有使用过vuex（状态管理）吗？请描述一下

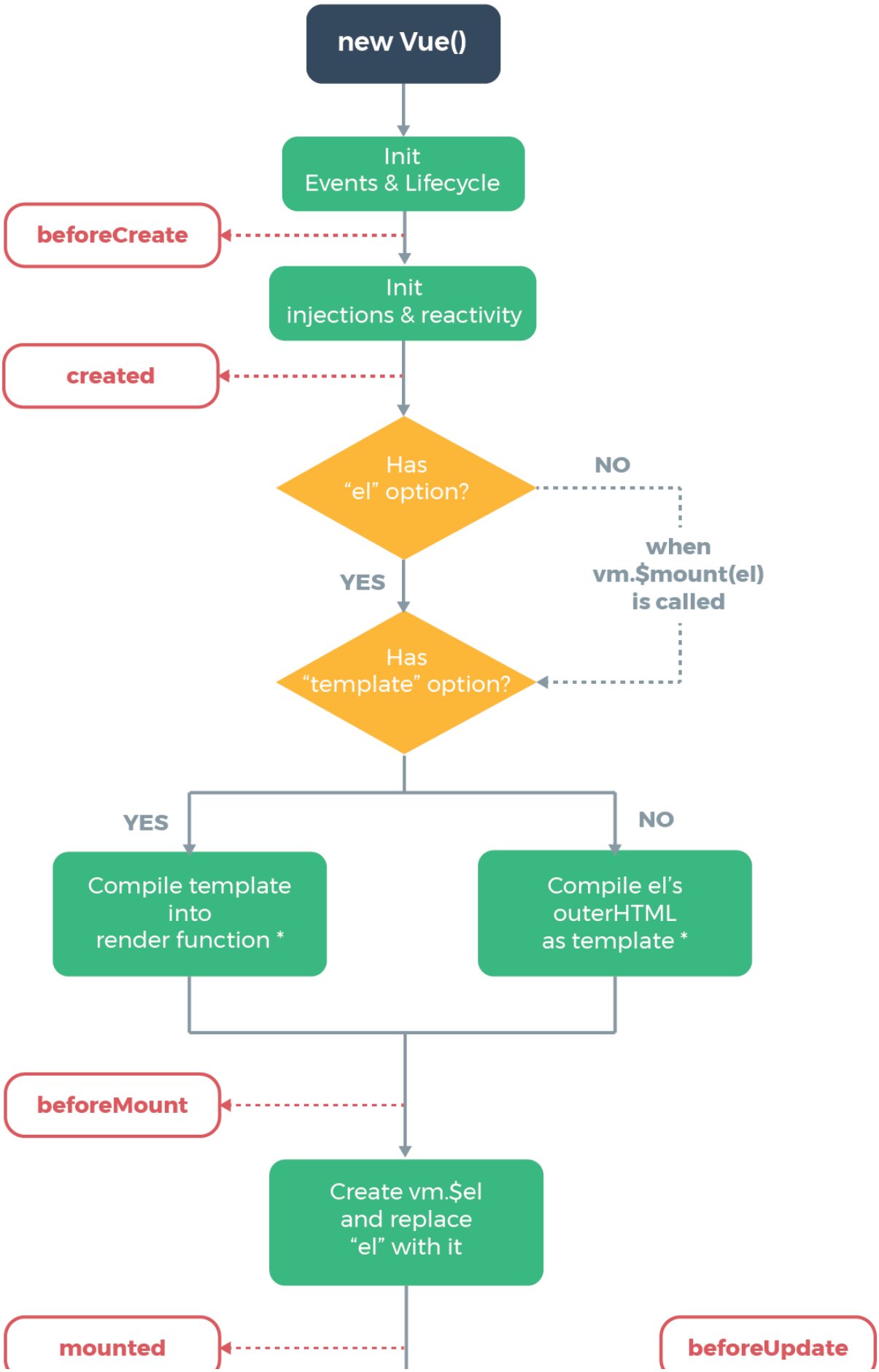
- 状态管理用于组件之间的数据共享，以store作为容器来管理
- State：用来存储共享的数据
- Getter：类似计算属性，同来获取经过处理后的数据，具有缓存作用
- Mutation：同步提交的更改
- Action：异步提交状态的更改
- Module：当状态管理多了，使用Module来划分多个模块管理

