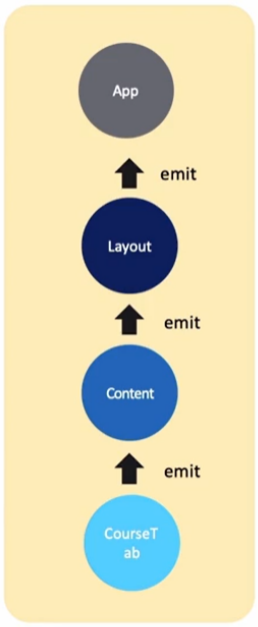
课程目标

组件跨层级访问

子组件如何访问外层组件



Vue是单向数据流，不允许子组件直接修改数据流，但是可以通过emit一层向上传递。如同王牌中的传话游戏，中间环节一旦出错，后面就会有问题，故障率非常高。那有什么简便的方法。

- 通过$root，$parent(定向消息)

```
// 获取 根组件 的数据
this.$root.pri;

// 写入 根组件 的数据
this.$root.pri = 2;

// 访问 根组件 的计算属性
this.$root.sm;

// 调用 根组件 的方法
this.$root.prism();
```

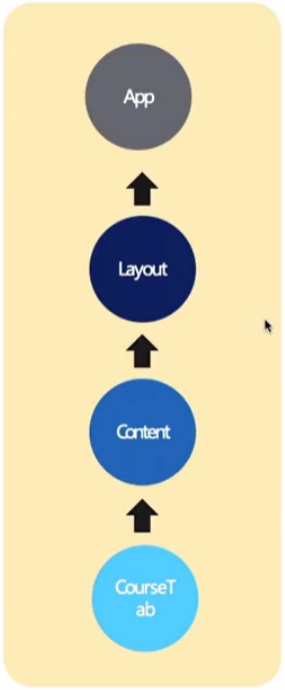
```
// 获取 父组件 的数据
this.$parent.pri;

// 写入 父组件 的数据
this.$parent.pri = 2;

// 访问 父组件 的计算属性
this.$parent.sm;

// 调用 父组件 的方法
this.$parent.prism();
```

通过$parent定向消息




```
dispatch(componentName, eventName, params) {
  var parent = this.$parent || this.$root;
  var name = parent.$options.componentName;

  while (parent && (!name || name !== componentName)) {
    parent = parent.$parent;

    if (parent) {
      name = parent.$options.componentName;
    }
  }
  if (parent) {
    parent.$emit.apply(parent, [eventName].concat(params));
  }
}
```

