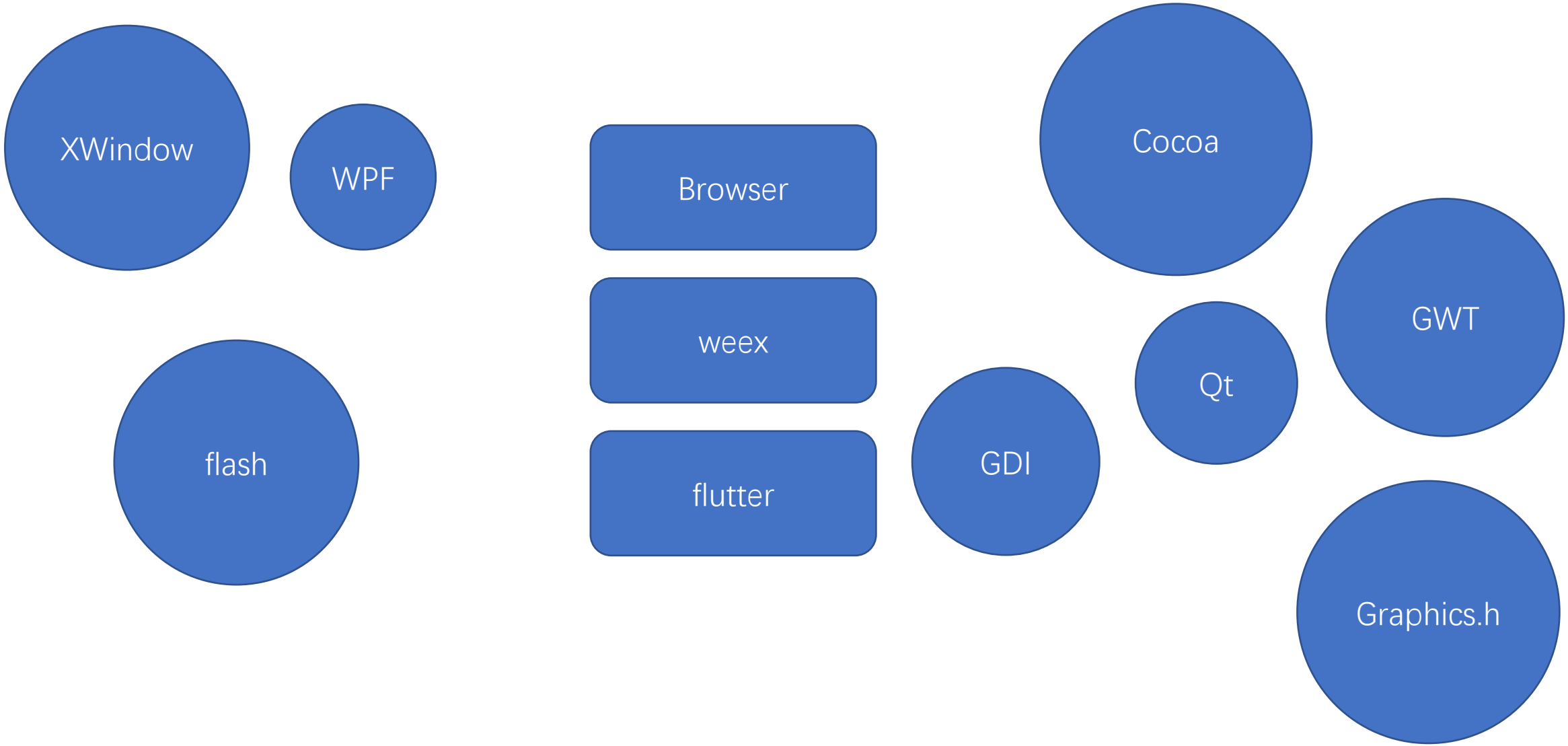
从spritejs谈UI系统

UI系统

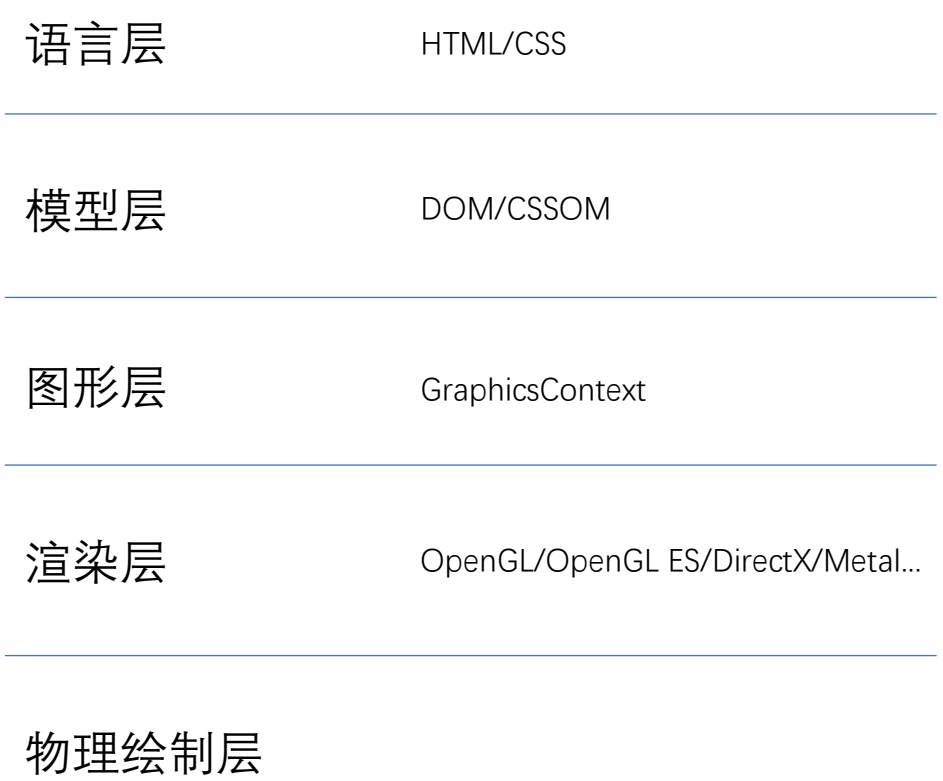
分析几个UI系统

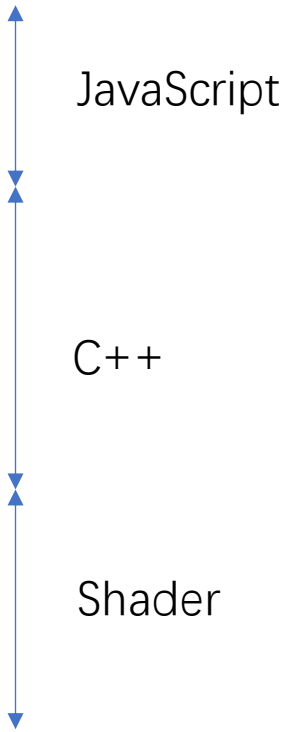

XWindow
WPF
Browser
weex
flutter
Cocoa
GWT
Qt
flash
GDI
Graphics.h

UI系统的分层

语言层	提供界面描述语言和编程范式	HTML XAML **ML
模型层	提供布局、层级关系和概念模型	DOM Controls
图形层	提供点、线、面、阴影、渐变等的绘制能力	Skia Quartz
渲染层	逐像素生成位图的能力，通常借助于GPU	DirectX OpenGL Vulkan Metal
物理绘制层	调用显卡驱动，实际把位图画到屏幕	ATI Nvidia

