

echarts

xs	<768px 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
sm	≥768px 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
md	≥992px 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
lg	≥1200px 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—
xl	≥1920px 响应式栅格数或者栅格属性对象	number/object (例如: {span: 4, offset: 4})	—	—

echarts

常用api

- 库对象.init(DOM元素): 实例
- 实例.setOption(配置), 实例.getOption();
- 实例.resize() 改变父容器大小时, 自动自适应调整图表大小
- 频繁绘制的时候, 调用实例.clear() 清空上一轮的数据
- 实例.dispose()销毁释放资源
- 库对象.resgiterMap(亚洲/中国/世界);

配置项

样式: 通用于所有单独的个体, 例如, legend、title之类的这些

legend: 图例

tilie: 标题

xAxis/yAxis: x、y轴线

grid: 格子

visualMap: 地图用的 角落中做大致的数据展示与筛选

tooltip: 鼠 标提示框

series: 系列[]

- 柱状图[蓝色、红色]
- 线图
- 饼图
- 地图

dataZoom: 数据缩放

社区

[bookmarks 2023 5 18.html](#)

判断类型补充

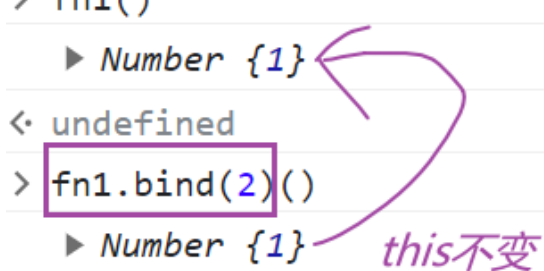
```
1
2 Object.prototype.toString.call({})
3 // '[object Object]'
4 Object.prototype.toString.call(new WeakMap())
5 // '[object WeakMap]'
6 function Person(){}
7 // undefined
8 Object.prototype.toString.call(new Person())
9 // '[object Object]'
10 Person.prototype[Symbol.toStringTag] = Person.name;
11 // 'Person'
12 Object.prototype.toString.call(new Person())
13 // '[object Person]'
14 // 范围太广了:obj instanceof Object
```

拦截器精华

1. 请求、响应、错误，三类都是按照use的顺序执行
2. 在请求拦截器中新发请求，会加入到队列中，立刻开始排队
3. 响应、错误拦截器中新发请求，会立刻发起请求
4. 响应config、响应response都会顺序获取，响应err
 - 4.1 如果没有写return Promise.reject(err)，后续就会作为response接收
 - 4.2 如果写了return Promise.reject(err)，后续作为err接收

第八天-FormCreator优化

```
> function fn(){ console.log(this) };
< undefined
> let fn1 = fn.bind(1)
< undefined
> fn1()
  ▶ Number {1}
< undefined
> fn1.bind(2)()
  ▶ Number {1}
< undefined
> |
```



this不变

- 相同函数被bind两次无效，只能被bind一次改变this

模块对象是持久共享的

组件创建 1
表单绑定this
组件创建 2
表单绑定this
组件创建 3
表单绑定this
组件创建 4
表单绑定this

课上来回切换申请贷款及首页引起的问题

```
function () { [native code] } 函数体
▶ [Vue warn]: Error in data(): "SyntaxError: Unexpected identifier 'code'"
found in
---> <Anonymous>
      <ElMain> at packages/main/src/main.vue
      <ElContainer> at packages/container/src/main.vue... (1 recursive calls)
      <Layout> at src/views/Layout.vue
      <App> at src/App.vue
      <Root>
▶ SyntaxError: Unexpected identifier 'code'
```

- bind来的函数再次被bind是无效的，原因是因为this已经封装到代码底层,[native code],
- 同时函数体字符串也会变成
`function () { [native code] }`
- 导致无法eval或者创建函数

解决方案

- 独立开相同属性反复被 bind赋值成为底层代码

```
"phone": [{
  // "required": true, "message": "请输入电话",
  _validator: phone_validator,
  "trigger": "blur" }],
```

- 页面再次赋值给另一个属性（保持原来的_validator）

```
let fn = loanInputConfig.rules.mobile_phone[1]._validator;
console.log(fn.toString(), 'fn.toString()'); //第二次变成 function () { [native code] }
eval("this.tmp = " + fn.toString());
loanInputConfig.rules.mobile_phone[1].validator = this.tmp.bind(this);
```

- 表单生成器处理的时候也避免给原属性赋值（保持原来的_validator）

```

if (rules) {
  for(let k in rules) {
    rules[k].forEach(rule=>{
      if (rule._validator && !rule.validator) {
        rule.validator = rule._validator.bind(this);
      }
    })
  }
}

```

- 其他验证器不需要this的，可以不写_，直接用validator

封装组件的思想

1. UI方向固定的能力，斑马纹、边框
 - 作为内部默认值 (A)
 - 外部如果传递，则覆盖 (A+)
 - 固定表头高度
2. 业务功能的选择
 1. 是否需要序号
 2. 多选
 3. 分页
3. 自由性的附加属性，不要阻塞

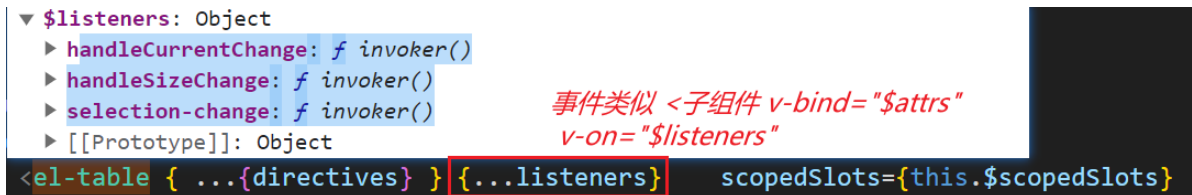
组件封装的优缺点

1. 比传统表格更简单（更简单）
2. 更贴合我们的业务（更适合）
3. 提高复用性
4. 减少代码冗余
5. 问题：
 - 5.1 不要阻断原生组件的能力attrs
 - 5.2 书写方式存在参数的遗漏风险
 - 5.3 底层原生存在的bug
 - 5.4. 提升或降低组件的效率

继承与混入

- 继承作为一个独立的个体来实现
- 混入，提炼代码，让各方都能复用该功能
- 继承同名函数被覆盖，混入生命周期各自执行
 - 混入相当于组合，继承的覆写是独占

不阻塞原生属性或事件传递的快捷方式



- 红字标识的是vue的template中的大概用法
- 下面el-table的则是JSX中的用法

关于dialog，如果先销毁子组件出现先的下方突然消失的问题

```
1 <el-dialog title="申请管理-编辑" :visible.sync="dialogvisible"
2   @open="childExist = true"
3   @closed="childExist = false"
4   width="50%">
5   <GFormCreator v-if="childExist" :config="editForm">
6     </GFormCreator>
7 </el-dialog>
```