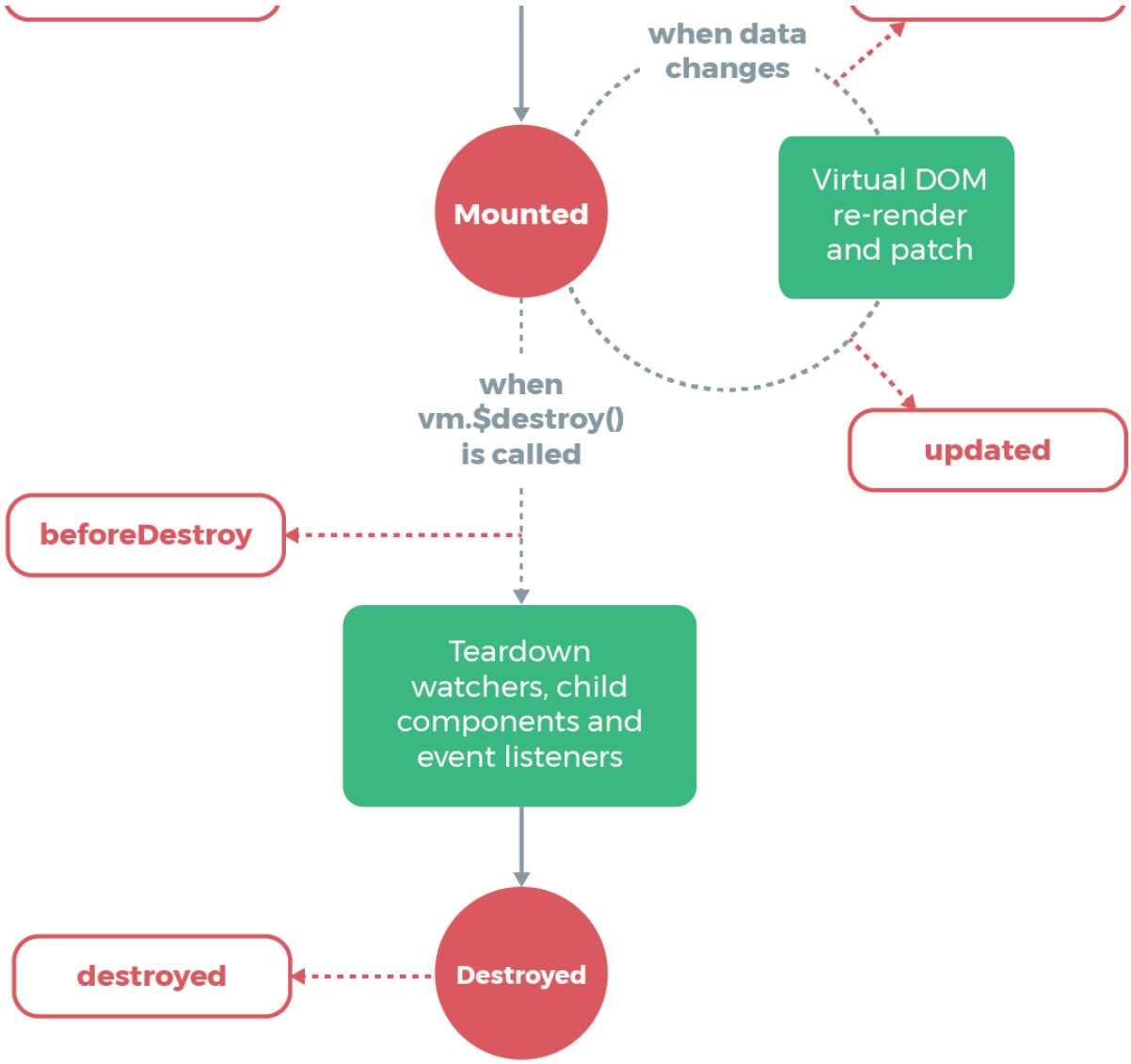
请描述一下Vue声明周期

- vue实例从创建到想回主要有8个阶段
- 创建前，创建后，渲染前，渲染后，更新前，更新后，销毁前，销毁后。

  使用new Vue（）创建vue的实例，然后调用Init函数初始化对象信息，在

  渲染前会检查是否有el属性，如果没有就调用$mount（el）方法；然后检查是否有template属性，如果有编译template的内容进行渲染，如果没有，就编译el指定的节点内容进行渲染，然后创建虚拟DOM$el对象，并用替换挂载节点的内容渲染到视图上。在渲染后和更新后就可以dom操作


new Vue()
Init
Events & Lifecycle
beforeCreate
Init
injections & reactivity
created
Has
“el” option?
NO
when
vm.$mount(el)
is called
YES
Has
“template” option?
YES
NO
Compile template
into
render function *
Compile el’s
outerHTML
as template *
beforeMount
Create vm.$el
and replace
“el” with it
mounted
beforeUpdate



* template compilation is performed ahead-of-time if using a build step, e.g. single-file components

