## Vue函数化组件

VueJS提供了一个functional的布尔值选项，设置为true可以使组件无状态和无实例，无状态**无实例也就是指没有data和this上下文。**

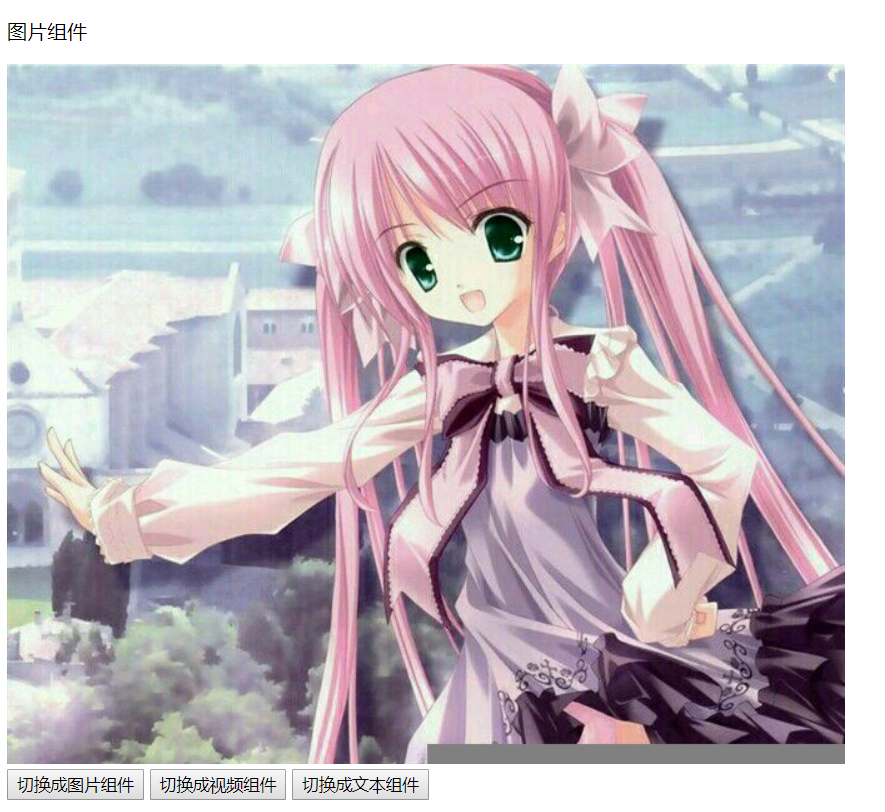
这样使用render函数返回虚拟节点可以更容易的渲染，因为函数化组件只是一个函数，渲染开销也小很多。

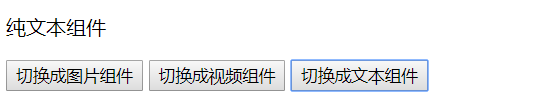
使用函数化组件的时候，render函数提供了一个第二个参数context（第一个参数是createElement）。组件需要的data、props、slots、children、parent都是通过上下文来进行传递的（我的理解是组件内并没有data，值都是props来的），因为不能使用this上下文,也就是说 this.level就得换成context.props.level。This.$slots.default也得换成context.children。

但是函数化组件并不常用，抽象麻烦，但是理解会用必然是好事，而且有的时候用过函数化组件可能更为方便。常用的方便情况比如说：

1. 函数化地在多个组件中选择一个（或是说在多个组件中选出一部分进行操作）。
2. 再将children,props,data传递给子组件之前操作他们。（这句话我并没有理解）。

下面记录一个写过的小例子，实现一个点击不同按钮变换不同组件的智能组件。





HTML：

<div *id*="app">

<smart-item *:data* = "data"></smart-item>

<button *@click* = "change('img')">切换成图片组件</button>

<button *@click* = "change('video')">切换成视频组件</button>

<button *@click* = "change('text')">切换成文本组件</button>

</div>

Vue

var img = {

props:['data'],

render:function(*createElement*){

return createElement(

"div",

[

createElement("P","图片组件"),

createElement("img",{

attrs:{

src:*this*.data.url

}

})

]

);

}

}

var video = {

props:['data'],

render:function(*createElement*){

return createElement(

"div",

[

createElement("p","视频组件"),

createElement(

"video",

{

attrs:{

src : *this*.data.url,

controls:"controls",

autoplay:"autoplay"

