# tmpUI

## 使用指南

预览版

# 前言

笔者在 2008 年的时候开始接触这个业界，一直专注于 Web 领域，到目前十多年了。

在过去的时代，PHP，ASP.net，Java 以及 jQuery 统治了 Web 开发的半壁江山。而现在，只剩下 PHP 和 Java。MVC 模式曾是主流，尽管时代变换，移动端崛起，但是仍然有许多思想也一直传承至今。

从过去的经验来看，笔者认为，尽可能不要去用某个框架（依赖）是非常重要的基本策略（这对要求对你所使用的编程语言本身要有相当程度的理解和应用，也就是所谓的编程基本功）。不过在实际开发中，不可能每项功能都要自己去再实现一遍（造轮子），因此这里面有一个准则：所选的轮子，不能限制了当前的功能，并对以后的功能造成约束。也就是所谓的低耦合度。

我们的最终目标是制作出好用实用的产品，一个产品在设计和实施的过程中，修修改改是很常见的情况，因此上面提到的准则可以帮助这个产品不会被束缚。

在设计 tmp.link 之初，我考察过 VUE 和 React ，这些框架都是为现代化应用程序设计，也具备相当不错的设计模式，我想如果有新项目，我应该会考虑用上它。

但是对于我们的项目来说，步子无法跨得很大，需要考虑到时间成本。因此，笔者设计了 tmpUI。

**那么 tmpUI 有没有突破上述准则（低耦合度）的呢？**

笔者认为是没有的。

Javascript 的世界有非常多，非常好用的插件，而且不会互相干扰。这对于依靠这些插件来实现功能的程序员来说真的是在友好不过了。

将项目从模板式结构的后端项目，迁移至 tmpUI，你不需要学习一种全新的设计模式甚至是语法。以前的 Javascript 插件现在也能用，这就是我们设计它的另外一个**重要目标**。

**tmpUI 的本质实际上是类似于 C++ 与 C 的关系 ，它要给浏览器添加 include 或者 require 的这样那样的功能，之前后端基于 MVC 模板架构的项目可以以最低成本迁移到前后端分离的模式。而开发人员也不会有高昂的学习成本。因此你也可以把 tmpUI 当作是另外一种形式的 Javascript 插件。**

目前。tmpUI 已经应用到了 tmp.link 和 vx.link 中，其中 vx.link 的前端也已开源，感兴趣的读者可以到 Github 上查看项目源代码：tmplink/vxlink

# 目录

# 1, 开始

## 项目结构

所以你已经知道，tmpUI 可以帮助你实现 PHP （或者类似其它后端语言）中模板模式中的 include 和 require。运用我们曾经的常用的项目结构即可。比如这样的结构：

- ./tmpui.js
- ./tmpui.checker.js
- ./index.html
- ./favicon.ico
- ./tpl/*.html
- ./lang/*.json
- ./assets/*.css
- ./assets/*.js
- ./plugin/*/*.js
- ./plugin/*/*.css
- ./plugin/*/*.html

足够简单，不是么？一切都井井有条，还是那个熟悉的味道。

tmpUI.js 作为 tmpUI 的引擎，index.html 是应用程序入口文件。

tpl 一般用于存放页面模板，assets 存放项目中用到的 css 和 js，lang 存放语言文件，plugin 存放第三方插件。

本文档附带有 5 个实例代码相关的项目，你可以在本地运行它们，阅读相关代码，获取灵感，甚至可以直接应用到实际的项目中。

## 应用程序入口

**通常**，`index.html` **是**应用程序入口，不过如果另有打算，也可以采用 `app.html` **或者** `application.html` 这样的名称。每个入口都可以单独配置为一个程序，也可以与其它入口**共享配置**。

**以下是一段常**规的应用程序入口的示例代码：

```
<!DOCTYPE HTML>

<html>

<head>

   <title>tmpUI</title>

   <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

   <meta name="viewport" content="width=device-width, initial-scale=1,
shrink-to-fit=no">

   <meta http-equiv="X-UA-Compatible" content="IE=Edge,chrome=1" />

   <meta name="renderer" content="webkit" />

   <script type="text/javascript" src="./tmpui.checker.js"></script>

   <script type="text/javascript" src="./tmpui.js"></script>

   <script type="text/javascript">

       var app = new tmpUI({});

   </script>

</head>

<body>

   <div id="tmpui"></div>

   <div id="tmpui_body" class="d-flex flex-column"></div>

</body>
```

```
</html>
```