组件传值的方式有哪些

父传子：在子组件使用props定义属性，父组件通过绑定定义的属性来传递

子传父：通过使用$emit自定义时间来传值，在子组件使用$emit自定义事件，在父组件绑定事件来获取子组件传递过来的值

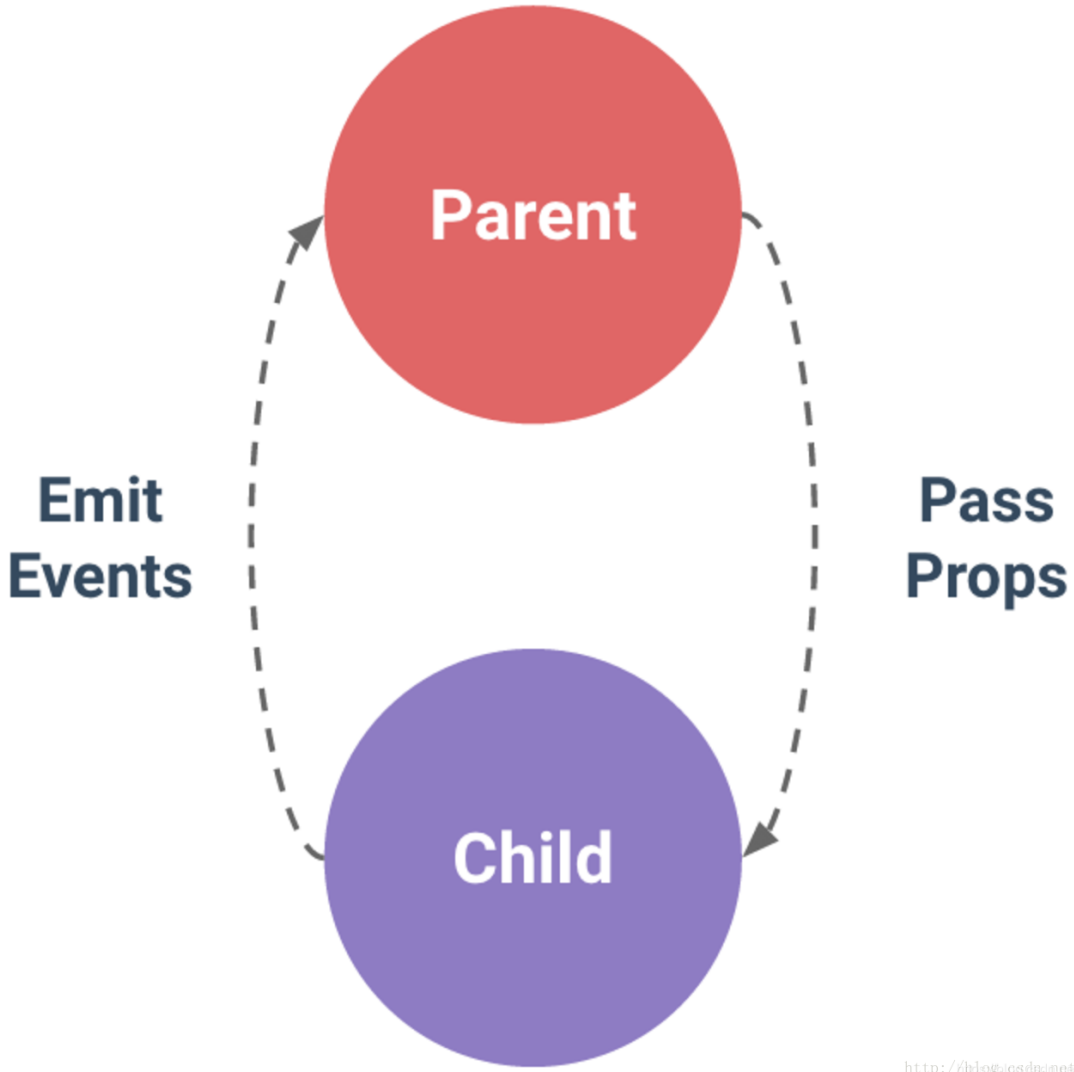
组件与组件之间：使用状态管理

子向父组件通信（vuebus）（扩展）

- 通过new Vue()这样的一个对象，来$on('事件名',fn(prop1,pro2))

- 另一个组件引入同一个vuebus, 来$emit('事件名',prop1,pro2)

在使用vue的过程中免不了要组件之间传值，那么这里就说一说传值的几种方法和怎么使用 先来看一张图 引用官网的一句话：**父子组件的关系可以总结为 prop 向下传递，事件向上传递。父组件通过 prop 给子组件下发数据，子组件通过事件给父组件发送消息**，如下图所示：



一.父组件向子组件传值 来，先上代码 **父组件:**

```
<template>
  <div>
    父组件:
    <input type="text" v-model="name">      //v-model绑定
    <br>
    <br>
    <!-- 引入子组件 -->
    <child :inputName="name"></child>
  </div>
</template>
<script>
      import child from './child'       //引入
      export default {
        components: {
          child
```

```
        },
        data () {
          return {
            name: ''
          }
        }
      }
</script>
12345678910111213141516171819202122 23
```

子组件：

```
<template>
  <div>
    子组件:
    <span>{{inputName}}</span>
  </div>
</template>
<script>
  export default {
    // 接受父组件的值
    props: {
      inputName: String,
      required: true
    }
  }
</script>
1234567891011121314 15
```

二.子组件向父组件传值

在子组件中创建一个按钮，并绑定一个点击事件

```
<template>
    <div>
        <li v-for="(item,index) in arr" >
            <p>{{item}}</p>
            <button @click="chuanzhi(index)">删除</button>
        </li>
    </div>
</template>
12345678
```

在js的methods中定义这个事件函数中使用$emit来触发一个事件并传递一个参数。

```
chuanzhi(index){
    this.$emit("jieshou",index)
}
123
```

在父组件中v-on监听这个事件（jieshou）并添加一个响应事件的方法

```
<template>
  <div class="hello">
    <router-link to="/6">426636</router-link>
        <h1>{{ msg }}</h1>
    <input type="text" v-model="txt">
    <button @click="dianji">留言</button>
    <Show v-bind:arr="arr" v-on:jieshou="xuanran"></Show>
  </div>
</template>
123456789
```

js中

```
xuanran(data){
    console.log(data)
    this.arr.splice(data,1)
}
1234
```

