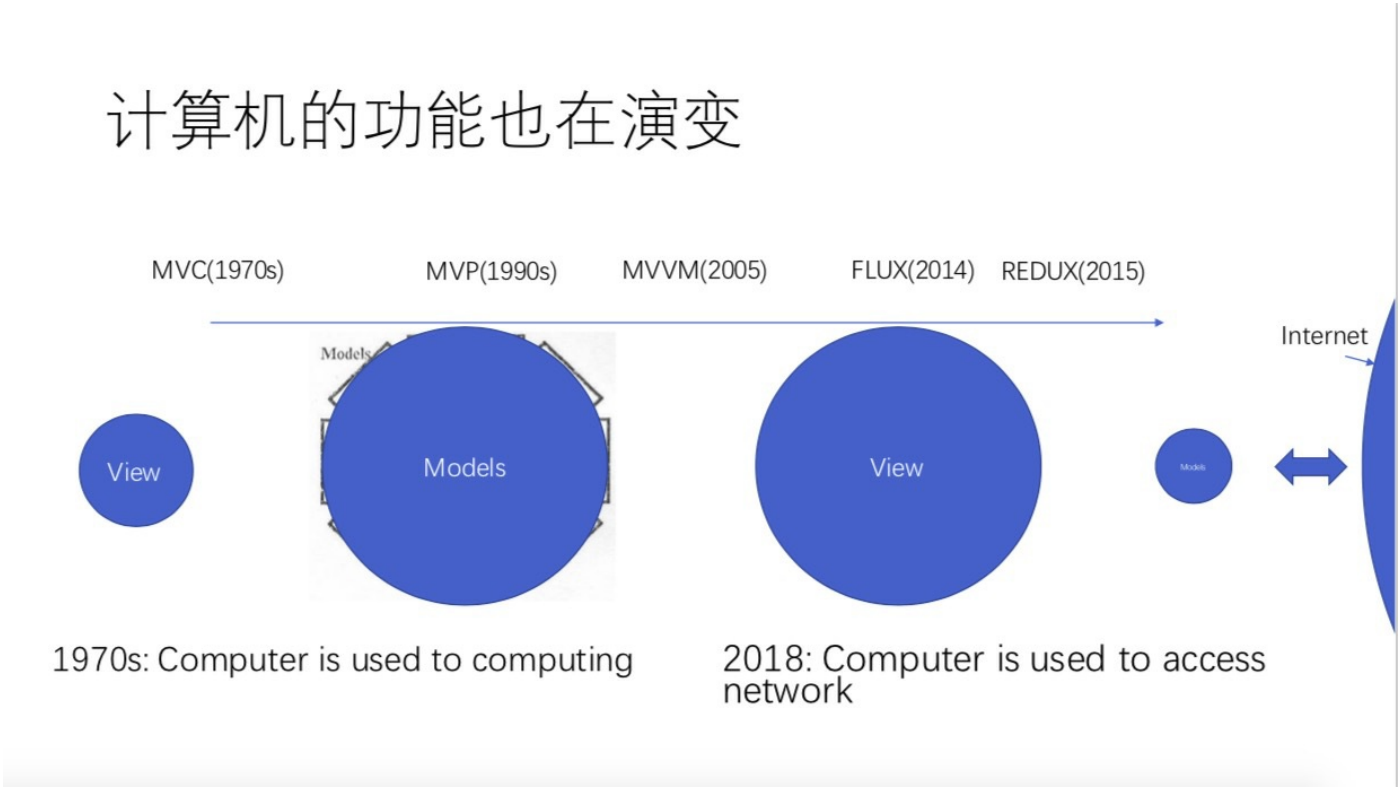
2006年左右的时候，Vista的界面已经开始有了一个非常大的变化了，这时已经是设计师在主导这个界面的了，但是性能并不佳。