}

}

)

]

);

}

}

var TextItem = {

props:['data'],

render:function(*createElement*){

return createElement(

"div",

[

createElement("p",'纯文本组件'),

createElement("p",*this*.data.text)

]

);

}

}

Vue.component("smart-item",{

functional:true,

props:{

data:{

type:*Object*,

required:true

}

},

render:function(*createElement*,*context*){

function getComponent(){

var data = context.props.data;

if(data.type === "img") return img;

if(data.type === "video") return video;

return TextItem;

}

return createElement(

getComponent(),

{

props:{

data:context.props.data

},

},

context.children

);

}

});

var app = **new** *Vue*({

el:"#app",

data:{

data:"请选择"

},

methods:{

change(*type*){

if(type === "img"){

*this*.data = {

type:'img',

url:"images/Fxm.jpg"

}

} else if(type === "video") {

*this*.data = {

type : "video",

url:"http://vjs.zencdn.net/v/oceans.mp4"

}

} else if(type === "text"){

*this*.data = {

type:'text',

content:"这是一段文本"

}

}

}

},

created:function(){

*this*.change('img');

}

})

这个例子来自于书上，但是最开始看的时候有很多地方没看懂，后来多琢么琢么想通了。

首先HTML结构没有难度，想法是让root实例通过点击按钮将选择的组件信息通知给smart组件。然后smart组件在“变身”成为那个对应的组件。

① 点击按钮，root执行change函数，改变内部的data值，之后将值以props的形式传递给smart。

② 在smart变身之前，我们需要将smart的替身准备好，这里有图片组件，视频组件，纯文本组件，替身本质是个vnode组件，替身组件还需要知道自己的功能需要的必须信息，比如图片组件要图片地址，视频组件要视频框上的功能键和视频地址。因此各个替身组件需要从父组件那里知道这些信息。因此他们都需要一个props：data用来存放这些信息。

③替身只是smart的道具，需要的时候穿在身上完成“变身”。因此有必要强调一下，其实真正接受root实例的传来的信息的应该是smart组件，而不是“替身”。

具体的流程为：

root做prop传值给smart --->smart做prop传值将root传来的值给替身--->替身组件根据传来的信息进行操作。

因此在smart的声明中需要一个props部分来接受root传来的data。

props:{

data:{

type:*Object*,

required:true

}

}

因为替身也需要从smart那里接值，因此也需要一个props。

var img = {

props:['data'], // 就是这里，注意这里只是声明一个需要准备接值的data

render:function(*createElement*){

return createElement(

"div",

[

createElement("P","图片组件"),

createElement("img",{

attrs:{

src:*this*.data.url

}

})

]

);

}

}

④然后在smart变身的时候为替身传值

render:function(*createElement*,*context*){

function getComponent(){

var data = context.props.data;

if(data.type === "img") return img;

if(data.type === "video") return video;

return TextItem;

}

return createElement(

getComponent(),

{

props:{

data:context.props.data // 这一步相当于在替身上:data = “data”

},

},

context.children

);

}

⑤ 主要过程结束。。

之前之所以懵住，主要是对createElement的使用方法理解不清。

createElement的第一个参数可以是String、Object、Function。

如果是String的话，就是直接值元素标签，生成一个对应标签的元素。

如果是Object的话，这个Object应该是一个包含组件信息的对象，createElement函数就会形成一个和这个Object修饰的组件一样的实体DOM。

第二个参数是数据对象。其中值得注意的是，attrs里的东西是DOM特性，特性即可以写在标签中的属性。domProps即普通的DOM属性，DOM属性大部分是不能出现在标签值中的，但是也有例外，《Js高级程序设计》说过，像是id,class,align等公认度较强的是”脚踏两只船”。即使属性也是特性。自定义的特性不会成为属性。（具体之后JS高程笔记会补充）

向dom标签中的 :dada = “data” 或者@click就不能按照属性或者特性处理，他们有自己的地方，比如props； on；

顺便一提：那个替身是一个含有组件信息的对象，并不是真正已经注册好的组件，也就是说createElemen的第一个参数需要的是一个包含组件信息的对象，而不是一个切实存在的组件。另外，那个组件对象的模板可以不用render函数，也可以有date和this的上下文以及生命周期等。