三.同级组件之间的传值(创建一个公共js文件) 这里我们先说一下如何创建一个公共文件去传值，下一篇会说到如何使用Vuex状态管理器来建立一个仓库进行组件之间的传值。

```
非父子组件之间传值，需要定义个公共的公共实例文件bus.js，作为中间仓库来传值，不然路由组件之间达不到传值的效果。
1、在工程src目录下新建一个Pub.js文件
Pub.js
    import Vue from "vue"
    export default new Vue()
2、在各组件中引入Pub.js文件
    import Pub from "../Pub.js"
3、在A组件中通过$emit调用自定义函数并且传参
    Pub.$emit('namefn',"str1","str2", ……)
4、在B组件中通过$on响应事件并接受参数
    Pub.$on('namefn',(str1,str2,……) =>{
        console.log(str1+"就是A组件中定义的数据")
    })
12345678910111213
```

示例代码： 公共bus.js

```
//bus.js
import Vue from 'vue'        //移入Vue
export default new Vue()         //暴露给Vue这个全局对象
123
```

组件A中

```
<template>
  <div>
    A组件:
```

```
    <span>{{elementValue}}</span>
    <input type="button" value="点击触发" @click="elementByValue">
  </div>
</template>
<script>
  // 引入公共的bug，来做为中间传达的工具
  import Bus from './bus.js'
  export default {
    data () {
      return {
        elementValue: 4
      }
    },
    methods: {
      elementByValue: function () {
        Bus.$emit('val', this.elementValue)
      }
    }
  }
</script>
12345678910111213141516171819202122 23
```

组件B中：

```
<template>
  <div>
    B组件:
    <input type="button" value="点击触发" @click="getData">
    <span>{{name}}</span>
  </div>
</template>
<script>
  import Bus from './bus.js'
  export default {
    data () {
      return {
        name: 0
      }
    },
    mounted: function () {
      var vm = this
      // 用$on事件来接收参数
      Bus.$on('val', (data) => {
        console.log(data)
        vm.name = data
      })
    },
    methods: {
      getData: function () {
        this.name++
      }
    }
```

```
  }
</script>
```

了解路由传参吗？怎么实现的?

列表:

- {{hero.name}} http://localhost:8080/?id=0 查看

path列表

http://localhost:8080/detail/1

- {{hero.name}} 查看

```
//以下规则 /detail?xxx = xx & xxx=xx 多少个查询字符串都不影响
      path: '/detail/:id',
      name: 'detail',
      component: Detail,

mounted(){

        //获取路由参数

        //vue-router中挂载连个对象的属性

        //$route()  $router(功能函数)

        console.log(this.$route.params.id)

        console.log(this.$route.query.id)

    }//已经将数据装载在到页面上去了，DOM，已经生成
```

有使用过Promise吗？用来干什么的

```
Promise用于异步处理一个代理对象有三个状态
1.pending初始状态
2.fulfilled操作成功
3.rejected操作失败
使用then捕捉成功的结果，使用catch捕捉失败的结果；构造函数的参数是一个函数，第一个参数是resolve函数用于返回成功的结果，第二个参数reject函数，用于返回错误的信息
```

双向绑定的原理是什么?

```
v-model用于<input>,<textarea>及<select>双向绑定数据，方便获取用户输入的内容，其实是一个语法糖，使用value或checked属性绑定要渲染的值，使用input时间或change获取用户的输入的值给绑定的变量。

对于text和taxtarea使用value属性和input事件
对于radio和checkbox使用checked属性和change事件
对于select使用value属性和change事件
```

路由底层实现原理是什么

路由有两种方式实现，hash模式和history模式，hash模式利用URL#后面的内容发生改变不会刷新页面的特性来实现的。history模式是利用了H5的新特性，使用history.pushState()改变url路径，但页面不会重新加载来实现的

Vue插槽，是学习vue中必不可少的一节，当初刚接触vue的时候，对这些掌握的一知半解，特别是作用域插槽一直没明白。

后面越来越发现插槽的好用。

分享一下插槽的一些知识吧。

插槽使用场景是；如果子组件需要显示的内容并非来自本身，而是父组件传递进来的，而假如直接这样写，是不可以实现的，因为如果me-component没有包含一个 元素，则任何传入它的内容都会被抛弃：

分一下几点：

1、插槽内可以放置什么内容?