再之后，手机出现了，比如iPhone的界面，这时不但交互模式发生了巨大的改变，而且屏幕也变了，甚至我们熟悉的鼠标不见了，变成了触屏。虽然两者之间操作上有一定的相似，但是变化还是非常的。



视图的职责也在演变，70年代，视图的职责是：任何一个视图，永远不应该知道用户的输入。

我们这个时代的视图则既负责输入，也负责输出，并且与Model之间有一个交互。

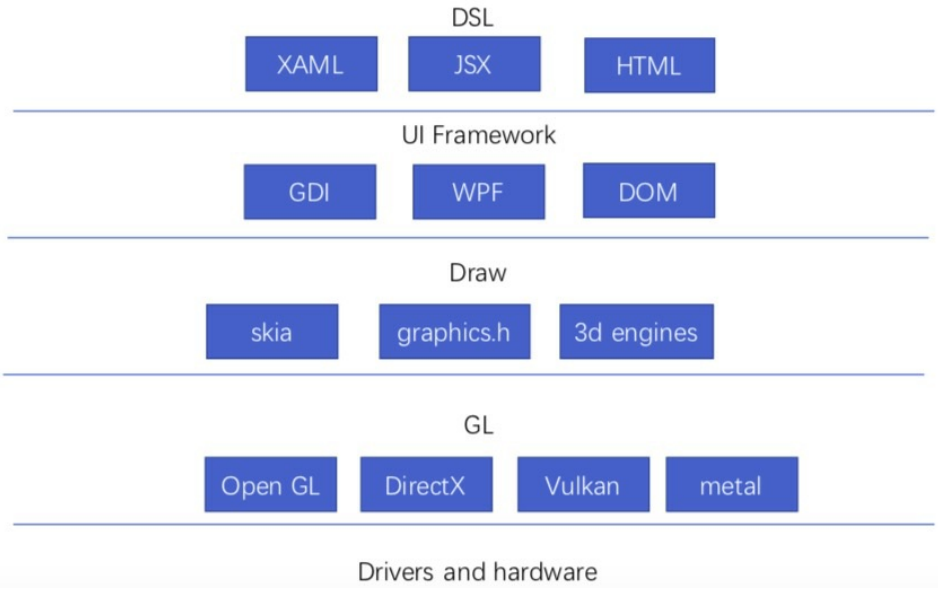


计算机的功能也在演变。70年代，计算机主要用来计算。

我们今天计算机主要用来上网，基本上，大家的计算机都是24小时联网的，你的手机也是24小时联网的，所以计算机的职责在发生变化。

这个变化对于UI有很大的影响，1970年的那个MVC那篇论文里的图，model很大，view很小，而到了2018年，今天我们很多的model，都是放在服务端的，而今天model的大小已经不是说一台机器上能去存的，你存在本地的只是视图展现一点点的model，这个是很小的一部分的东西。而同时view却越来越重要了。

视图技术变得越来越复杂



我们来看一下视图的技术。

从最底层的有很多人是做显卡和drivers，有这样的大佬人才。

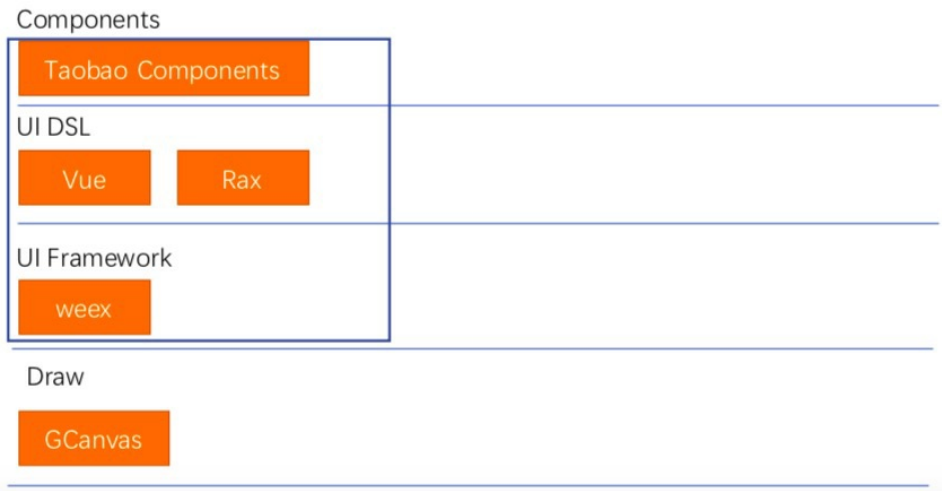
还有现在非常流行的OpenGL等的GL层，做这一层的人非常专业，基本上都集中在各种大公司，最近苹果和安卓还竞争，推出了新一代的这个GL架构。

