# 组件化开发

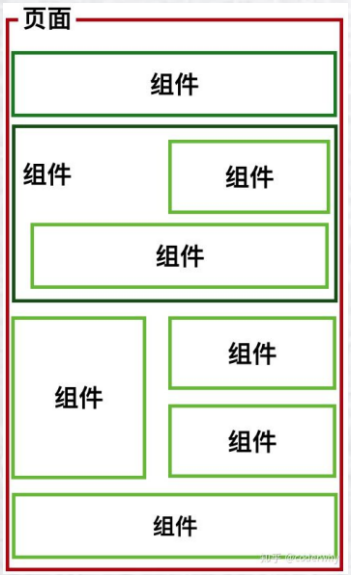
讲师：王红元
微博：coderwhy

更多学习资源

老师答疑交流

# 什么是组件化?

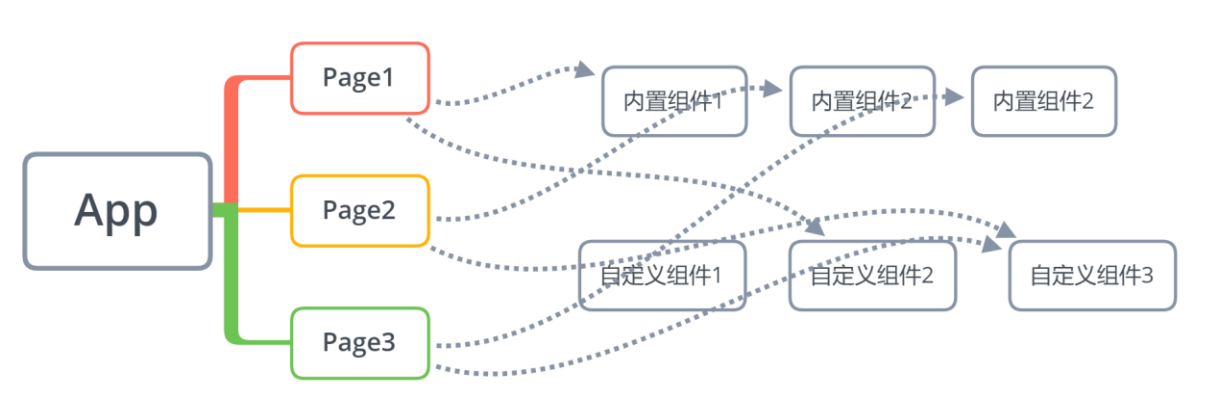
- 人面对复杂问题的处理方式：
  - 任何一个人处理信息的逻辑能力都是有限的
  - 所以，当面对一个非常复杂的问题时，我们不太可能一次性搞定一大堆的内容。
  - 但是，我们人有一种天生的能力，就是将问题进行拆解。
  - 如果将一个复杂的问题，拆分成很多个可以处理的小问题，再将其放在整体当中，你会发现大的问题也会迎刃而解。
- 组件化也是类似的思想：
  - 如果我们将一个页面中所有的处理逻辑全部放在一起，处理起来就会变得非常复杂，而且不利于后续的管理以及扩展。
  - 但如果，我们讲一个页面拆分成一个个小的功能块，每个功能块完成属于自己这部分独立的功能，那么之后整个页面的管理和维护就变得非常容易了。


- 我们将一个完整的页面分成很多个组件。
- 每个组件都用于实现页面的一个功能块。
- 而每一个组件又可以进行细分。

# 小程序组件化思想

- 小程序在刚刚推出时是不支持组件化的，也是为人诟病的一个点.
  - 但是从v1.6.3开始，小程序开始支持自定义组件开发，也让我们更加方便的在程序中使用组件化.