Browser





- 浏览器把界面和语义做了分离，所以产生了HTML和CSS两种语言，相应的模型层面也有DOM和CSSOM
- Browser在图形层做了兼容处理，支持从PC到移动的所有图形框架

Weex

语言层	Vue/Rax/Vanilla	JavaScript
模型层	Android UI/Cocoa Touch	Java/OC
图形层	Skia/Quartz	C++
渲染层	OpenGL ES/Metal/Vulkan	C++
物理绘制层		Shader

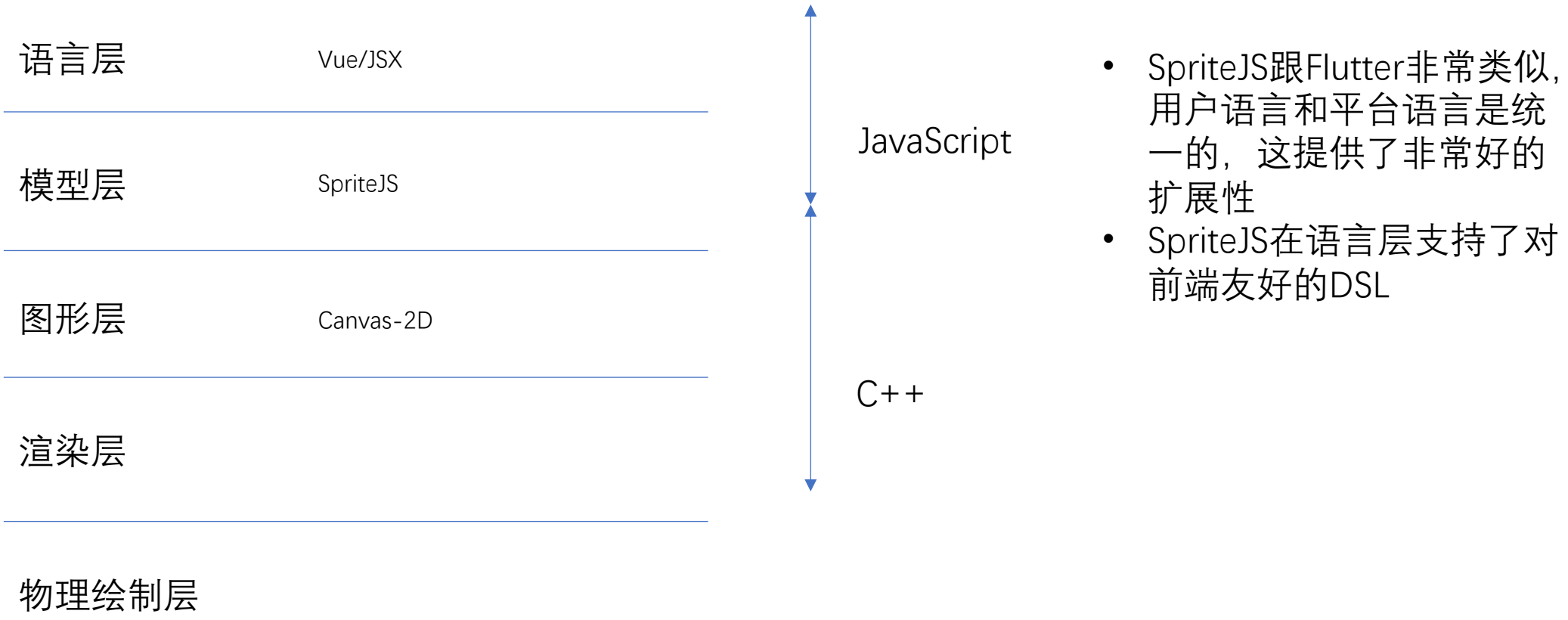
- Weex提供了多DSL支持，官方提供了vue和react两种风格
- weex在模型层使用了虚拟DOM，因此在JS和Native环境各有一份
- Weex的模型对Android和iOS做了抽象，因此是两者的最大公集加上一些自己补充的组件