还有一个这个Draw层，这一层的内容非常多，基本上就爆发了，skia是安卓的底层绘制系统，graphics.h是最早的C语言带的一个图形库，基本上相当于一个基础库，还有很多3D引擎。

UI Framework这一层，它提供了一套基本的UI结构，有了绘制层， 一般人都不会在绘制层直接去工作，需要有些控件，这层有我们比较熟悉的Dom。GJI是Windows的图形系统，WPF也是Windows的图形系统。

最上面其实会有一些DSL，这是描述图形的语言，WPF对应的就是XAML，JSX对应的是React，HTML大家都知道了，想说这个视图技术变得越来越复杂，

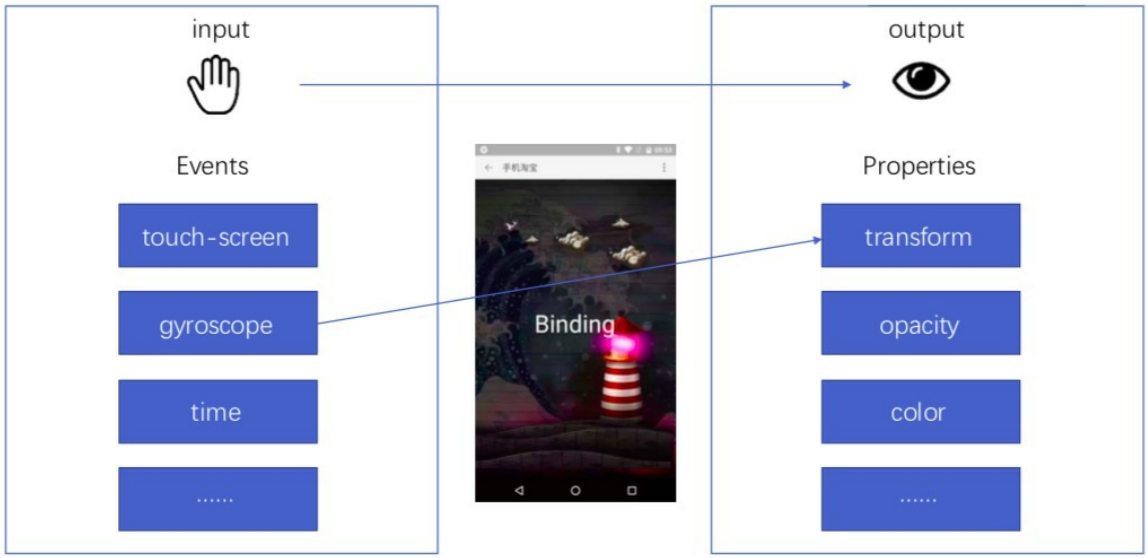
淘宝终端技术



那么我们的主战场是怎么样的，我们可以看一下淘宝终端技术在各层上的分布状况。