出处：element/src/mixins/emitter.js

- 通过ref

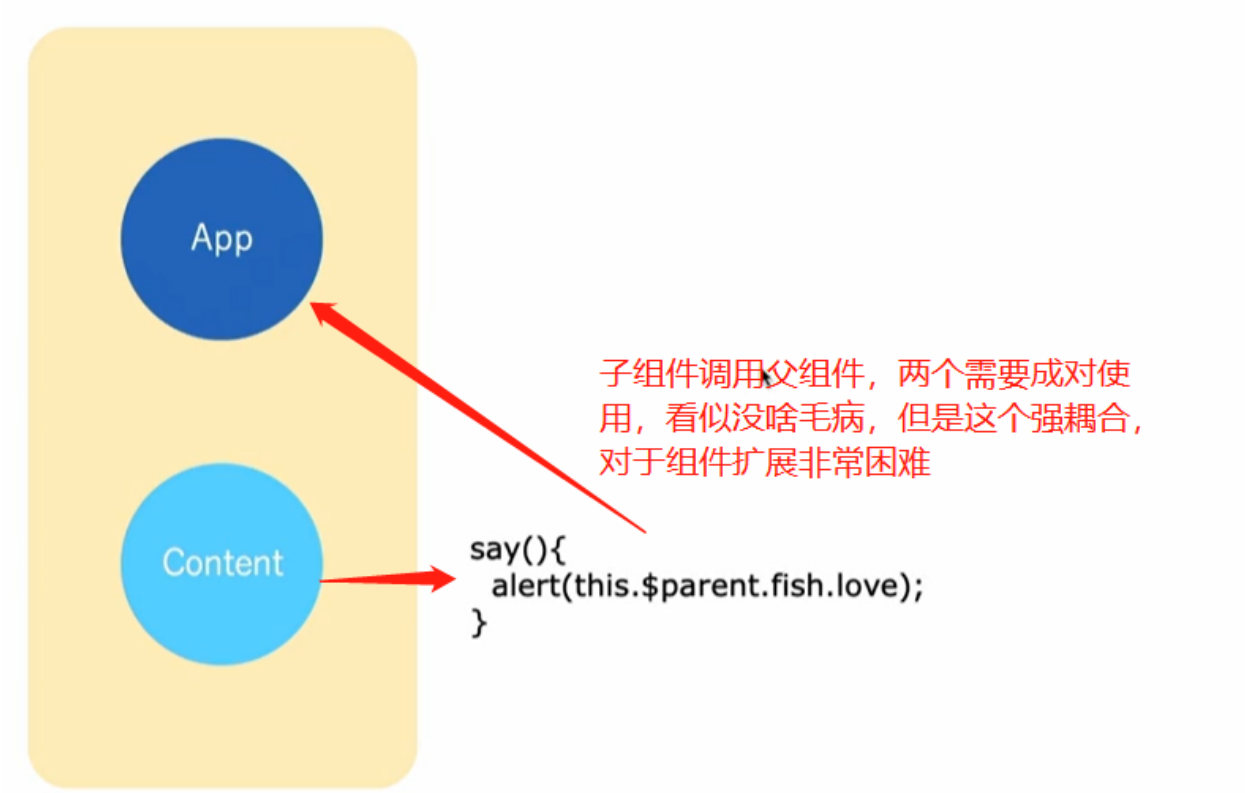
```
1  <base-input ref="usernameInput"></base-input>
```

```
1  this.$refs.usernameInput.focus();
```

注意

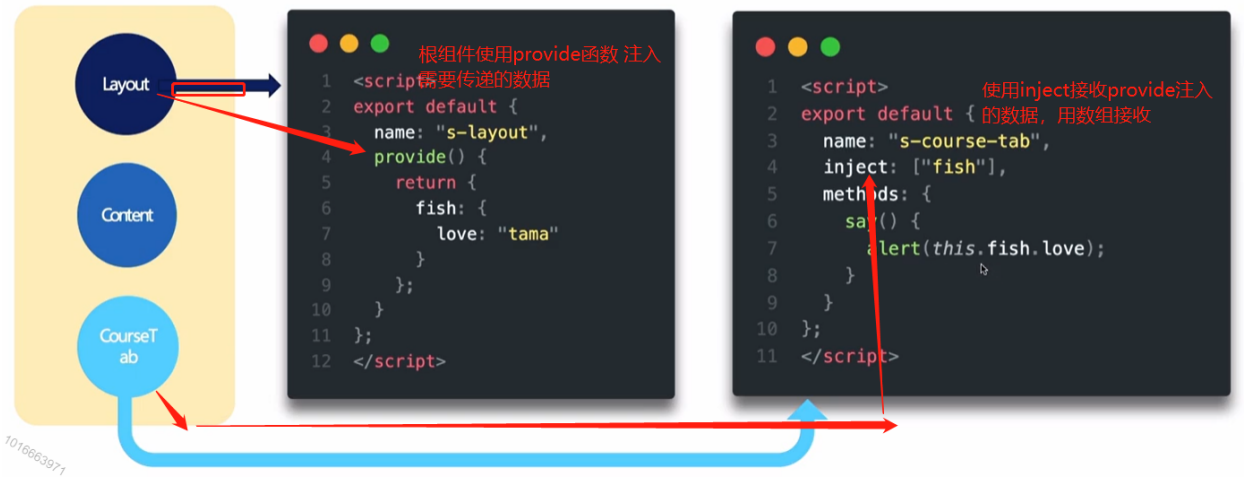
- $ref 只能在 mounted 生命周期钩子函数被调用之后才能使用
- $parent 和 $root 在各个生命周期钩子函数中都可以使用