Flutter

- Flutter完全抛弃了Android和iOS的原生的UI，建立了全新的模型
- Flutter完全依赖skia，通过移植skia来保证跨平台使用
- 完全使用Dart语言来实现，用户语言和平台语言统一

SpriteJS

SpriteJS



- SpriteJS跟Flutter非常类似，用户语言和平台语言是统一的，这提供了非常好的扩展性
- SpriteJS在语言层支持了对前端友好的DSL

SpriteJS-JSX

```
const {Group, Path, Scene, Sprite} = spritejs

const scene = new Scene('#demo-quickStart', {
  viewport: ['auto', 'auto'],
  resolution: [800, 800],
})

const layer = scene.layer();

const group =
  <Group
    size={[300, 300]}
    pos={[400, 400]}
    anchor={[0.5, 0.5]}>
    {
        [...Array(6).keys()].map( i => <Path path={{
          d: 'M0 0L 50 0A50 50 0 0 1 43.3 25z',
          transform: {scale: 3, rotate: -15},
          trim: true,
        }}
        pos={[150, 150]}
        anchor={[0, 0.5]}
        strokeColor='red'
        fillColor={`rgb(${i * 139 % 255}, 0, 0)`}
        rotate = {i * 60}></Path> )
    }
  </Group>
layer.append(group);

group.animate([
  {rotate: 0},
  {rotate: 360},
], {
  duration: 3000,
  iterations: Infinity,
})
```

SpriteJS-JSX

config

```
module.exports = {
  presets: [
  ],
  plugins: [
    [require('babel-plugin-syntax-jsx')],
    [
        require('babel-plugin-transform-react-jsx'), {
          pragma:'spritejs.createElement'
        }
    ]
  ]
}
```

runtime

```
function createElement(type, attrs, content) {
  var Node = typeof type === 'string' ? getNodeType(type) : type;
  if (!Node) return null;
  var sprite = new Node(typeof content === 'string' ? content : undefined);

  if (attrs !== null) {
    sprite.attr(attrs);
  }

  if ((0, _typeof2.default)(content) === 'object' && sprite.append) {
    if (content instanceof Array) {
      sprite.append.apply(sprite, (0, _toConsumableArray2.default)(content));
    } else {
      sprite.append(content);
    }
  }

  return sprite;
}
```

SpriteJS-Vue

- 用sprite-vue.min.js替换Vue
- 像Vue一样使用！
- 好消息：据说Vue3.0之后我们不需要改源码了

```
Vue.component('my-circle', {
  data () {
    //...
  },
  props: [
    'radius',
    'x',
    'y'
  ],
  methods: {
    click () {
      const state = this.state
      this.state = state === 'stateA' ? 'stateB' : 'stateA'
    }
  },
  template: `<sprite ref="circle" anchor="0.5" :x="x" :y="y" :size="[2*radius, 2*radius]"
:states="states" :actions="actions" :state="state" :borderRadius="radius" @click="click"></sprite>`
})

new Vue({
  el: '#app',
  data () {
    return {
      font: '48px Arial',
      fillColor: '#f50'
    }
  }
})
```

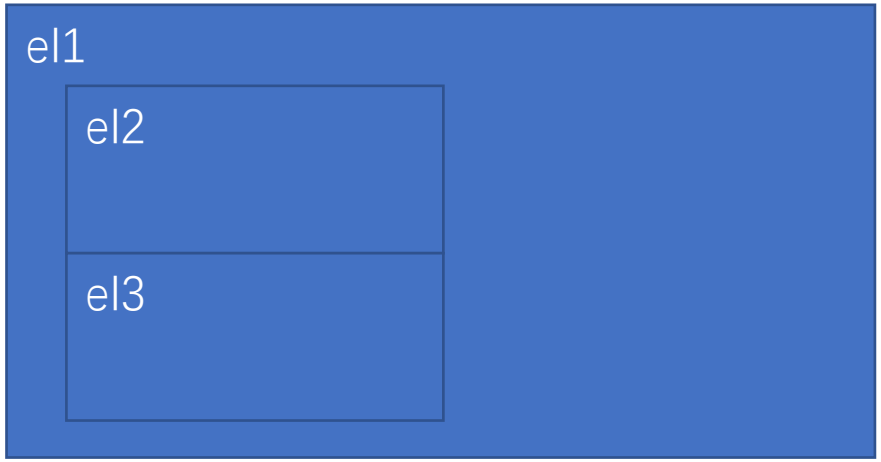
```
export function createElement (tagName: string, vnode: VNode): Element {
  let isSpriteNode = !isReservedTag(tagName) && isValidNodeType(tagName)
  let hasPrefix = false
  if (tagName.startsWith('s-')) {
    tagName = tagName.slice(2)
    hasPrefix = true
    isSpriteNode = isValidNodeType(tagName)
  }
  if (isSpriteNode) {
    let attrs = {}
    if (vnode.data && vnode.data.attrs) {
      attrs = vnode.data.attrs
      if (!vnode._hasTransition) {
        // set transition attributes
        let parent = vnode.parent
        while (parent && parent.tag.startsWith('vue-component-')) {
          if (parent._hasTransition) {
            const { states, actions } = parent.data.attrs
            attrs.states = Object.assign({}, states, attrs.states)
            attrs.actions = Object.assign({}, actions, attrs.actions)
            break
          }
          parent = parent.parent
        }
      }
    }
    if (tagName === 'scene') {
```