交互体系其实是这里面的一部分，但它不是这里面的全部，我觉得我们要讲这个交互呢，我们还是要做一下抽象的，我们要认识到，交互的本质是什么。

交互的本质抽象

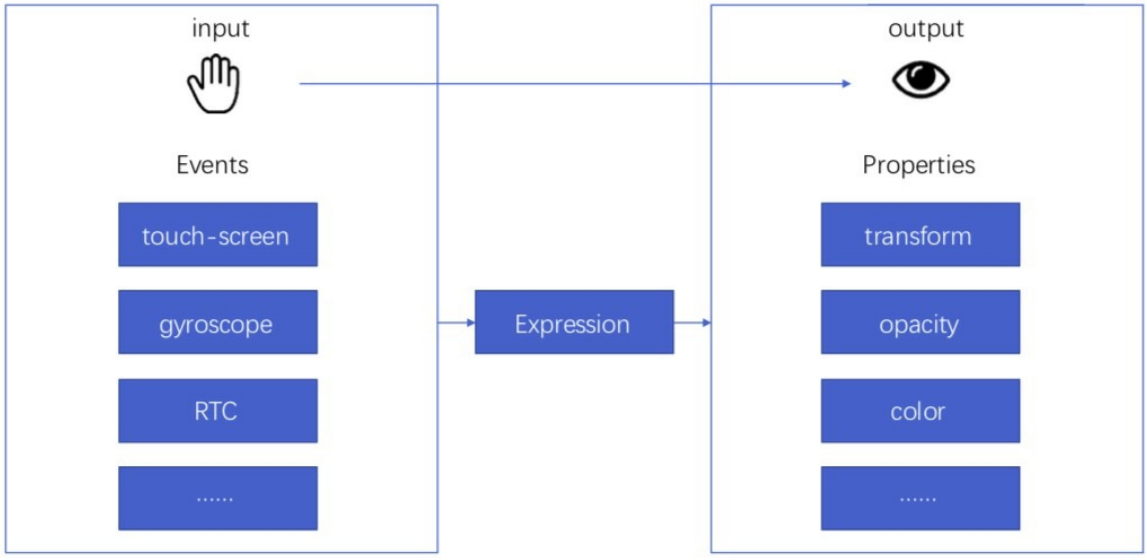


交互的本质是什么呢，我画了一个手和一个眼睛，其实无非是操作和看。

操作最常见的一个抽象的模式就是事件。这个比如说这个touch-screen事件，陀螺仪事件，或者是时钟芯片触发的持续事件，这些作为输入。

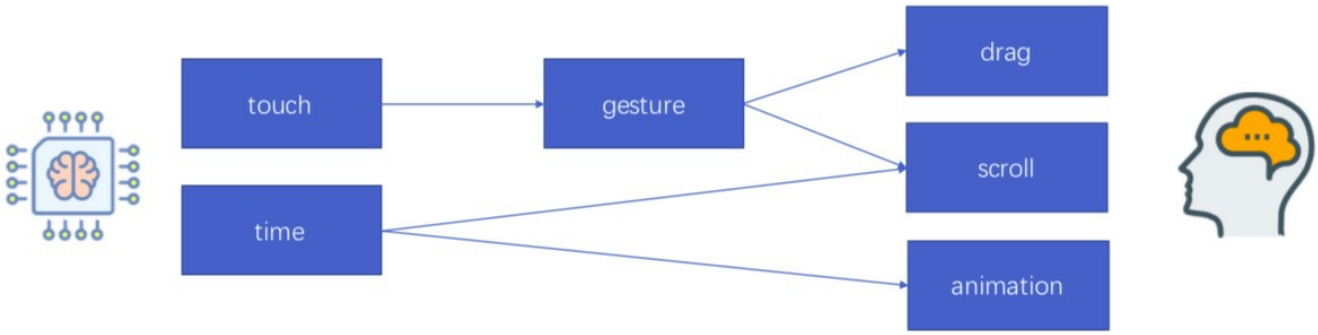
输出一定是通过属性的形式体现的，在任何一个现在的UI框架下，都是通过属性的方式反映出来的。transform是变形，opacity是透明度，color是颜色，这就是一个比较完整的抽象了。你在任意的输入和输出连成一条线后，它都会产生一种效果。