- 组件化思想的应用：
  - 有了组件化的思想，我们在之后的开发中就要充分的利用它。
  - 尽可能的将页面拆分成一个个小的、可复用的组件。
  - 这样让我们的代码更加方便组织和管理，并且扩展性也更强。
- 所以，组件是目前小程序开发中，非常重要的一个篇章，要认真学习。

# 创建一个自定义组件

- 类似于页面，**自定义组件由 json wxml wxss js 4个文件组成。**
  - 按照我的个人习惯, 我们会先在根目录下创建一个文件夹 components, 里面存放我们之后自定义的公共组件.
  - 常见一个自定义组件 my-cpn: 包含对应的四个文件.
- **自定义组件的步骤：**
  - 1.首先需要在 json 文件中进行自定义组件声明（将 component 字段设为 true 可这一组文件设为自定义组件）：
  - 2.在wxml中编写属于我们组件自己的模板
  - 3.在wxss中编写属于我们组件自己的相关样式
  - 4.在js文件中, 可以定义数据或组件内部的相关逻辑(后续我们再使用)

index.json

```
{
  "component": true,
  "usingComponents": {}
}
```

index.wxml

```
<!--components/my-cpn/index.wxml-->
<view>
  <view class="title">我是组件标题</view>
  <text class="content" id="info">我是组件的内容，哈哈哈</text>
</view>

```

index.wxss

```
/* components/my-cpn/index.wxss */
.title {
  font-size: 48rpx;
  font-weight: 700;
}

.content {
  text-decoration: underline;
}
```

index.js

```
// components/my-cpn/index.js
Component({
  data: {

  },
  methods: {

  }
})
```

# 使用自定义组件和细节注意事项

- 一些需要注意的细节：
  - 因为 WXML 节点标签名只能是 小写字母、中划线和下划线 的组合，所以自定义组件的标签名也只能包含这些字符。
  - 自定义组件也是可以引用自定义组件的，引用方法类似于页面引用自定义组件的方式（使用 usingComponents 字段）。
  - 自定义组件和页面所在项目根目录名 不能以“wx-”为前缀，否则会报错。
  - 如果在app.json的usingComponents声明某个组件，那么所有页面和组件可以直接使用该组件。

app.json

```
{
  "pages": [
    "pages/home/home",
    "pages/about/about"
  ],
  "usingComponents": {
    "my-cpn": "/components/my-cpn/my-cpn"
  },
```

about.wxml

```
<!--pages/about/about.wxml-->
<text>pages/about/about.wxml</text>
<my-cpn/>
```

# 组件的样式细节

- **课题一：** 组件内的样式 对 外部样式 的影响
  - 结论一：组件内的class样式，只对组件wxml内的节点生效, 对于引用组件的Page页面不生效。
  - 结论二：组件内不能使用id选择器、属性选择器、标签选择器
- **课题二：** 外部样式 对 组件内样式 的影响
  - 结论一：外部使用class的样式，只对外部wxml的class生效，对组件内是不生效的
  - 结论二：外部使用了id选择器、属性选择器不会对组件内产生影响
  - 结论三：外部使用了标签选择器，会对组件内产生影响

- **整体结论:**
  - 组件内的class样式和组件外的class样式, 默认是有一个隔离效果的；
  - 为了防止样式的错乱，官方不推荐使用id、属性、标签选择器；