SpriteJS-树形结构

- Group
  - 在SpriteJS中，Group实现了子元素
  - Group的多层嵌套，形成了树形结构
- 虚拟Group
  - anchor（[0,0]）
  - size（[0,0]）
  - borderWidth（0）
  - borderRadius（0）
  - bgcolor（''）

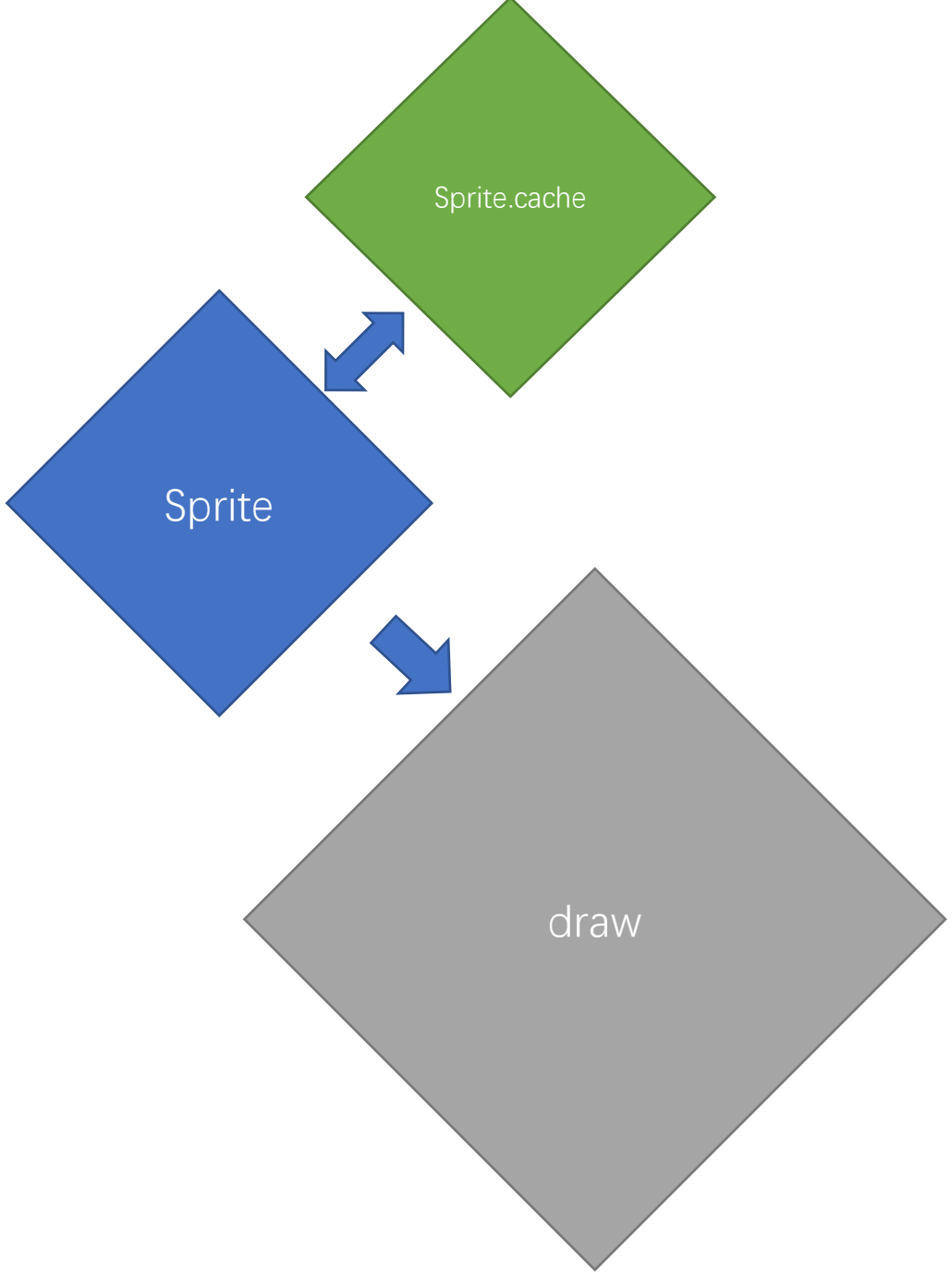
SpriteJS-缓存

- 浏览器的Compositing



