1. mixin

1.1 应用：抽离公共逻辑(逻辑相同，但是模板不一样，可用mixin)

1.2 缺点：数据来源不明确

1.3 全局混入：会影响每一个之后创建的 Vue 实例

```
Vue.mixin({
  created: function () {
    // 处理一些逻辑
  }
})
```

1.4 局部混入

```
mixins: [myMixin]
```

1.5 element-ui中的mixin应用

- 定义mixin
  - https://github.com/ElemeFE/element/blob/dev/src/mixins/emitter.js#L29

```
broadcast(componentName, eventName, params) {
  broadcast.call(this, componentName, eventName, params);
}
```

- 组件中引入mixin
  - https://github.com/ElemeFE/element/blob/dev/packages/dialog/src/component.vue#L48

```
export default {
  name: 'ElDialog',

  mixins: [Popup, emitter, Migrating],

  props: {
    title: {
      type: String,
      default: ''
    },
  }
  ....
}
```

- 组件中使用mixin中的方法/属性
  - https://github.com/ElemeFE/element/blob/dev/packages/dialog/src/component.vue#L184

```
updatePopper() {
  this.broadcast('ElSelectDropdown', 'updatePopper');
  this.broadcast('ElDropdownMenu', 'updatePopper');
},
```

1.6 合并策略

- 数据对象，在内部会进行递归合并，并在发生冲突时以组件数据优先

```
var mixin = {
  data: function () {
    return {
      message: 'hello',
      foo: 'abc'
    }
  }
}

new Vue({
  mixins: [mixin],
  data: function () {
    return {
      message: 'goodbye',
      bar: 'def'
    }
  },
  created: function () {
      console.log(this.$data)
      // => { message: "goodbye", foo: "abc", bar: "def" }
  }
 })
```

- 同名钩子函数将合并为一个数组，因此都将被调用。另外，混入对象的钩子将在组件自身钩子之前调用。

```
var mixin = {
  created: function () {
      console.log('混入对象的钩子被调用')
  }
}

new Vue({
  mixins: [mixin],
  created: function () {
    console.log('组件钩子被调用')
  }
})
```

```
// => "混入对象的钩子被调用"
// => "组件钩子被调用"
```

- 值为对象的选项，例如 methods、components 和 directives，将被合并为同一个对象。两个对象键名冲突时，取组件对象的键值对。

2. 插槽

2.1 通过弹窗组件来了解插槽

```
<el-dialog
  title="提示"
  :visible.sync="dialogVisible"
  width="30%"
  :before-close="handleClose">
  <span>这是一段信息</span>
  <span slot="footer" class="dialog-footer">
    <el-button @click="dialogVisible = false">取 消</el-button>
    <el-button type="primary" @click="dialogVisible = false">确 定</el-button
  </span>
</el-dialog>
```

2.2 默认插槽

```
<el-dialog>
  <div>这是一段信息</div>
</el-dialog>
```

```
// el-dialog
<div>
  <div>弹窗</div>
  <slot>这是默认的内容</slot>
</div>
```

2.3 具名插槽

```
<el-dialog>
  <div>这是一段信息</div>
  <template v-slot:footer>
    <div>footer</div>
  </template>
</el-dialog>
```

```
// el-dialog
<div>
  <div>弹窗</div>
  <slot>默认内容</slot>
  <slot name="footer">默认footer</slot>
</div>
```

2.4 插槽作用域

- 组件内部将数据传给插槽(比如，上传组件上传成功后会拿到file信息，可将file数据传给插槽)

2.4.2 具名插槽作用域

```
<el-dialog>
  <div>这是弹窗内容</div>
  <template v-slot:footer="footerSlotProps">
    <div>footer</div>
    <div>{{ footerSlotProps }}</div>
  </template>
</el-dialog>
```

```
// el-dialog
<div>
  <div>弹窗</div>
  <slot>默认内容</slot>
  <slot name="footer" :footerInfo="footerInfo">默认footer</slot>
</div>

data() {
  return {
    footerInfo: {
      name: 'ok'
    },
  }
},
```

2.4.2 默认插槽作用域

- 默认插槽作用域与具名插槽作用域同时存在时，默认插槽不能简写，必须写成v-slot形式

```
<el-dialog>
  <template v-slot:default="contentInfo">
      <div>这是弹窗内容</div>
      <div>{{ contentInfo }}</div>
  </template>

  <template v-slot:footer="footerSlotProps">
    <div>footer</div>
```

```
    <div>{{ footerSlotProps }}</div>
  </template>
</el-dialog>
```

```
// el-dialog
<div>
  <div>弹窗</div>
  <slot :contentInfo="contentInfo">默认内容</slot>
  <slot name="footer" :footerInfo="footerInfo">默认footer</slot>
</div>
```