虽然$root和$parent还有ref可以解决跨组件跨层级访问，但实际也造成一个强耦合，就样造成父子组件需要成对使用，缺一不可。这样的方式，非常不好，如何解决了呢？就看下面的依赖注入。



依赖注入

vue里面提供了依赖注入的方式所谓 依赖注入 **:**声明当前组件依赖的父组件们(直系的祖宗)的外部prop有哪些。这样当前组件需要哪些'传家包，他们才能过好这一生'， provide和inject的使用。注意provide和inject是一次性配置，并不是响应式的。



依赖注入的源码：

```
export function resolveInject (inject: any, vm: Component): ?Object {
  if (inject) {
    // inject is :any because flow is not smart enough to figure out cached
    const result = Object.create(null)
    const keys = hasSymbol
      ? Reflect.ownKeys(inject)
      : Object.keys(inject)

    for (let i = 0; i < keys.length; i++) {
      const key = keys[i]
      // #6574 in case the inject object is observed...
      if (key === '__ob__') continue
      const provideKey = inject[key].from
      let source = vm
      while (source) {
        if (source._provided && hasOwn(source._provided, provideKey)) {
          result[key] = source._provided[provideKey]
          break
        }
        source = source.$parent
      }
      if (!source) {
        if ('default' in inject[key]) {
          const provideDefault = inject[key].default
          result[key] = typeof provideDefault === 'function'
            ? provideDefault.call(vm)
            : provideDefault
        } else if (process.env.NODE_ENV !== 'production') {
          warn(`Injection "${key}" not found`, vm)
        }
      }
    }
    return result
  }
}
```

出处：vue2.0/src/core/instance/inject.js

总结:依赖注入其实对$root和$parent的一个封装，

优势:

祖先组件不需要知道哪些后代组件使用它提供的属性。

后代组件不需要知道被注入的属性来自哪里。

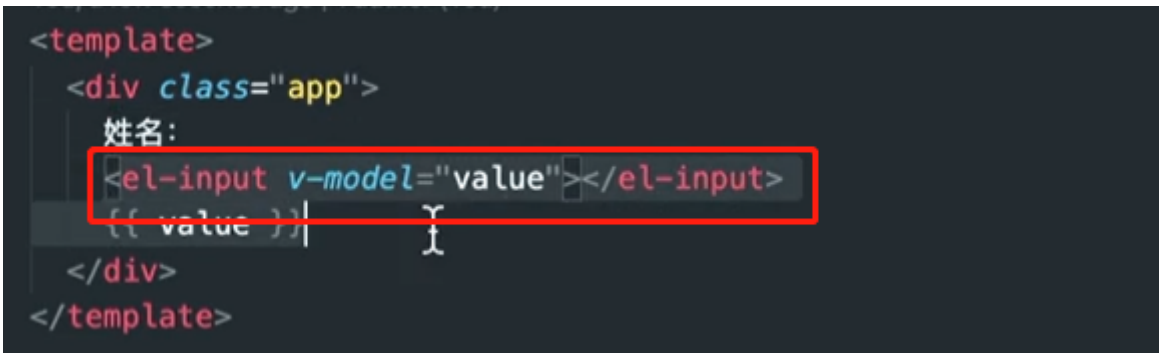
缺点:

但是组件间的耦合较为紧密，不易重构。

提供的属性是非响应式的，还是需要强买强卖。

组件二次封装

有时候，因为需求，或者其他的客户要求，需要对第三方UI库进行改装或者二次封装来达到相关需要。例如修改elementUi的样式。二次封装一个组件



- >>> 三箭头样式语法 修改当前组件el-input样式

修改el-input的boder样式。直接修改el-input__inner是不生效的，因为style是scoped，只是再当前组件内，生成的是一个的data-hash唯一值。vue当中提供了一个语法三箭头符号 >>> 即可修改当前组件的el-input_inner