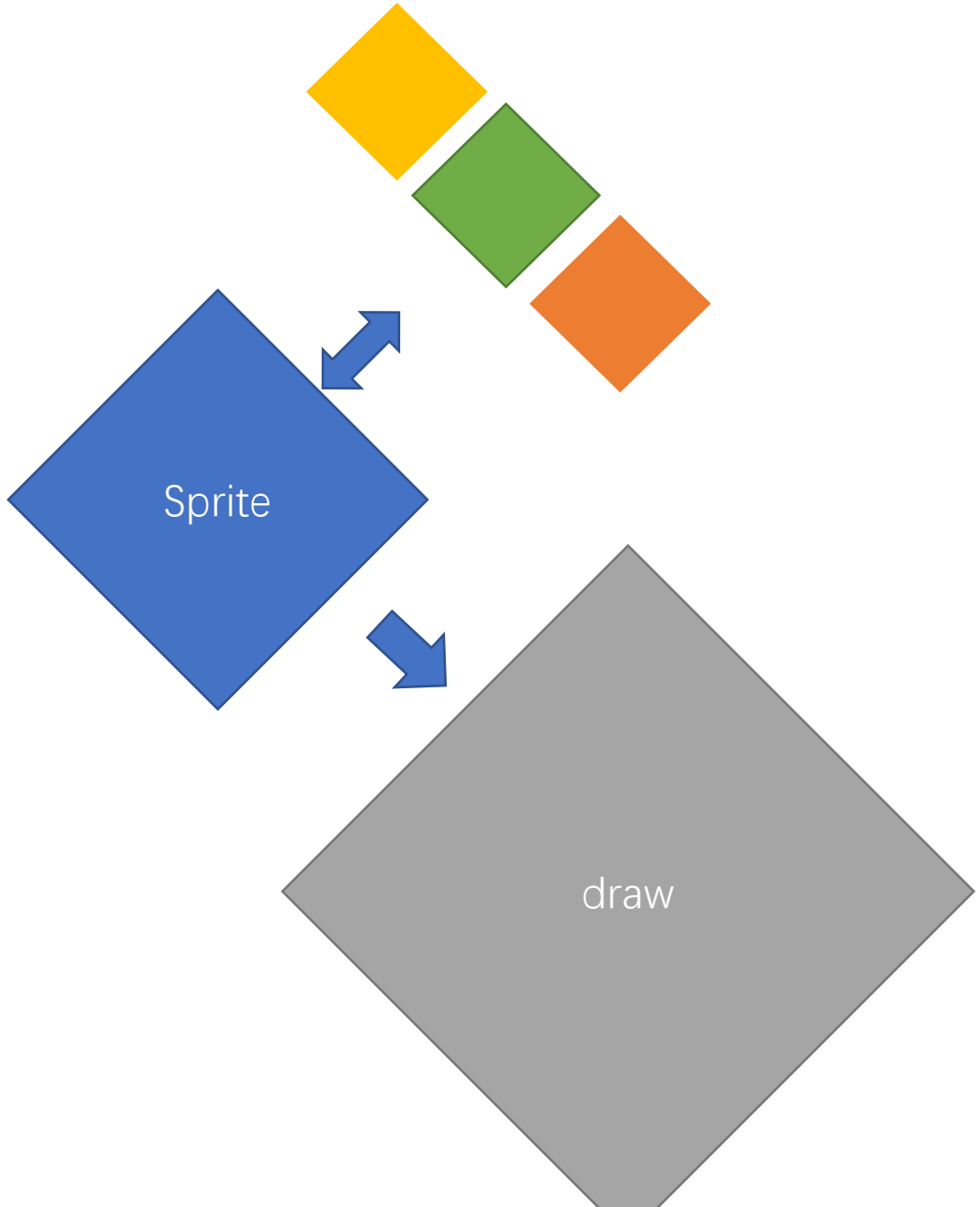
SpriteJS-缓存

- 缓存触发机制



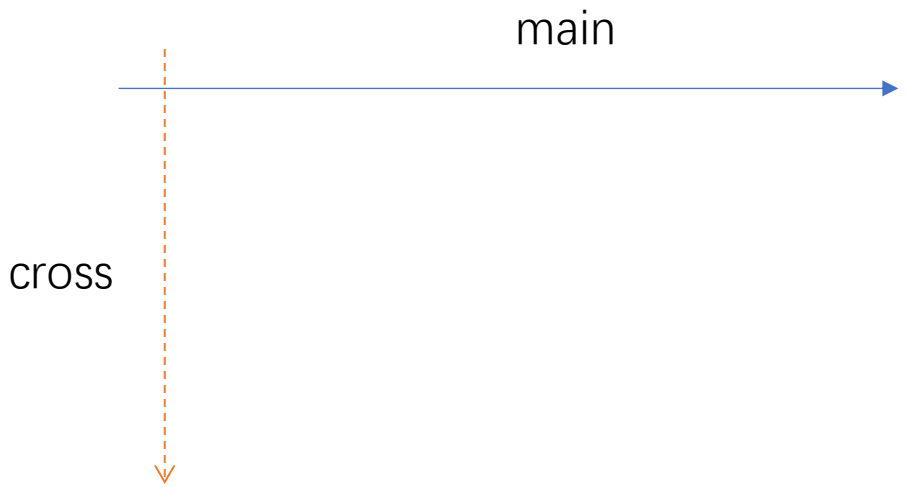
SpriteJS-缓存

- 自定义缓存机制



SpriteJS-flex布局

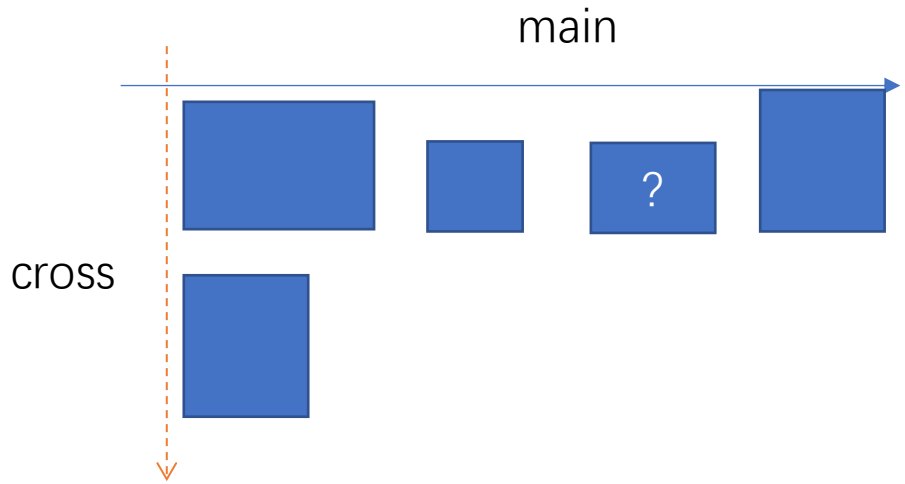
- 主轴和交叉轴



Flex direction决定了主轴和交叉轴方向
默认情况下，主轴属性是width、x、left、right