在上面这段代码中，其实与普通的主页没有太大差异。通过 `script` 标签引入两个文件：

- `tmpui.checker.js`
- `tmpui.js`

**第一个文件用来**检查浏览器是否支持 `tmpui.js` 所用到的 `ES6` 特性，如果不支持的话则显示一段提示。

**第二个文件是 `tmpUI` 的主体文件。**

**接下来，通**过这段代码，初始化 `tmpUI` 实例。

```
<script type="text/javascript">

   var app = new tmpUI({});

</script>
```

**目前在花括号中是空的，**这里会放置一些 `tmpUI` 的配置参数，这是接下来我们要介绍的内容。

## 基本参数

这些参数是应用程序通用参数。

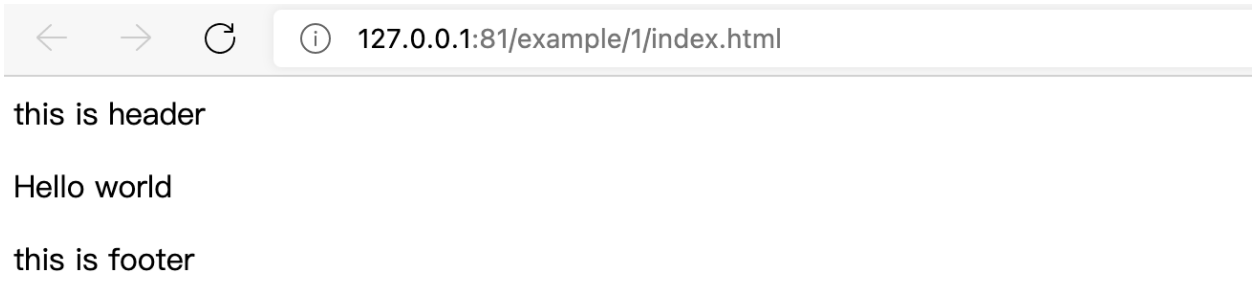
| 参数名 | 说明 | 默认值 |
| --- | --- | --- |

| | | |
|---|---|---|
| version | **版本号。版本号会在**请求资源时使用。 | 0 |
| siterRoot | 应用程序根目录。如果部署在站点子目录，你可以配置这**个参数**。 | / |
| loadingIcon | 载入页面的图像 | null |
| loadingPage | **是否启用**载入页面 | false |
| loadingProgress | 显示加载进度条（真实） | true |
| googleAnalytics | Google 统计 ID，配置后会自动加载 Google 统计的 JS | null |
| pageNotFound | **定制一个 404 的**处理页面，这里需要填入的是已配置好**的路由。当**发生 404 时，重定向到这个 404 的路由。<br><br>**当没有配置此参数**时，发生 404 时重定向到 / | null |
| dynamicRouter | 动态路由 | false |
| index | 应用程序入口位置，比如为 app.html | null |
| path | **静**态路由 | null |
| lang | 语言文件配置 | null |
| preload | **前置**资源组 | |
| append | **后置**资源组 | |

# 2. 界面模块化与路由

好了，现在我们可以开始第一份 tmpUI 程序了。为了方便行事，我已经提前把这个项目的源代码放置到了 github 仓库。

这个项目很简单，传统艺能 hello world。 在 vscode 中安装 live server 插件，然后可以直接启动它，你应该会看到这样的界面。




## 静态路由

在这个例子中，打开 /example/1/index.html ， 可以看到程序的配置是这样的：

```
var app = new tmpUI({
   "googleAnalytics": "G-4DGYKM9EHS",
   "loadingIcon": "/tpl/img/logo.png",
   "loadingPage": true,
   "loadingProgress": true,
```

```
   "path": {
        "/": {
            "title": "tmpUI App!",
            "body": {
                "./tpl/index.html": { "type": "html", "target": { "type": "body" } },
                "./tpl/header.html": { "type": "html", "target": { "type":
"id","val":"tpl_header" } },
                "./tpl/footer.html": { "type": "html", "target": { "type":
"id","val":"tpl_footer" } }
            }
        }
   }
});
```

这里重点要讲解的是 **path 参数**，**path 参数用于配置静**态路由。其结构如代码中所示。

**其中 "/" : {} 表示配置 / 路径下的**页面。如下参数

| **参数** | 说明 | **例子** |
| --- | --- | --- |
| title | 页面标题 | tmpUI App |
| body | 页面主体构成，由一组资源**来**组织。 | 见上方代码 |
| append | **可**选项。<br>**置入的后置**资源组，这里仅**需要填入后置**资源组的名**称**。 | ["append_res"] |
| preload | **可**选项。<br>**置入的前置**资源组，这里仅**需要填入前置**资源组的名**称**。 | ["preload_res1","preload_res2"] |

## 资源组

这里这里只用到了 body ，因此现在我们只说明 body。body 中使用 "资源组" 来组织要在其中放入什么。可以理解为喜闻乐见的 include 或者 require。

以代码为例

```
"./tpl/index.html": { "type": "html", "target": { "type": "body" } }
```