- 提取出来，二次封装el-input的组件

子组件

```vue
<template>
  <div>
    <el-input v-model="value" @input="$emit('input', value)"></el-input>
  </div>
</template>

<script>
export default {
  model: {
    prop: "initValue",
    event: "input"
  },
  props: ["initValue"],
  data() {
    return {
      value: this.initValue
    };
  }
};
</script>

<style scoped>
.el-input >>> .el-input__inner {
  border-top: none;
  border-left: none;
  border-right: none;
}
</style>
```

父组件

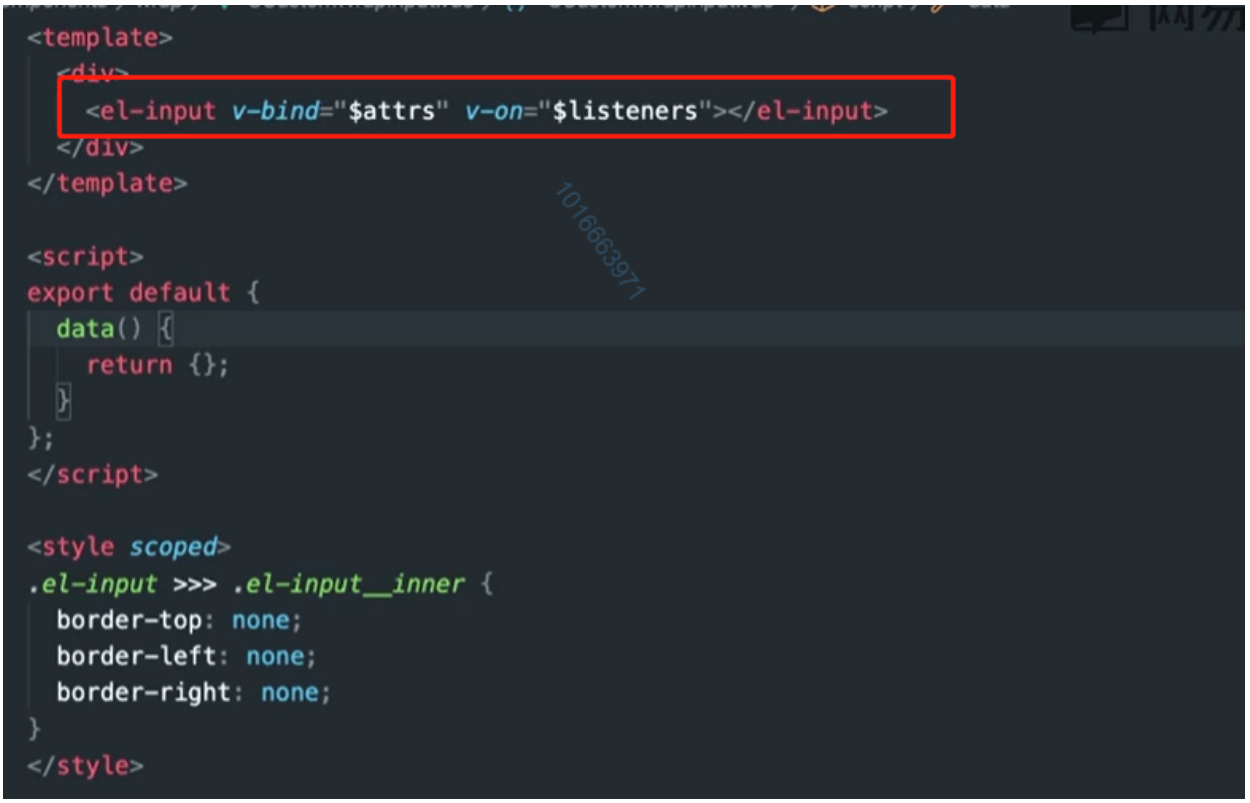
```vue
<template>
  <div class="app">
    姓名:
    <s-custom-wrap-input v-model="value"></s-custom-wrap-input>
  </div>
</template>

<script>
import SCustomWrapInput from "./components/wrap/SCustomWrapInput";

export default {
  name: "app",
  data() {
    return {
      value: "333"
    };
  },
  components: {
    SCustomWrapInput
  }
};
</script>
```