交叉轴属性是height、y、top、bottom

SpriteJS-flex布局

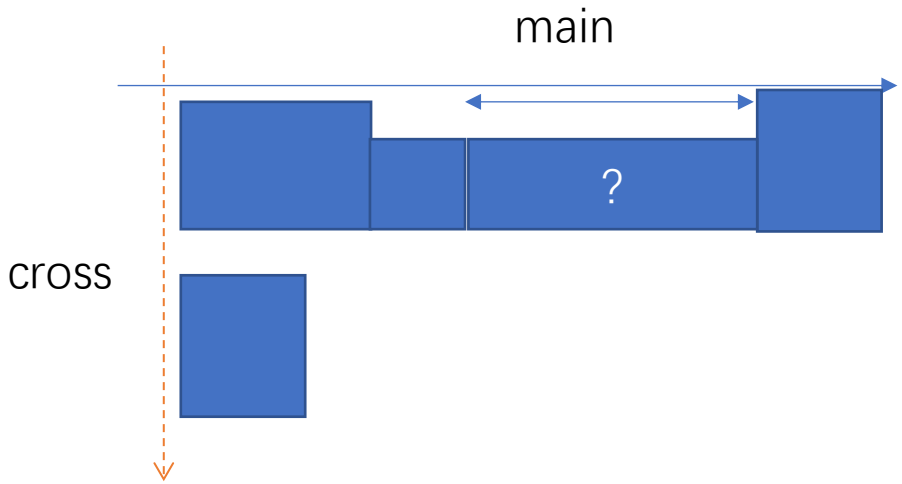
- 把元素收集到行中
  - 每行可能有剩余空间
  - 如果flex-wrap不允许换行就强行收进一行



把元素收集进各行

SpriteJS-flex布局

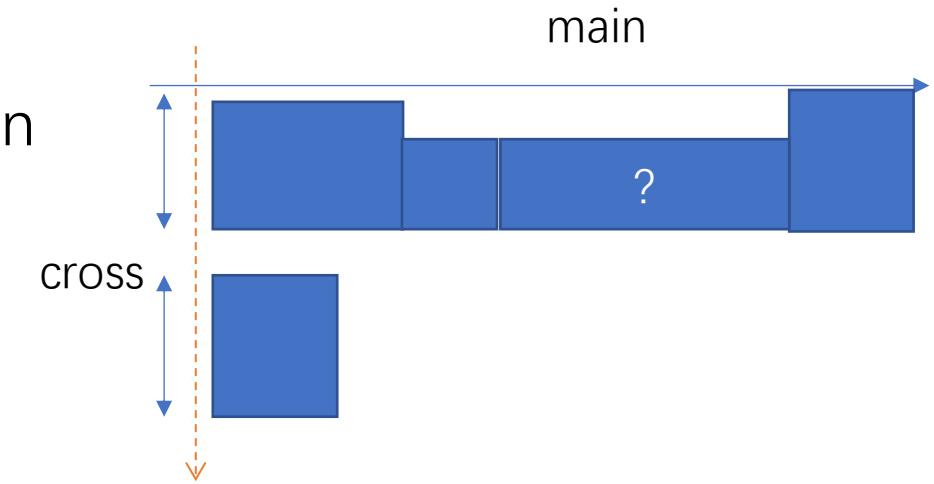
- 计算主轴方向：
  - 根据flex属性，把剩余空间分配给元素
  - 剩余空间为负数，则等比压缩元素