交互的设计——Expression



不过直接把陀螺仪得到的参数输入到transform里肯定是不行的，它需要有个关系，我们在这里面选择了Expression。我们可以用JavaScript去做计算。我觉得这是一个完备的抽象。

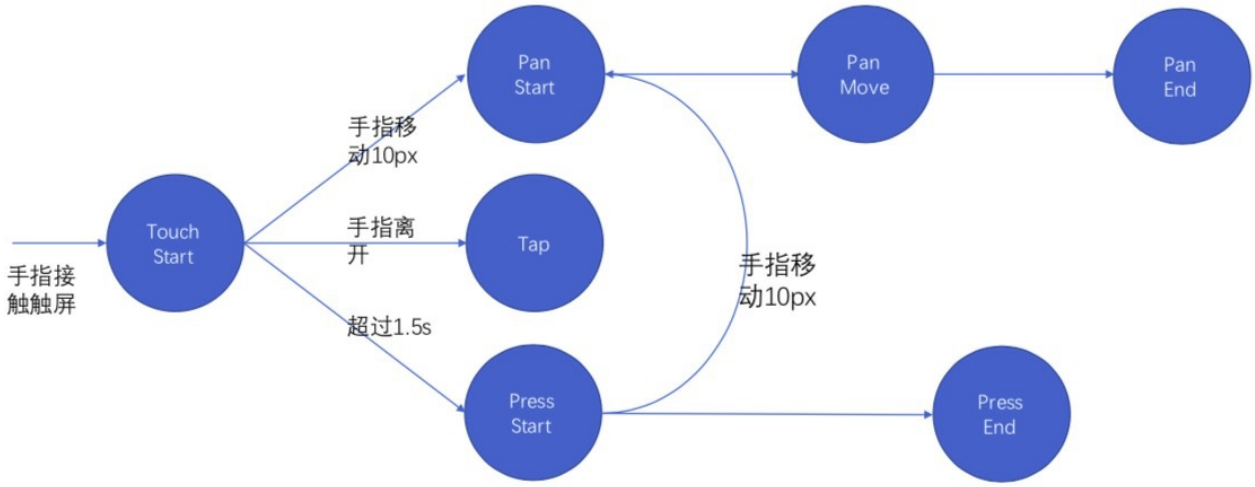
输入具有复杂性



不过这里还有一个坑是需要迈过去的，对计算机理解的输入跟人类理解的输入有非常大的偏差，对计算机来说呢，有多少种硬件，就有多少种输入。

我们发现输入非常复杂，在做基础设施建设的时候，我们在输入上面其实投入了很大的精力，最后出来的是一个更接近于人脑概念的一系列的输入。

Touch vs Gesture



比如说，touch和gesture，我们知道触屏其实是触屏事件，触屏事件其实非常简单，只有四个，touch start，touch move，touch end，touch cancel则不太常用。

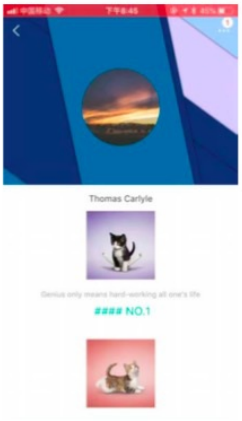
比如我想摁或者点一个东西，它都是是touch start，touch move，touch end，如果你要监听这些事件，中间的判断很繁琐，作为交互的基础设施，我们不可能提供这些给我们的前端工程师使用，我们肯定做一些操作。

比如手指移动10px，我们就认为这个touch start到了pan start，这个后面就是pan move，pan end这样，

手指很快离开，那么它就会产生一个tap事件。

如果超过1.5秒那就一个press start，如果手指没移呢，就会产生一个press end，如果手指移了，它还会产生一个pan start。所以gesture已经比touch复杂了很多了。