2.4.3 编译作用域

- 父级模板里的所有内容都是在父级作用域中编译的；子模板里的所有内容都是在子作用域中编译的。

```
<el-dialog title="弹窗标题">
  <template v-slot:default>
    <div>{{ title }}</div>
  </template>
</el-dialog>
```

- 在slot中访问不到title，需要通过作用域插槽传递

3 插件

- 插件通常用来为 Vue 添加全局功能，比如vue-router vuex

3.1 注册插件

- Vue.use(VueRouter); Vue.use(Vuex)
- Vue.use 会自动阻止多次注册相同插件，届时即使多次调用也只会注册一次该插件。

3.2 编写插件

- 插件暴露install方法
- Vue.use(MyPlugin, options)会执行插件的install方法，并传入Vue和options

```
MyPlugin.install = function (Vue, options) {
  // 1. 添加全局方法或 property
  Vue.myGlobalMethod = function () {
      // 逻辑...
  }

  // 2. 添加全局资源
  Vue.directive('my-directive', {
      bind (el, binding, vnode, oldVnode) {
      // 逻辑...
      }
```

```
    ...
  })

  // 3. 注入组件选项
  Vue.mixin({
      created: function () {
      // 逻辑...
      }
      ...
  })

  // 4. 添加实例方法
  Vue.prototype.$myMethod = function (methodOptions) {
      // 逻辑...
  }
}
```

3.3 实现element-ui中的message插件

- Message组件有两种使用方式
  - 全局方法：Element 为 Vue.prototype 添加了全局方法 $message。因此在 vue instance 中可以采用本页面中的方式调用 Message
  - 单独使用：import { Message } from 'element-ui';
- 编写Message.vue

```
<template>
  <div class="el-messages">
    <div v-for="m in messages" :key="m.id">{{m.message}}</div>
  </div>
</template>

<script>
export default {
  data() {
    return {messages: []}
  },
  mounted() {
    this.id = 0; // 当前消息组件的id
  },
  methods: {
    add(options) {
      const id = this.id++;
      const layer = {...options, id}
      this.messages.push(layer)

      setTimeout(() => {
        this.remove(layer)
      }, options.duration)
    },
    remove(layer) {
```

```
      this.messages = this.messages.filter(message => message.id !== lay
    }
  }
}
</script>
```

- 对外提供JS的使用方法

```
let vm = null

const getInstance = function() {
  if(!vm) {
    // 单例模式
    vm = new Vue({
      render: h => h(MessageComponent)
    }).$mount() // $mount的作用是 render函数 -> 实际的dom元素

    document.body.appendChild(vm.$el)
  }

  return vm.$children[0]; // $children[0]就是我们传入的MessageComponent
}

const Message = {
  info(options) {
    getInstance().add(options)
  }
}

export {
  Message,
}
```

- 暴露提供install方法，使其成为插件

```
export default {
  install(Vue) {
    Vue.prototype.$message = {
      info: Message.info
    }
  }
}
```

4. 过滤器

4.1 定义：将原数据进行格式化显示，而不改变原数据

4.2 应用：货币符号、时间格式化（一般用在与业务关联不大的情况下，否则用computed）

4.3 全局过滤器

```
<template>
    <div>
        {{ timer | format }}
    {{ timer | format('YYYY:MM:DD hh:mm:ss') }}
    </div>
</tempalte>
```

```
Vue.filter('timeFormat', function(val, formatter = 'YYYY:MM:DD') {
    // 没有this
    return moment(val).format(formatter)
})
```

4.4 局部过滤器

```
filters: {
    timeFormat(val, formatter) {
        return moment(val).format(formatter)
    }
}
```

4.5 无法访问this

5. Vue响应式原理

5.1 核心api Object.defineProperty

```
  <div>
    <div>{{ a }}</div>
    <div>{{ info.name }}</div>
  </div>

  data() {
    return {
      a: 1,
      info: {
        name: 'xiaoming',
      }
    }
  }
```