把元素收集进各行

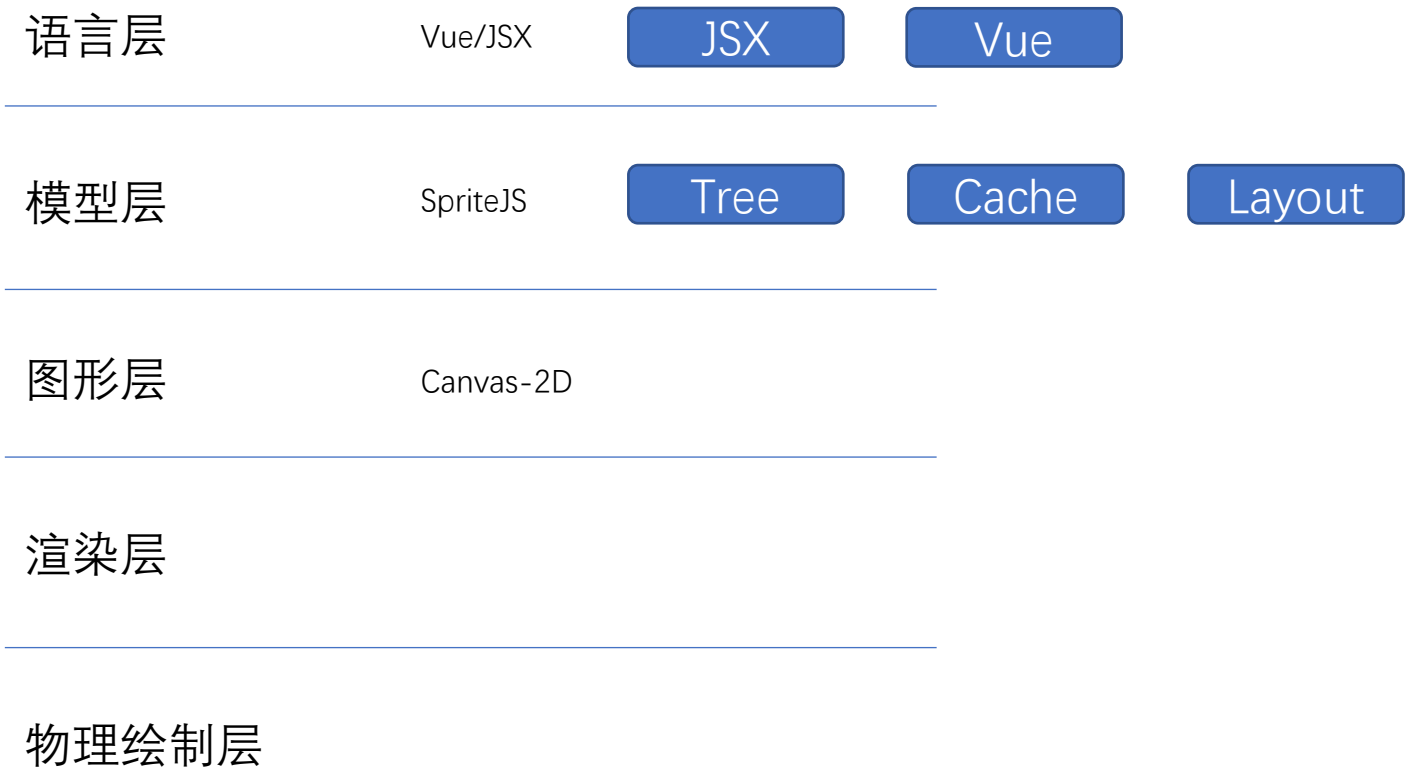
SpriteJS-flex布局

- 计算交叉轴方向：
  - 根据最大的元素的交叉轴尺寸确定每一行的尺寸
  - 把交叉轴剩余空间根据align来处理



把元素收集进各行

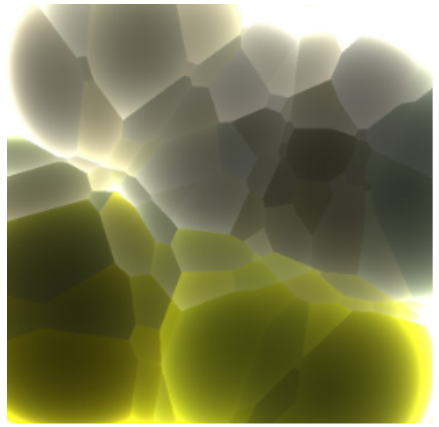
SpriteJS

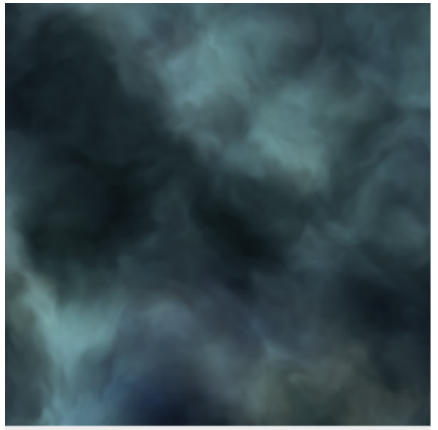

语言层
Vue/JSX
JSX
Vue
模型层
SpriteJS
Tree
Cache
Layout
图形层
Canvas-2D
渲染层
物理绘制层

未来展望

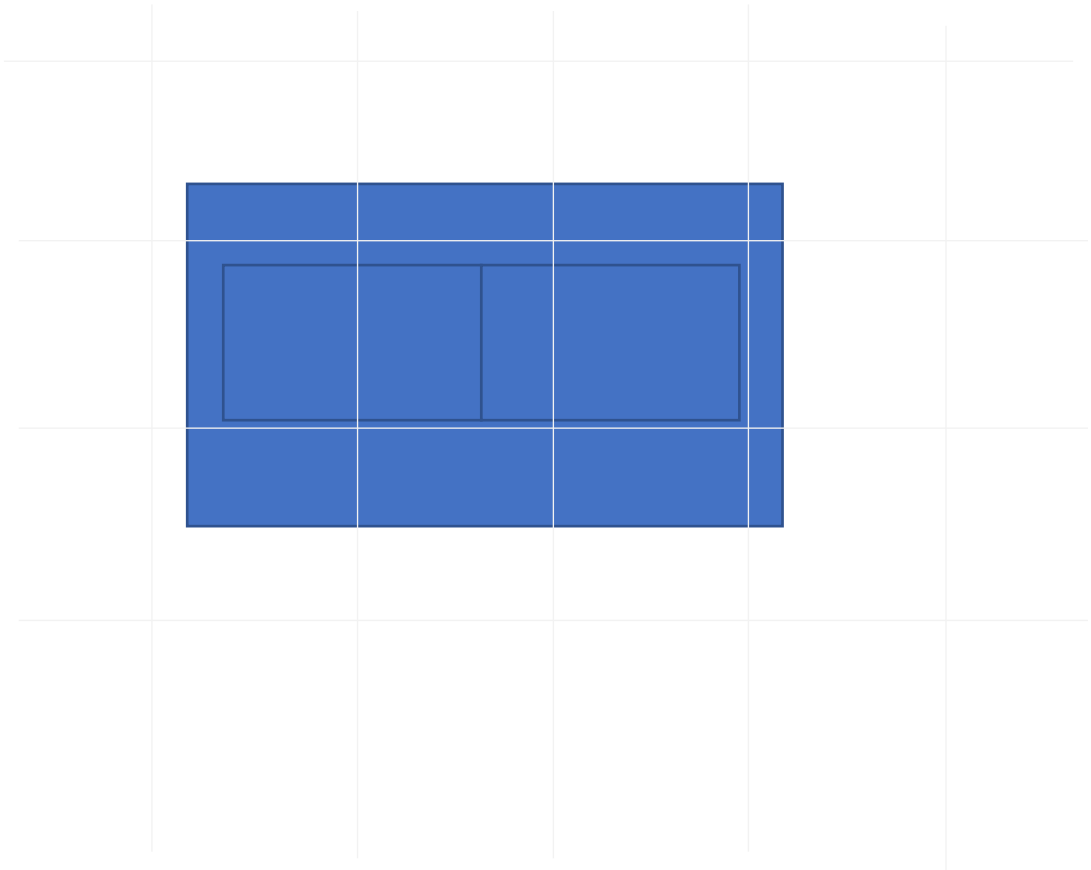
渲染层级下沉

语言层	Vue/JSX
模型层	SpriteJS
图形层	SpriteJS
渲染层	Canvas-WebGL
物理绘制层	



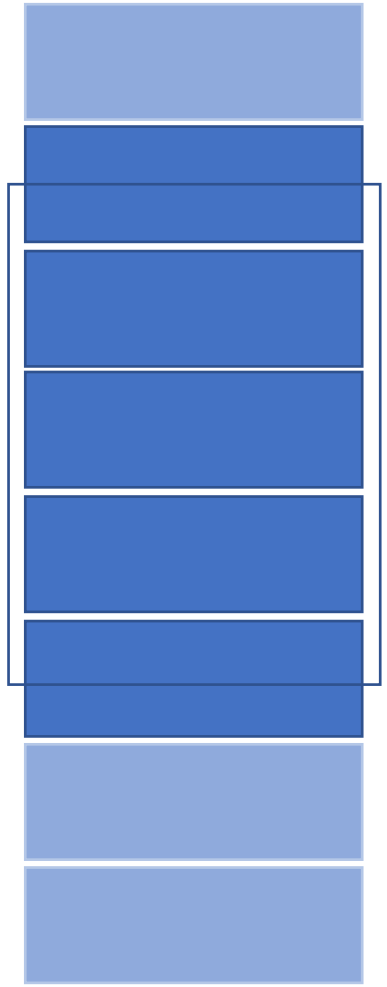


引入脏矩形算法



- 脏矩形算法是将整个可视区域分割成均匀的方格
