表单设计器进阶

本进阶文档适用于有前端编程经验的人

一、全局CSS增强

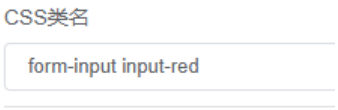
可以在**表单属性**里最下方找到**CSS增强**的输入框，在这里可以写css样式，在预览的时候会自动应用你写的样式。

配合几乎每个组件都有**CSS类名**属性，就能做到单独对某一个或某些组件写样式修改了。

CSS增强例子

例如我向页面中拖入了一个**单行文本**组件，并且想要把这个输入框的文字颜色改为**红色**，需要有以下步骤：

1. 修改组件的CSS类名属性，增加一个类名，例如叫 `input-red`



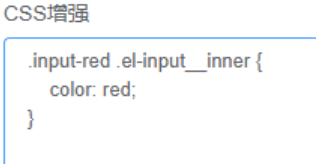
2. 在表单属性里的CSS增强中修改样式，如下所示：

3. 接着点预览查看效果

我们可以发现输入框的颜色并没有变成红色，这是因为element-ui将真正的input组件放到了更深层的dom里，我们可以用chrome的devTools工具查看一下

4. 找到真实的input类名，我们可以发现真实的input类名叫做 `el-input__inner`

5. 再次修改CSS增强代码，如下图所示



6. 再点击预览查看一下效果，发现输入框的颜色已经变成红色了，并且还不会影响到其他组件。

二、全局JS增强

可以在**表单属性**里最下方找到**JS增强**的输入框，在这里可以写js代码，在预览的时候会自动执行你写的JS代码。

在JS增强输入框里除了可以用挂载到window中的全局变量之外，还可以用我们给你封装好的几个变量以及方法，他们分别是：

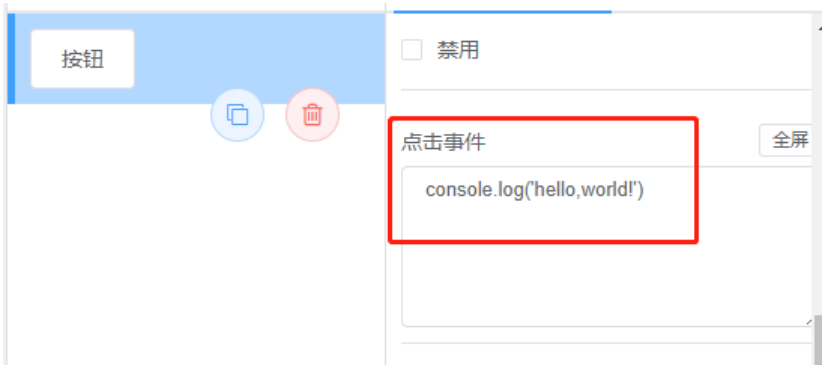
- `vm` Vue实例，可以调用Vue的一系列的方法，例如 vm.$nextTick();
- `event` Event对象，可以调用 event.type 来判断当前是什么增强类型（全局还是按钮）
- `api` 封装了一下api，具体如下
  - `getFormData(key)` 获取form表单的值，如果 key 为空，则返回所有的Data
  - `setFormData(key, value)` 设置form表单的值
  - `setFormOptions(key, optionsKey, optionsValue)` 设置 组件 的options
  - `watch(watchItems)` 设置监听 models 值的变化
    - 示例：

```
// 与vue的watch用法相同，可参见vue的官方文档
api.watch([
    name(val, oldVal){
        // name 发生了变化
    },
    info: {
        deep: true,
        handler(val, oldVal){
            // info 发生了变化
        }
    }
]);
```

  - `get(url, parameter)` 发送Get请求
  - `post(url, parameter)` 发送Post请求
  - `put(url, parameter)` 发送Put请求
  - `request(url, parameter, method)` 发送请求

三、按钮点击事件JS增强

除了全局JS增强，还支持按钮的JS增强，就是会在点击按钮的时候执行的JS代码



上图中，红框里的代码是按钮的默认点击事件，可以在预览界面点击界面看到输出。

与全局JS增强一样，按钮JS增强也封装好了几个变量以及方法，并且与全局JS增强完全一致。

示例：点击按钮获取输入框的值

如果想做到点击按钮的时候获取输入框的值，我们需要下面这几步：

1. 在页面中拖入一个输入框，并且将它的数据绑定Key改为 `name`

2. 在页面中拖入一个按钮，并且将它的点击事件改为下面这样：


```
// 可以用 api 里的 getFormData 方法获取
var name = api.getFormData("name");
alert("你填写的姓名是：" + name);
```

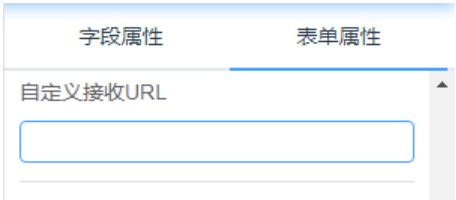
3. 点击预览，查看效果

三、自定义接收URL

目的是为了让用户自定义接收数据的后台接口地址，你可以在你自己定义的接口里做数据处理，并保存到数据库、

配置方式

在“表单属性”最下方有一个“自定义接收URL”输入框，你可以输入你的后台API地址



接收规则

请求类型	操作
POST	新增
PUT	修改

可以通过请求类型的不同来判断当前是新增还是修改操作

接收的参数均为一个 JSON 对象 `@RequestBody JSONObject json`

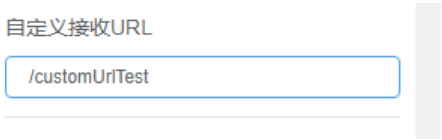
注意：如果只定义了 `POST` 方法而没有定义 `PUT` 方法，那么会在修改的时候报错，反之亦然。所以建议不要单独写 `@PostMapping` 或 `@PutMapping` ，而是写 `@RequestMapping` 来接收所有请求，在方法内进行判断请求类型，判断方式见下面的示例。

返回规则

建议返回 `org.jeecg.common.api.vo.Result` 类，但如果想要自定义返回类型的话，则需要包含以下字段：

字段名	类型	备注
success	Boolean	如果请求成功，请返回 `true` ，否则将视为失败
message	String	成功可不返回，若失败请返回失败的原因

参考示例




```
/**
 * 自定义接收URL：将数据抽取到新的表单
 * 【测试示例】
 * @param json
 * @param request
 * @return
 */
@RequestMapping("/customUrlTest")
public Result customUrlTest(@RequestBody JSONObject json, HttpServletRequest request) {
    boolean isPost = HttpMethod.POST.matches(request.getMethod());
    // post 为新增，put为修改，这里只对新增数据做处理
    if (isPost) {
        // 获取传递的 token
        String token = TokenUtils.getTokenByRequest(request);
        // 从传递进来的 formData 里抽取三个字段
        JSONObject formData = json.getJSONObject("desformDataJson");
        JSONObject staff = new JSONObject();
        staff.put("name", formData.getString("name"));
        staff.put("sex", formData.getString("sex"));
        staff.put("age", formData.getString("age"));
        // 通过 RESTful 风格的接口保存数据
        return RestDesformUtil.addOne("extract_test_staff", staff, token);
    } else {
        // 其他请求不做处理
        return Result.ok();
    }
}
```

上一篇：配置到菜单

下一篇：表单组件详解