2、默认插槽

3、具名插槽

4、作用域插槽

一、插槽内容

一句话：插槽内可以是任意内容。

先看一下下面的代码：声明一个child-component组件，

如果现在我想在内放置一些内容，结果会是怎样?

```
<div id="app">
    <child-component></child-component>

</div>
<script>
    Vue.component('child-component',{
        template:`
            <div>Hello,World!</div>
        `
    })
    let vm = new Vue({
        el:'#app',
        data:{

        }
    })
</script>
```

```
<child-component>你好</child-component>
```

输出内容还是在组件中的内容，在 内写的内容没起作用。

这就是插槽出现的作用。

我们现在给组件增加一个插槽

我们在内写的"你好"起作用了！！！

```
Vue.component('child-component',{
        template:`
            <div>
            Hello,World!
            <slot></slot>
            </div>
        `
    })
```

到现在，我们知道了什么是插槽：

插槽就是Vue实现的一套内容分发的API，将元素作为承载分发内容的出口。

这句话的意思就是，没有插槽的情况下在组件标签内些一些内容是不起任何作用的，当我在组件中声明了slot元素后，在组件元素内写的内容

就会跑到它这里了！

二、具名插槽

具名插槽，就是给这个插槽起个名字

在组件中，我给插槽起个名字，一个名字叫"girl"，一个名字叫"boy"，还有一个不起名字。

然后再内，slot属性对应的内容都会和组件中name一一对应。

而没有名字的，就是默认插槽！！

```
<div id="app">
    <child-component>
        <template slot="girl">
            漂亮、美丽、购物、逛街
        </template>
        <template slot="boy">
            帅气、才实
        </template>
        <div>
            我是一类人，
            我是默认的插槽
        </div>
    </child-component>
</div>
<script>
    Vue.component('child-component',{
        template:`
            <div>
            <h4>这个世界不仅有男人和女人</h4>

            <slot name="girl"></slot>

            <div style="height:1px;background-color:red;"></div>

            <slot name="boy"></slot>

            <div style="height:1px;background-color:red;"></div>

            <slot></slot>
            </div>
        `
    })
    let vm = new Vue({
        el:'#app',
        data:{

        }
    })
</script>
```

三、默认插槽

上面已经介绍过了，这里不做描述

四、作用域插槽

之前一直没搞懂作用域插槽到底是什么！！！

说白了就是我在组件上的属性，可以在组件元素内使用！

先看一个最简单的例子！！

我们给元素上定义一个属性say（随便定义的！），接下来在使用组件child，然后在template元素上添加属性slot-scope！！随便起个名字a

我们把a打印一下发现是 {"say" : "你好"}，也就是slot上面的属性和值组成的键值对！！！

这就是作用域插槽！

我可以把组件上的属性/值，在组件元素上使用！！

```
<div id="app">
    <child>
        <template slot-scope="a">
          <!-- {"say":"你好"} -->

            {{a}}
        </template>
    </child>
</div>
<script>
    Vue.component('child',{
        template:`
            <div>
                <slot say="你好"></slot>
            </div>
        `
    })

    let vm = new Vue({
        el:'#app',
        data:{

        }
    })
</script>
```

再看一下下面的例子：

```
<div id="app">
    <child :lists="nameList">
        <template slot-scope="a">
            {{a}}
        </template>
    </child>
```

```
</div>
<script>
    Vue.component('child',{
        props:['lists'],
        template:`
            <div>
                <ul>
                    <li v-for="list in lists">
                        <slot :bbbbb="list"></slot>
                    </li>
                </ul>
            </div>
        `
    })

    let vm = new Vue({
        el:'#app',
        data:{
            nameList:[
            {id:1,name:'孙悟空'},
            {id:2,name:'猪八戒'},
            {id:3,name:'沙和尚'},
            {id:4,name:'唐僧'},
            {id:5,name:'小白龙'},
            ]
        }
    })
</script>
```

