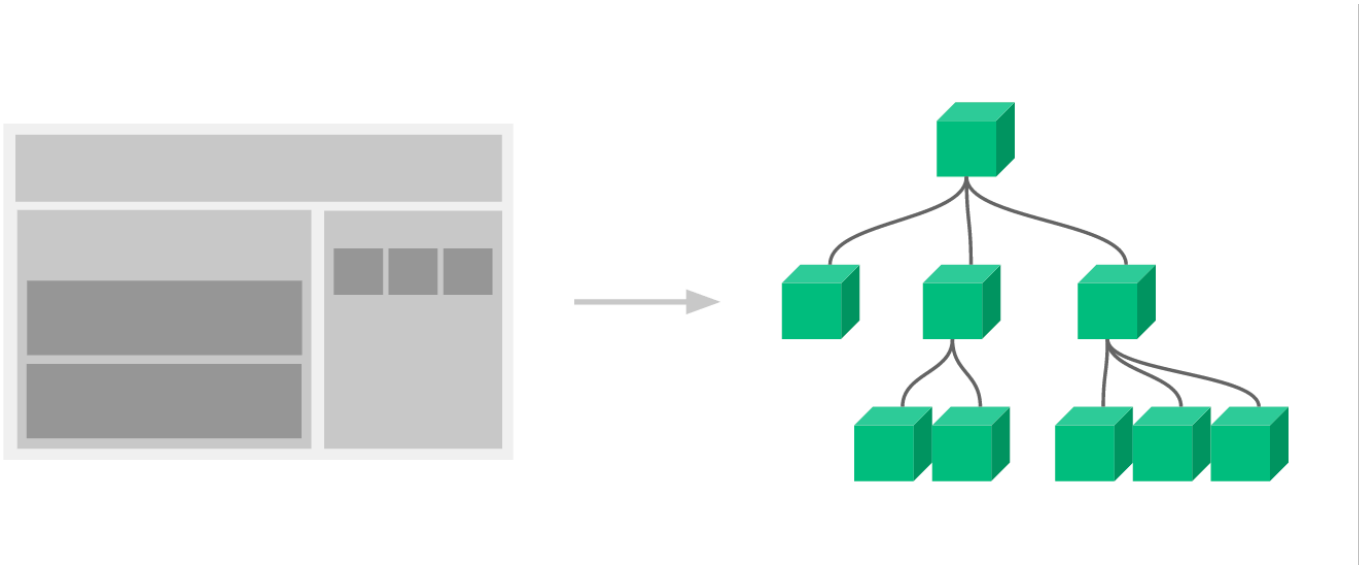
vue 组件化

课堂目标

1. 学习Vue.js 组件化
2. 组件的分类
3. 组件化思想设计
4. 深入了解Vue的组件化机制
5. 动态组件
6. 异步组件
7. 递归组件

知识要点

1. vue
2. 组件化



组件的分类

1. 通用组件
    i. 基础组件，大部分UI库 都是这种组件，比如表单，弹窗，布局等等
2. 业务组件
    i. 业务需求强挂钩 会被复用，比如抽奖，摇一摇等等
3. 页面组件
    i. 每个页面都是组件，不会复用 完成功能

Vue组件开发

1. 注册
2. 使用
3. 给传递什么值 props
4. 组件通知外部 事件
5. 扩展组件 插槽

组件注册

HTML 中的特性名是大小写不敏感的，所以浏览器会把所有大写字符解释为小写字符。这意味着当你使用 DOM 中的模板时，camelCase的 prop 名需要使用其等价的 kebab-case

```
// 全局
Vue.component('kaikeba-comp', {  })
Vue.component('KaikebaComp', {  })
```

使用

```
<div>
    <kaikeba-comp></kaikeba-comp>
    <kaikeba-comp></kaikeba-comp>
    <kaikeba-comp></kaikeba-comp>
</div>
```

局部（推荐，依赖可追溯）

```
new Vue({
  el: '#app',
  components: {
    'kaiekba-comp': KaikebaComp
  }
})
```

Props

使用v-bind:xx(简写:xxq)来传递数据，组件内部props字段接受，使用和挂在在this上的数据没有本质区别

props属于单向数据流，也就是只能通过父级修改，组件自己不能修改 props 的值，只能修改定义在 data 里的数据，非要修改，也是通过后面介绍的自定义事件通知父级，由父级来修改。

事件

组件内部通知外部的变化 `this.$emit`

```
<template>
  <button @click="handleClick">
    开课吧
  </button>
</template>
<script>
  export default {
    methods: {
      handleClick (event) {
        this.$emit('bind-click', event);
      }
    }
```

```
  }
</script>
```

v-model是一个特殊的属性，相当于绑定了:value和@input俩事件

```
<custom-input
    v-model="searchText"

></custom-input>

<custom-input
  :value="searchText"
  @input="searchText = $event"
></custom-input>
```

.