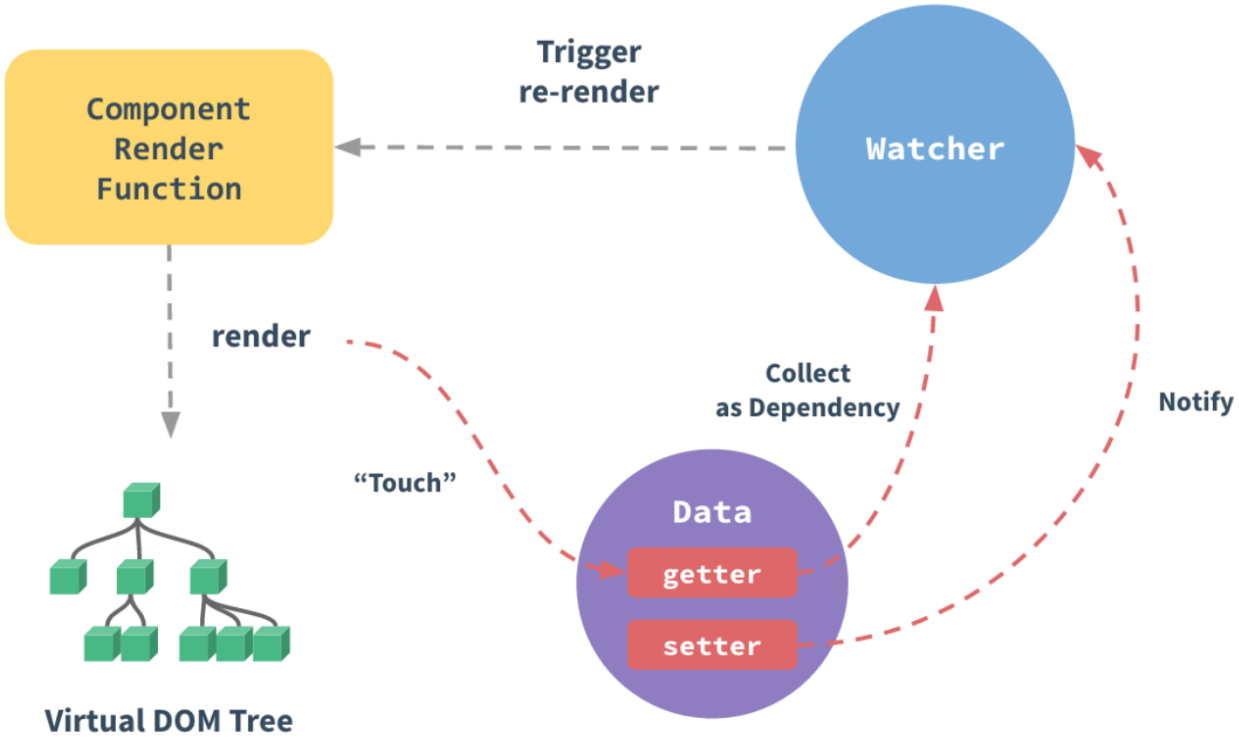
- 通过Object.defineProperty将属性转换为getter/setter

```
const dep1 = new Dep()
Object.defineProperty(this.$data, 'a', {
  get() {
    dep1.depend() // 收集依赖
    return value
  },
  set(newValue) {
    if (newValue === value) return;
    value = newValue
    dep1.notify() // 通知依赖
  }
})

const dep2 = new Dep()
Object.defineProperty(this.$data, 'info', {
  ...
})

const dep3 = new Dep()
Object.defineProperty(this.$data.info, 'name', {
  ...
})
```

- 收集依赖
  - 每个组件实例对应一个watcher实例
  - 在组件渲染过程中，把“touch”过的数据记录为依赖(触发getter -> 将当前watcher实例收集到属性对应的dep中)
- 触发更新
  - 数据更新后 -> 会触发属性对应的setter -> 通过dep去通知watcher -> 关联的组件重新渲染



5.2 注意事项

对象

- vue无法监测对象的添加
- 解决方案：this.$set(this.someObject, 'b' ,2)
- 注意：Vue 不允许动态添加根级别的响应式 property

数组

- Object.defineProperty无法监听数组索引值的变化,比如 this.a[0] = 44
- 解决方案：
  - this.$set(this.a, 0, 44)
  - this.a.splice(0, 1, 44)
- 数组长度的变化也无法监听
  - 解决方案：this.a.splice(newLength) // 省略第二个参数，表明删除从newLength及后面的数据
- 重写了数组的方法(push() pop() shift() unshift() splice() sort()reverse())
- this.a[1].name = 'lisi' // 这是更改对象的属性值