# 样式的相互影响

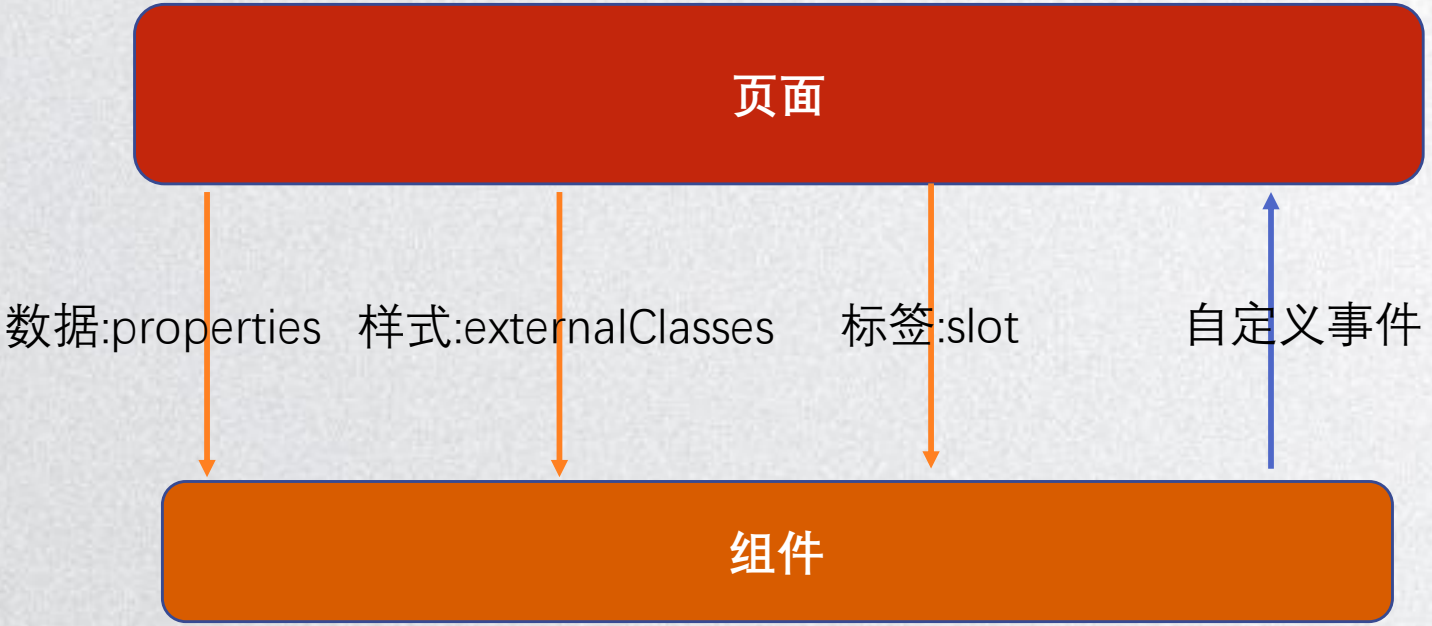
- **课题三：**如何让class可以相互影响
- 在Component对象中，可以传入一个options属性，其中options属性中有一个styleIsolation（隔离）属性。styleIsolation有三个取值：
  - - isolated 表示启用样式隔离，在自定义组件内外，使用 class 指定的样式将不会相互影响（默认取值）；
  - - apply-shared 表示页面 wxss 样式将影响到自定义组件，但自定义组件 wxss 中指定的样式不会影响页面；
  - - shared 表示页面 wxss 样式将影响到自定义组件，自定义组件 wxss 中指定的样式也会影响页面和其他设置了

```
Component({
  options: {
    styleIsolation: "isolated"
  }
})
```

- 其他一些相关样式细节，参考官网：https://developers.weixin.qq.com/miniprogram/dev/framework/custom-component/wxml-wxss.html

# 组件和页面通信

- 很多情况下，组件内展示的内容（数据、样式、标签），并不是在组件内写死的，而且可以由使用者来决定。



# 向组件传递数据 - properties

- **给组件传递数据：**
  - 大部分情况下，组件只负责布局和样式，内容是由使用组件的对象决定的。
  - 所以，我们经常需要从外部传递数据给我们的组件，让我们的组件来进行展示。如何传递呢？
  - **使用properties属性：**
- 支持的类型：
  - String、Number、Boolean
  - Object、Array、null（不限制类型）

my-props.wxml

```
<!--components/my-props/my-props.wxml-->
<view>
  <view>title:{{title}}</view>
  <view>content:{{content}}</view>
  <view>counter:{{counter}}</view>
</view>
```

```
Component({
  properties: {
    title: String,
    content: {
      type: String,
      value: ''
    },
    counter: {
      type: Number,
      value: 0,
      observer: function (newVal, oldVal) {
        console.log(newVal, oldVal)
      }
    }
  }
})
```

```
<my-props title="标题"
          content="内容,哈哈哈"
          counter="{{123}}"/>
```

```
title:标题
content:内容,哈哈哈
counter:123
```