插槽

插槽用来扩展组件的内容

```
  <button @click="handleClick">
    <slot>
        开课吧
    </slot>
  </button>
```

```
    <kaikeba @bind-click="compClick">
      <strong>
        嘿嘿
      </strong>
    </kaikeba>
```

组件设计

表单组件，组件分层

1. Form 负责定义校验规则
2. FormItem 负责显示错误信息
3. k-input 负责数据双向绑定
4. 使用provide和inject内部共享数据

provide & inject

provide 和 inject 主要为高阶插件/组件库提供用例。并不推荐直接用于应用程序代码中。

```
  provide(){
    return {
      form:"kaikeba"
    }
  },

// 子组件注入
var Child = {
  inject: ['form'],
  created () {
    console.log(this.form) // => "kaikeba"
  }
  // ...
}
```

实现v-model

1. 其实就是v-model是一个特殊的属性，相当于绑定了:value和@input俩事件

```
<template>
<!-- 支持v-model -->
  <input
  type="text"
    :value="inputValue"
    @input="handleInput"
    @blur="handleBlur"
  />


</template>

<script>
export default {
```

```
  name:"kInput",
  props:{
    value:{
      type:String,
      default:'',
      required:true
    },
    name:{
      type:String
    }
  },

  data(){
    return {
      inputValue:this.value
    }
  },
  methods:{
    handleInput(e){
      console.log(e)
      const value = e.target.value
      this.inputValue = value
      // 通知父元素修改props
      this.$emit('input',value)
      this.$bus.$emit('kFormItem',{
        value,
        name:this.name})

    },
    handleBlur(){
      const value = this.inputValue
      this.$bus.$emit('kFormItem',{
        value,
        name:this.name
      })
    }
  },
  created(){
    // this.value = 'xx'
  }
}
</script>

<style>

</style>
```

FormItem

1. 获取当前输入框的规则
2. 获取输入框的值 对rule规则进行匹配 过滤不是自己的输入事件
3. 如果输入值和rule不匹配 显示错误信息

```
<template>
 <div>
   <!-- 负责显示一些错误信息 -->
   <label v-if="label">{{label}}</label>
    <div>
      <slot></slot>
      <p v-if="validateStatus=='error'" class='error'>
        {{errorMessage}}
      </p>
    </div>
 </div>
</template>

<script>
export default{
  name:"kFormItem",
  // 可以获取form的实例
  inject:['form'],
  data(){
    return {
      validateStatus:'',
      errorMessage:''
    }
  },
  created(){
    this.$bus.$on('kFormItem',(value)=>{
      this.validate(value)
    })
    // this.getRules()
  },
  methods:{
    getRules(){
      let formRules = this.form.rules[this.prop]
      return formRules
    },
```

```
    // 校验
    validate(obj){
      if(obj.name !== this.prop){
        return
      }
      const rule = this.getRules()
      const value = this.form.model[this.prop]
      console.log(rule,value)
      if(Arrary.isArray(value)){
        value.forEach(v=>{
          if(rule.required && !value){

          }
          if(rule.mixLength && value.length<rule.mixLength){

          }
        })
      }else{

      }
      if(rule.required && !value){
        this.errorMessage = rule.message
        this.validateStatus = 'error'
      }else{
        this.validateStatus = 'validating'
      }
    }
  },
  // watch:{

  // }
  // props:["label","props"],
  props:{
    label:{
      type:String,
    },
    prop:{
      type:String
    }
  }
  // 依赖注入
  // inject:["name"],
  // // props:['name'],
  // name:'formitem'
}
</script>

<style>
```

```
p.error{
  color:red;
}
</style>
```

form

```
<template>
  <form>
    <slot></slot>
    <!-- <hr>
    <button>点击</button>
    <hr>
    <slot name='slot2'>插槽2</slot> -->
  </form>
</template>

<script>
export default{
  provide(){
    return {
      form:this
    }
  },
  name:'form',
  props:{
    model:{
      type:Object
      },
    rules:{type:Object},
  }
  // props:['name']
}
</script>

<style>

</style>
```

组件使用

```
 <k-form :model="form" :rules="rules">
   <k-form-item label="用户名" prop="name">
     <k-input v-model="form.name" name='name'></k-input>
   </k-form-item>
   <k-form-item label="年龄" prop="age">
     <k-input v-model="form.age" name='age'></k-input>
   </k-form-item>

 </k-form>
```

```
data(){
  return{
    form:{
      name:'',
      age:""
    },
    rules:{
      name:[
        {required:true,message:"用户名不能为空"},
        {minLength:3,message:"用户名长度要大于3"},
        {maxLength:10,message:"用户名长度要小于10"},
        ],
      age:{required:true,message:"年龄不能为空"}
    }
  }
}
```