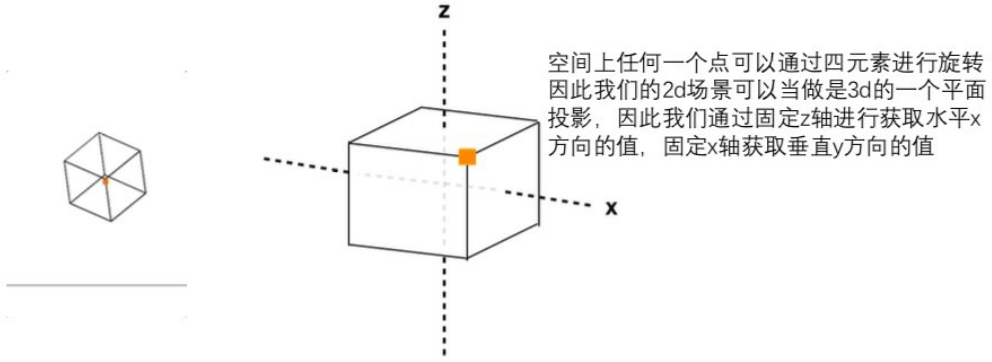
Scroll



scroll就在gesture的基础上又复杂了一层，它不但手指在屏幕的时候响应，手指离开屏幕的时候它也响应，比如说轮播，它是一个变形的轮播，它在轮播的过程中，不但产生位移，还会产生大小的变化，这就让用户更舒服一些。

还有一个滚动导航，一边滚动出来一个导航，近年来还有一个交互设计，不是滚动到某个位置导航出来，而是一直再往下滚动的时候它不出来，突然往上滚动一下，导航就出来。这个部分还有更难的设计交互，所以我们还需要在scroll的基础上再做一层。

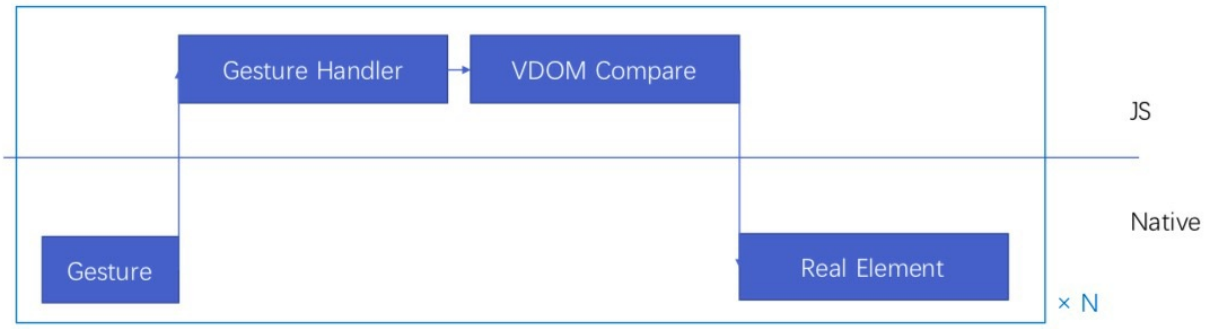
Binding —— input



我们再来看陀螺仪，它只提供了三个分量，并且它是0到360度，所以如果不经过任何处理，前端工程师基本上是没有办法用的，比如在某个角度，它可能会突然从0跳变成360度，这个在数据计算时候非常可怕。