# 向组件传递样式 - externalClasses

- **给组件传递样式：**
  - 有时候，我们不希望将样式在组件内固定不变，而是外部可以决定样式。
  - 这个时候，我们可以使用externalClasses属性：
    - 1.在Component对象中，定义externalClasses属性
    - 2.在组件内的wxml中使用externalClasses属性中的class
    - 3.在页面中传入对应的class，并且给这个class设置样式

my-extern.wxml

```
<!--components/my-extern/my-extern.wxml-->
<view>
  <view class='title'>我是标题</view>
  <view class='content'>我是内容,哈哈哈</view>
</view>
```

my-extern.wxml / my-extern.js

```
// components/my-extern/my-extern.js
Component({
  externalClasses: ['title', 'content']
})
```

external.wxml

```
<my-extern title="title" content="content"/>
```

external.wxss

```
.title {
  font-size: 48rpx;
  color: red;
}

.content {
  color: purple;
}
```

# 组件向外传递事件 – 自定义事件

- 有时候是自定义组件内部发生了事件，需要告知使用者，这个时候可以使用自定义事件：

event-cpn.wxml

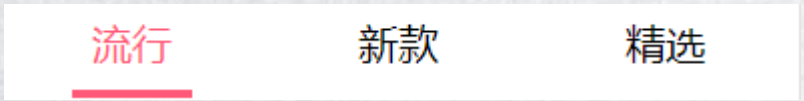
```
<!--components/event-cpn/event-cpn.wxml-->
<view class='event-cpn'>
  <view wx:for="{{titles}}"
        wx:key="{{index}}"
        class="item, {{currentIndex === index ? 'active': ''}}"
        bind:tap="titleClick"
        data-index="{{index}}">
    {{item}}
  </view>
</view>
```

```
/**
 * 组件的方法列表
 */
methods: {
  titleClick(e) {
    const index = e.target.dataset.index;
    this.setData({
      currentIndex: index
    })
    this.triggerEvent('titleclick', {index}, {})
  },
  increment() {
    this.setData({
      counter: ++this.data.counter
    })
  }
}
```

```
<text>pages/event/event.wxml</text>
<event-cpn
    titles="{{['商品', '新闻', '消息']}}"
    id="event-cpn"
    bind:titleclick="titleclick"></event-cpn>
<text>{{infos[currentIndex]}}</text>
```

```
titleclick(e) {
  this.setData({
    currentIndex: e.detail.index
  })
},
```

# 自定义组件练习




w-tab-control.wxml

```wxml
<!--components/w-tab-control/w-tab-control.wxml-->
<view class='tab-control'>
  <block wx:for="{{titles}}" wx:key="index">
    <view class='tab-item {{currentIndex == index ? "active": ""}}'
          bindtap='onItemClick'
          data-index="{{index}}">
      <text>{{item}}</text>
    </view>
  </block>
</view>
```

w-tab-control.js

```js
// components/w-tab-control/w-tab-control.js
Component({
  properties: {
    titles: {
      type: Array,
      value: []
    }
  },
  data: {
    currentIndex: 0
  },
  methods: {
    onItemClick(event) {
      // 1.获取传入的index
      const index = event.currentTarget.dataset.index;

      // 2.改变记录的currentIndex
      this.setData({
        currentIndex: index
      })

      // 3.发出自定义事件
      this.triggerEvent('titleclick', {index}, {})
    }
```

# 页面直接调用组件方法

- this.selectComponent

监听页面中的点击:

```
onClick(e) {
  console.log('按钮被点击')
  const eventCpn = this.selectComponent('#event-cpn')
  eventCpn.increment()
}
```

组件内的监听:

```
increment() {
  this.setData({
    counter: ++this.data.counter
  })
}
```

# 什么是插槽

- **slot翻译为插槽：**
  - 在生活中很多地方都有插槽，电脑的USB插槽，插板当中的电源插槽。
  - 插槽的目的是让我们原来的设备具备更多的扩展性。
  - 比如电脑的USB我们可以插入U盘、硬盘、手机、音响、键盘、鼠标等等。
- **组件的插槽：**
  - 组件的插槽也是为了让我们封装的组件更加具有扩展性。
  - 让使用者可以决定组件内部的一些内容到底展示什么。
- **栗子：移动网站中的导航栏。**
  - 移动开发中，几乎每个页面都有导航栏。
  - 导航栏我们必然会封装成一个插件，比如nav-bar组件。
  - 一旦有了这个组件，我们就可以在多个页面中复用了。
- 但是，每个页面的导航是一样的吗？类似右图