其他

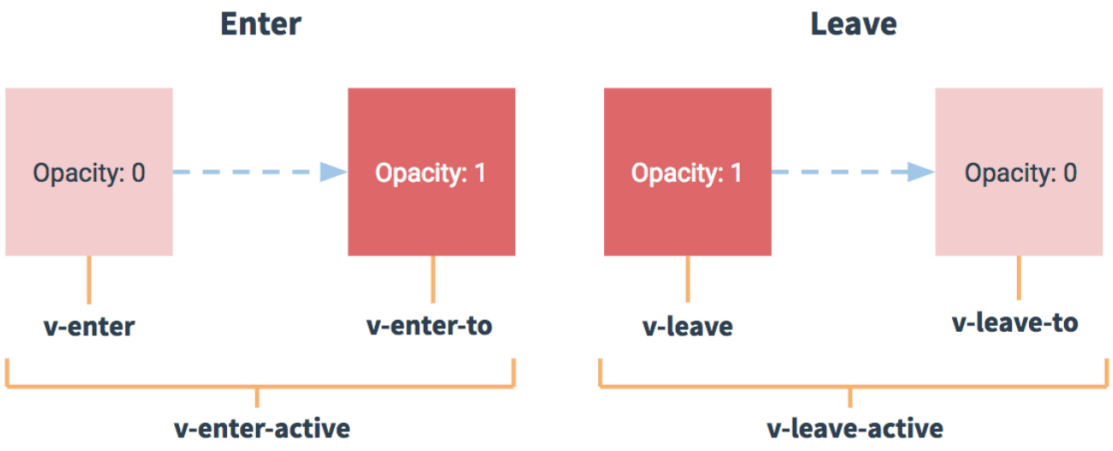
- 递归的循环data中的属性(可能会导致性能问题)
- 对于一些数据获取后不更改，仅仅用来展示的数据(比如说省、市)可以使用Object.freeze来优化性能
  `this.city = Object.freeze(data.city)`

6. 动画

6.1 进入&离开动画

6.1.1 单元素/组件动画(transition 组件)

- 用途：单元素/组件动画（比如：进入/离开）
- css3 transition
  - v-enter
    - 用来定义动画的初始状态
    - 元素插入之前被添加，元素插入后被移除
  - v-enter-active
    - 定义动画语句
    - 元素插入之前被添加，动画结束后被移除
  - v-enter-to
    - 用来定义动画的结束状态(一般不设置，因为元素有默认的显示状态，比如opacity默认为1)
    - 元素插入后被添加，动画结束后被移除
  - v-leave
  - v-leave-active
  - v-leave-to



  -
- 默认v-前缀，设置name后可以改变前缀，比如，v-enter变为my-transition-enter
- css3 animation
  - 与transition的区别是v-enter不会在元素被插入时删除，而是在animationed事件被触发时删除
- 自定义class名字

```
<transition
    enter-active-class="animate__animated animate__bounce"
    leave-active-class="animate__animated animate__tada"
>
    <div class="box" v-show="visible"></div>
</transition>
```

- Javascript钩子

```
<transition
  v-on:before-enter="beforeEnter"
  v-on:enter="enter"
  v-on:after-enter="afterEnter"
  v-on:enter-cancelled="enterCancelled"

  v-on:before-leave="beforeLeave"
  v-on:leave="leave"
  v-on:after-leave="afterLeave"
  v-on:leave-cancelled="leaveCancelled"
>
  <!-- ... -->
</transition>
```

```
beforeEnter: function (el) {
  // ...
},
// 当与 CSS 结合使用时
// 回调函数 done 是可选的
enter: function (el, done) {
  // ...
  done()
},
afterEnter: function (el) {
  // ...
},
enterCancelled: function (el) {
  // ...
},
```

6.2 列表动画

6.2.1 transition-group

- 会生成真实的元素，默认span，可以用个tag属性更改
- 提供一个唯一的key值

```
<div id="list-demo" class="demo">
<button v-on:click="add">Add</button>
```

```
<button v-on:click="remove">Remove</button>
  <transition-group name="list" tag="p">
    <span v-for="item in items" v-bind:key="item" class="list-item">
      {{ item }}
    </span>
  </transition-group>
</div>
```

```
data: {
  items: [1,2,3,4,5,6,7,8,9],
  nextNum: 10
},
methods: {
  randomIndex: function () {
    return Math.floor(Math.random() * this.items.length)
  },
  add: function () {
    this.items.splice(this.randomIndex(), 0, this.nextNum++)
  },
  remove: function () {
    this.items.splice(this.randomIndex(), 1)
  },
}
```

```
.list-item {
  display: inline-block;
  margin-right: 10px;
}
.list-enter-active, .list-leave-active {
  transition: all 1s;
}
.list-enter, .list-leave-to /* .list-leave-active below version 2.1.8 */
  opacity: 0;
  transform: translateY(30px);
}
```

6.3 状态动画

- 比如数字过渡 1 -> 2(1 1.1 1.2 1.3 ... 2)

6.4 动画库介绍

animate.css

- css动画库

velocity

- Js动画库（类似JQuery的$.animate）处理一些CSS属性，比如opacity, position等

- 注意 npm install velocity-animate

gasp

- Js动画库，除了CSS属性外，可提供状态过渡

作业及常见面试题

作业：通过插件实现一个预览图片的功能

面试题

- vue响应式原理，什么情况下使用this.$set，为什么有的情况下直接更改数组的索引值也会引起更新?
- 工作中公共逻辑是怎么处理的，你是怎么做的(介绍下你在项目中写的mixin或者插件等)