看一下输出结果

这太有用了兄弟们！！！

这样我就可以在这元素上随便玩了啊！！

当id等于1的时候，我前面加个你好。

我可以随便根据这个对象的属性值进行操作！

```
<child :lists="nameList">
        <template slot-scope="a">
            <div v-if='a.bbbbb.id==1'>你好：<span>{{a.bbbbb.name}}</span></div>
            <div v-else>{{a.bbbbb.name}}</div>
        </template>
    </child>
```

最后！如果用过elementui的同学，一定知道表格就是这样来的！！

表格的本质就是这样！

单个插槽

除非子组件模板包含至少一个 插口，否则父组件的内容将会被丢弃。`当子组件模板只有一个没有属性的插槽时，父组件传入的整个内容片段将插入到插槽所在的 DOM 位置，并替换掉插槽标签本身。`

最初在 标签中的任何内容都被视为备用内容。备用内容在子组件的作用域内编译，并且只有在宿主元素为空，且没有要插入的内容时才显示备用内容。

假定 my-component 组件有如下模板：

```
<div>
  <h2>我是子组件的标题</h2>
  <slot>
    只有在没有要分发的内容时才会显示。
  </slot>
</div>
```

父组件模板：

```
<div>
  <h1>我是父组件的标题</h1>
  <my-component>
    <p>这是一些初始内容</p>
    <p>这是更多的初始内容</p>
  </my-component>
</div>
```

渲染结果：

```
<div>
  <h1>我是父组件的标题</h1>
  <div>
    <h2>我是子组件的标题</h2>
    <p>这是一些初始内容</p>
    <p>这是更多的初始内容</p>
  </div>
</div>
```

具名插槽

场景： 设计组合使用的组件时，内容分发 API 是非常有用的机制

`<slot>` 元素可以用一个特殊的特性 name 来进一步配置如何分发内容。多个插槽可以有不同的名字。具名插槽将匹配内容片段中有对应 slot 特性的元素。

仍然可以有一个匿名插槽，它是默认插槽，作为找不到匹配的内容片段的备用插槽。如果没有默认插槽，这些找不到匹配的内容片段将被抛弃。

例如，假定我们有一个 app-layout 组件，它的模板为：

```
<div class="container">
  <header>
    <slot name="header"></slot>
  </header>
  <main>
    <slot></slot>
  </main>
  <footer>
    <slot name="footer"></slot>
  </footer>
</div>
```

父组件模板：

```
<app-layout>
  <h1 slot="header">这里可能是一个页面标题</h1>

  <p>主要内容的一个段落。</p>
  <p>另一个主要段落。</p>

  <p slot="footer">这里有一些联系信息</p>
</app-layout>
```

渲染结果为：

```
<div class="container">
  <header>
    <h1>这里可能是一个页面标题</h1>
  </header>
  <main>
    <p>主要内容的一个段落。</p>
    <p>另一个主要段落。</p>
  </main>
  <footer>
    <p>这里有一些联系信息</p>
  </footer>
 </div>
```

作用域插槽

> 作用域插槽还不是特别理解。。。。。主要是没想清楚他的应用场景。。。如果有比较理解的可以留言指点一下~ 下面是一个例子但是这种场景我也没太理解这个list有何意义，通用性在哪里？http://blog.csdn.net/oak160/article/details/65447195

> 2.1.0 新增

作用域插槽是一种特殊类型的插槽，用作一个 (能被传递数据的) 可重用模板，来代替已经渲染好的元素。

在子组件中，只需将数据传递到插槽，就像你将 prop 传递给组件一样：

```
<div class="child">
   <slot text="hello from child"></slot>
</div>
```

在父级中，具有特殊特性 slot-scope 的