# 单个插槽的使用

- 除了内容和样式可能由外界决定之外，也可能外界想决定显示的方式
  - 比如我们有一个组件定义了头部和尾部，但是中间的内容可能是一段文字，也可能是一张图片，或者是一个进度条。
  - 在不确定外界想插入什么其他组件的前提下，我们可以在组件内预留插槽：

slot-cpn.wxml

```wxml
<!--components/slot-cpn/slot-cpn.wxml-->
<view class='slot-cpn'>
  <view class='header'>我是头部</view>
  <slot></slot>
  <view class='footer'>我是尾部</view>
</view>
```

```wxml
<text>pages/slot/slot.wxml</text>
<!-- 1.插入一段文字 -->
<slot-cpn>
  <text>我是一段文字，哈哈哈</text>
</slot-cpn>

<!-- 2.插入一个按钮 -->
<slot-cpn>
  <button>我是按钮</button>
</slot-cpn>

<!-- 3.插入一张图片 -->
<slot-cpn>
  <image src="https://pic4.zhimg.com/80/v2-667c8450739e38d899d7593ebcfd7c78_qhd.jpg"/>
</slot-cpn>
```

# 多个插槽的使用

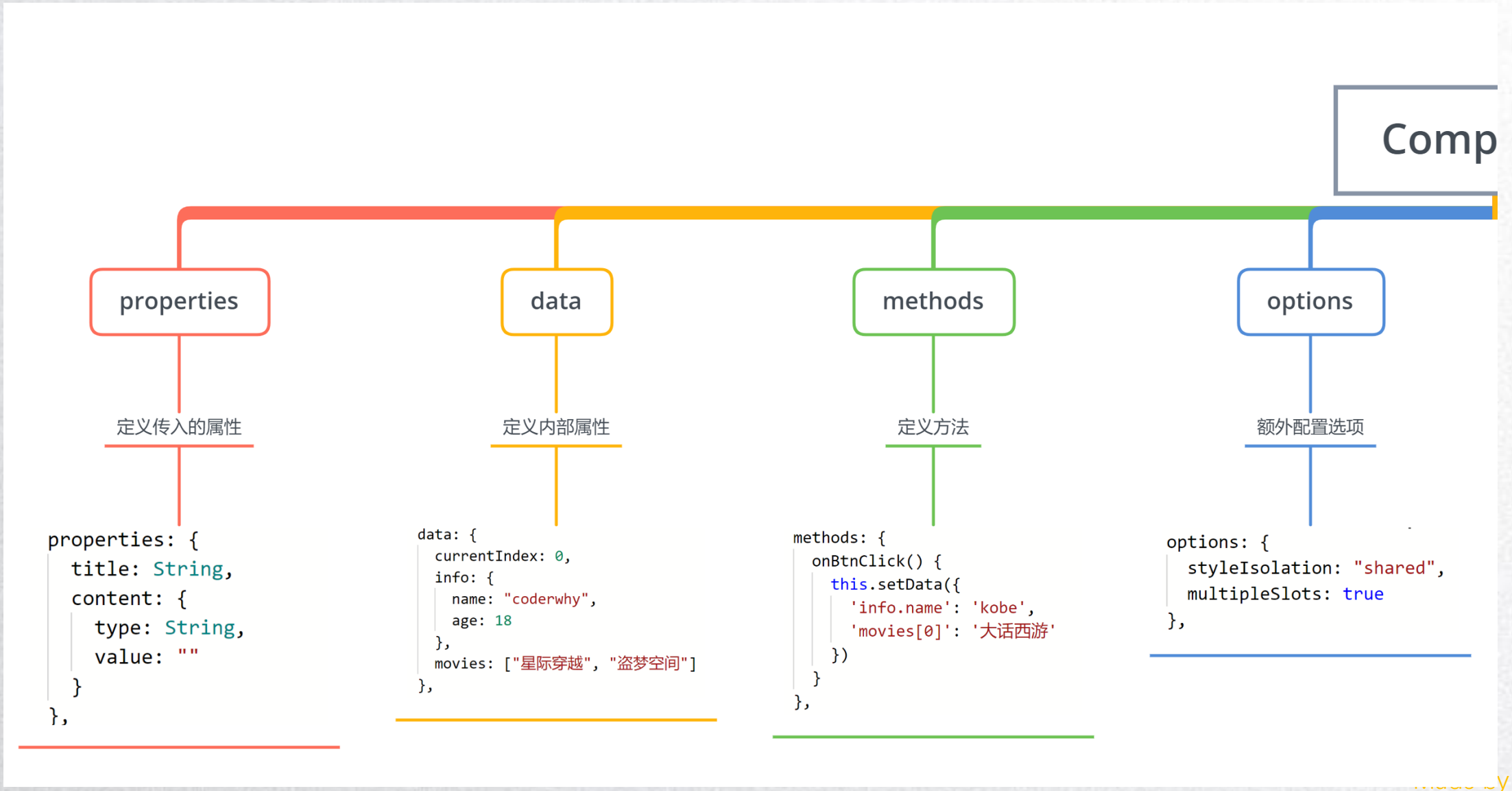
- 有时候为了让组件更加灵活, 我们需要定义多个插槽:

```
<view class='mslot-cpn'>
  <view class='left item'><slot name="left"></slot></view>
  <view class='center item'><slot name="center"></slot></view>
  <view class='right item'><slot name="right"></slot></view>
</view>
```

```
2  Component({
3    options: {
4      multipleSlots: true
5    }
6  })
```

```
<mslot-cpn>
  <text slot="left">哈哈</text>
  <text slot="right">呵呵</text>
  <text slot="center">嘿嘿</text>
</mslot-cpn>
```