- 当一个元素属性发生变化时，仅重新绘制它覆盖的矩形区域（即脏矩形区域）
- 对于局部变化较多、整体面积较大的场景，脏矩形算法可以显著提升性能
- 在SpriteJS中实现脏矩形，可以允许业务订制脏矩形尺寸，还可以自由选择关闭脏矩形检测

ScrollView



- 现在SpriteJS还不支持ScrollView
- 一旦有了scrollView，即可实现虚拟化
- 虚拟化是把渲染操作延迟到元素滚动到视口区域之后进行
- 虚拟化后页面的性能只跟屏幕相关，对于内容较多的大型页面，这项技术有巨大优势
- Weex就是通过虚拟化技术获得复杂页面的性能的
- 目前原生浏览器还没有任何一种引入虚拟化技术

结语

- 凡是能用JavaScript实现的东西，必定会出现一个JavaScript的版本的实现，UI系统也是一样。
- JavaScript能快过C++吗？不能，但是一个自上而下的纯JavaScript的UI系统，可能整体快过C++/JS混合的浏览器。
- 我认为SpriteJS现在需要开发者，多过于需要用户，所以我不推荐你们来使用SpriteJS，我希望你们**加入**SpriteJS

Q & A

- 微博 @寒冬winter
- Github wintercn