子组件这样写简单实现，如果el-input还有很多添加很多attr ， 除了input事件，还有很多其他的事件。这是时候需要用到v-bind="$attrs" v-on="$listeners" ,这里所谓得 $atters和$listeners 就是调用者 atters属性和注入的相关事件。这都抛出去由调用者自己去添加属性和注入事件。子组件最终的书写方式。

子组件



父组件

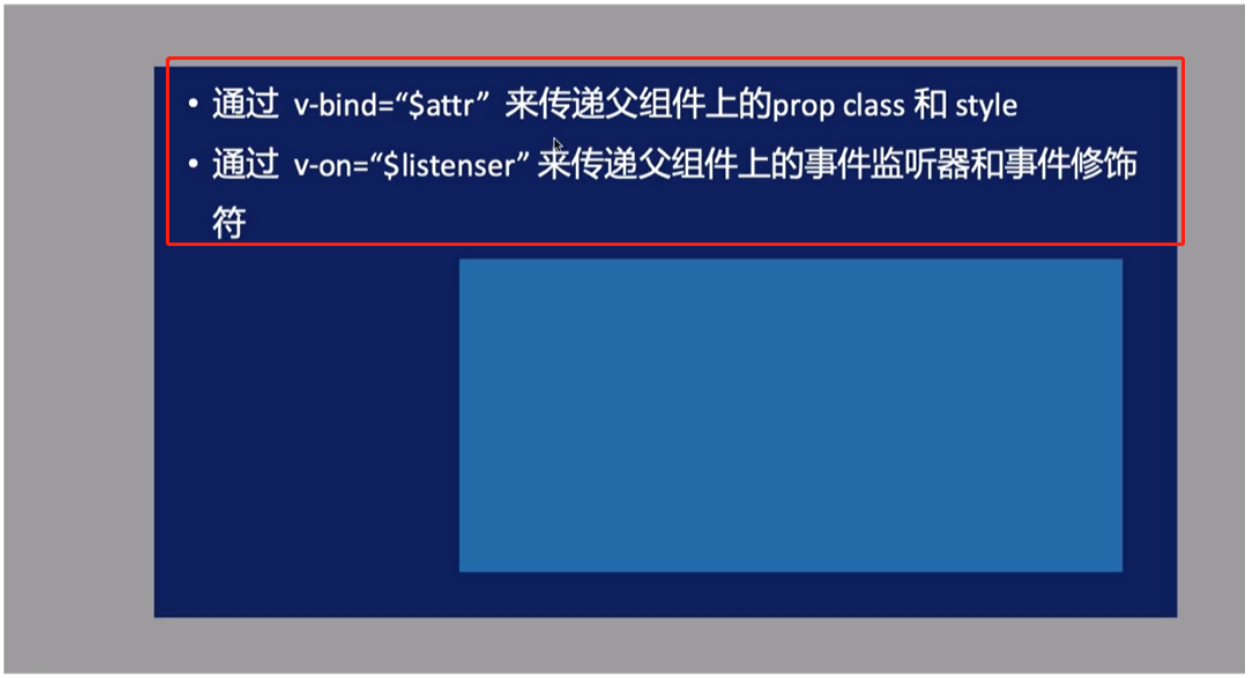
```vue
<template>
  <div class="app">
    姓名:
    <s-custom-wrap-input v-model="value" @blur="onBlur"></s-custom-wrap-input>
    {{ value }}
  </div>
</template>

<script>
import SCustomWrapInput from "./components/wrap/SCustomWrapInput";

export default {
  name: "app",
  data() {
    return {
      value: "333"
    };
  },
  components: {
    SCustomWrapInput
  },
  methods: {
    onBlur() {
      // eslint-disable-next-line no-console
      console.log("blur");
    }
  }
};
</script>
```

自己去注入事件 el-input有的事件都可以注入。添加属性 el-input有的属性都可以去添加

总结：



严格来说是$attrs和$listeners，调用者就是父组件直接传递 然后到达子组件。触发子组件的原始组件。