# Component构造器



# Component构造器

onent

## externalClasses

引用外部样式

```
externalClasses: ['title'],
```

## observers

属性和数据监听

```
observers: {
  title: function(newVal, oldVal)
    console.log(newVal, oldVal)
  },
  "info.name": function(newVal) {
    console.log(newVal)
  },
  "movies[0]": function(newVal) {
    console.log(newVal)
  }
},
```

## pageLifetimes

页面生命周期

```
pageLifetimes: {
  show() {
    console.log('页面显示出来')
  },
  hide() {
    console.log('页面隐藏起来')
  },
  resize() {
    console.log('尺寸发生改变')
  }
},
```

## lefetimes

组件生命周期

```
lifetimes: {
  created() {
    console.log('组件被创建')
  },
  attached() {
    console.log('组件被添加到页面中')
  },
  ready() {
    console.log('组件被渲染出来')
  },
  movied() {
    console.log('组件被移动到节点树另一个位置')
  },
  detached() {
    console.log('组件被移除掉')
  }
}
```

# Component构造器

- Component构造器用户创建我们的自定义组件对象, 调用Component时, 可以传入属性、数据、方法等

```
// 属性: 使用者(页面/组件)传递数据到这里
properties: {

},

// 数据: 定义内部的数据(写死/读取文件/网络请求)
data: {
  counter: 0,
  info: {
    name: 'why',
    age: 18
  },
  nums:[10, 20, 33]
},

// 方法: 调用方法/监听事件的方法
methods: {
  onIncrement(e) {
    this.setData({
      counter: ++this.data.counter,
      'info.age': 100,
      'nums[2]': 123
    })
  }
},
```

```
// behaviors: 类似于其他框架中的mixins
// 观察者:监听属性的变化
observers: {
  "counter": function(newValue) {
    console.log(newValue)
  },
  'info.age': function(newAge) {
    console.log(newAge)
  },
  'nums[2]': function(newNum) {
    console.log(newNum)
  }
},

// options: 一些特殊选项
options: {
  styleIsolation: 'shared', // 样式的隔离方式
  multipleSlots: true, // 组件内包含多个插槽
},

// 引用外部的class
externalClasses: []
```

```
lifetimes: {
  attached() {
    console.log('被添加到页面或者其他组件上')
  },
  moved() {
    console.log('组件被移动到其他位置')
  },
  detached() {
    console.log('被从其他页面或者组件中移除')
  }
},
pageLifetimes: {
  show() {
    console.log('页面显示出来')
  },
  hide() {
    console.log('页面隐藏起来')
  },
  resize() {
    console.log('改变大小')
  }
}
```