所以我们建立这样一个模型，我们把手机看作这样一个立方体，去计算在空间中对立方体产生的旋转效果，我们拿着立方体上面的一个点呢，去做我们定位的一个依据。

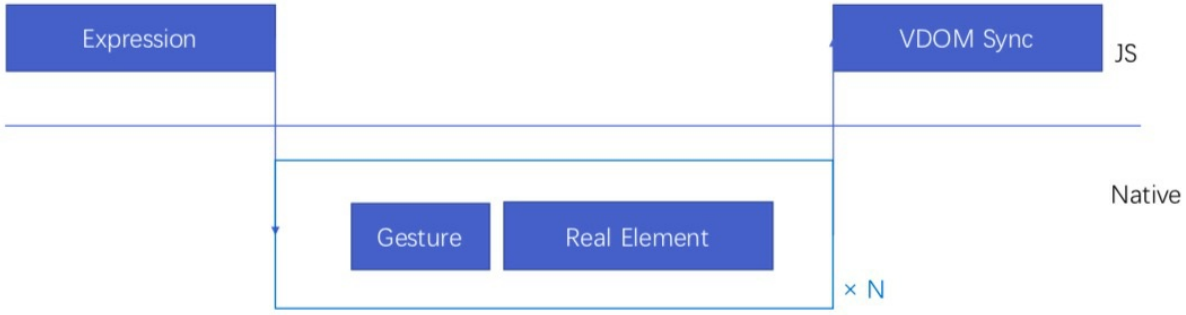
Native-JS模式的问题



因为我们在用Weex，所以有一个Native跟JS通讯的问题，比如说从gesture事件到gesture handler，这一步就会到JS去执行，图中我们可以看到这个线，跨过中间JS和Native的分界线，跨越地非常频繁。

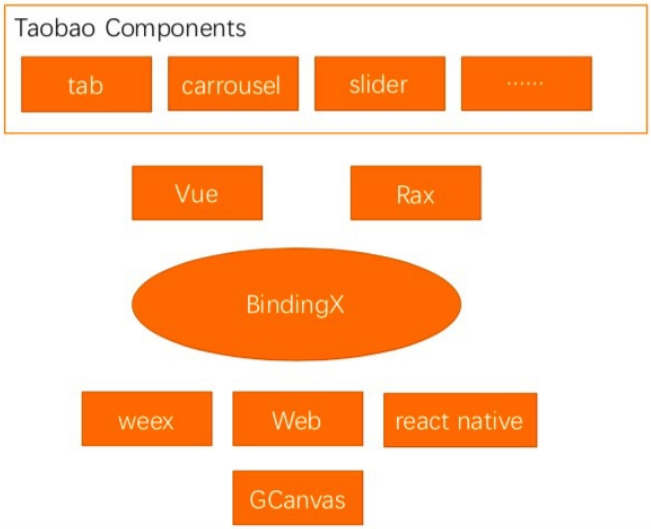
假如一个Touch move事件或者Pan move事件，你手指每移动一小点它都会触发一次JS跟Native的一个跨语言通讯，所以说整个的性能会非常差，最后基本上会有5毫秒到10毫秒左右的一个延迟，有60帧的话，每一秒钟有300毫秒被占掉了，帧率就下去了。

Binding模式



这就是我们最初开始做Binding模式的原因。我们的Binding模式，expression传递一次给Native，然后它会去做大量的绑定，所有的过程都是由Native来完成的，Native做完了以后，还需要再更新一下VDOM，所以这操作就完全由Native完成，通讯次数就降下来了。除此之外，我们还额外收获了性能上的收益。

淘宝终端交互技术



我们的结论，其实淘宝一个交互体系是这样的，是以Binding为核心，下面的平台支持了weex，Web，React Native。DSL上面，我们支持了View和Rax两种，在上面，是由我们自己建的Components体系。

更多想象空间

Binding 和 矢量图

ixter

Binding 和 Shader

最后，还有一个展望，我们用绘制层相结合，会有更多的想象空间，我们通过各种各样的输入、手势、时间、陀螺仪，我们其实可以去控制矢量图，也可以去控制绘制，这些都是前端未来的想象空间。

如果你对今天的内容有所思考，可以给我留言，我们一起讨论。