其中，"./tpl/index.html" 作为资源的实际路径，其值为具体的属性描述

当资源文件是 html 时，需要在属性描述中，将 type 设置为 html，这样 tmpUI 引擎会将这个资源作为 html 代码写入到页面中，具体要写入到哪里，则由 target 决定。

| 组合 | 说明 |
| --- | --- |
| `"target":{"type":"body" }}` | 此资源将写入到 tmpui_body 中。 |
| `"target":{"type":"id", "val":"dom"}}` | 此资源将替换到指定 id 的 DOM ， id 由 val 指定。注意，这里是替换。 |

在本例中，我们通过 target : body 来设定整个页面的基本模板，然后通过 target : id 来给页面设定模板的头部和页底。

如果要在每个路由中都这么配置，那会很累，所以，在下一个列子中，将介绍使用 "前置资源组" 和 "后置资源组" 来批量完成这一任务。

# 3, 前置与后置资源组

## 加载顺序

如果要在很多页面上嵌入很多的文件或资源，在 PHP 中，你可以在某个 header 文件中再包含另外一份 PHP 文件。而在 tmpUI 中稍许不同，我们采用更快捷的方式：前置和后置资源组。

tmpUI 加载资源文件时，是按这样的顺序加载的，这与一般的浏览器没有太大不一致的地方。

**前置资源组(preload) -> 主体(body) -> 后置资源组(append)**

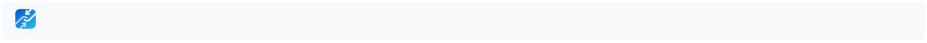
在上面的 例子1 中，由于我们没有配置前置资源组和后置资源组，因此只加载了 body 中设置的资源组。现在，我们要稍微改变一下代码，加入 bootstrap。bootstrap 将被设置到前置资源组。以下是变更后的代码 （ 代码可在 example 2 中获取 ）：

```
var app = new tmpUI({
   "googleAnalytics": "G-4DGYKM9EHS",
   "loadingIcon": "/tpl/img/logo.png",
   "loadingPage": true,
   "loadingProgress": true,
   "preload": {
       "bootstrap": {
           "/tpl/plugin/bootstrap4/jquery.min.js": { "type": "js", "reload": false },
           "/tpl/plugin/bootstrap4/bootstrap.bundle.min.js": { "type": "js", "reload":
false },
           "/tpl/plugin/bootstrap4/bootstrap.min.css": { "type": "css", "reload":
false },
           "/tpl/css/reset.css": { "type": "css", "reload": false },
       }
   },
   "path": {
       "/": {
           "title": "tmpUI App!",
           "preload":["bootstrap"],
           "body": {
               "./tpl/index.html": { "type": "html", "target": { "type": "body" } },
               "./tpl/header.html": { "type": "html", "target": { "type": "id", "val":
"tpl_header" } },
               "./tpl/footer.html": { "type": "html", "target": { "type": "id", "val":
"tpl_footer" } }
           }
       }
   }
});
```

## 资源组支持的其他类型文件

资源组目前支持 **html**，**css**，**js** 三种资源，如果是类型为 **css** 或 **js**，你还可以设置一个额外的 **reload** 参数，它可以控制在页面刷新或跳转时，是否重复加载对应的资源。在某些情况下可以设置为 **false**，这样可以避免重复加载，提升性能。

上述代码可以在 example 2 中找到，运行 example 2 的代码，你应该会得到类似如下的界面截图：



hello world



# 4, 进阶：嵌入式模板

## 使用嵌入代码标签

有时候，我们需要在页面中执行一些捎带逻辑的操作，比如生成列表。这种情况下，直接使用原生 JS 代码当然是最香的，tmpUI 的自带的模板引擎可以完成这个要求。

tmpUI 通过 <% ...your code %> 来嵌入 JavaScript 代码，这种代码与 PHP 的风格几乎一致，如果你此前使用过 PHP 那么接下来的事情应该很简单。

模板解析函数位于 tmpUI.tpl(DOM-id,Params) 第一个参数是模板文件的 DOM-id ，第二个参数是可选的，可以给模板传入一个变量，模板内使用变量 obj 来进行访问。这个函数会将解析好的 html 代码返回，然后你就可以插入到页面中的任何地方。

## 生成一份列表

首先，我们需要准备模板代码，这次我们准备两种写法，带参数和不带参数。模板代码一般直接放置到页面中，当然，你也可以另外单独放置到一份页面中，记得加载到 body 中即可。

```
<script type="text/template" id="tpl_no_array">
   <ul class="list-unstyled">
       <% for(let i = 0;i<=10;i++){ %>
           <li><% randomString(6) %></li>
       <% } %>
   </ul>
</script>

<script type="text/template" id="tpl_with_object">
   <ul class="list-unstyled">
       <% for(let i in obj){ %>
           <li><% obj[i] %></li>
       <% } %>
   </ul>
</script>
```

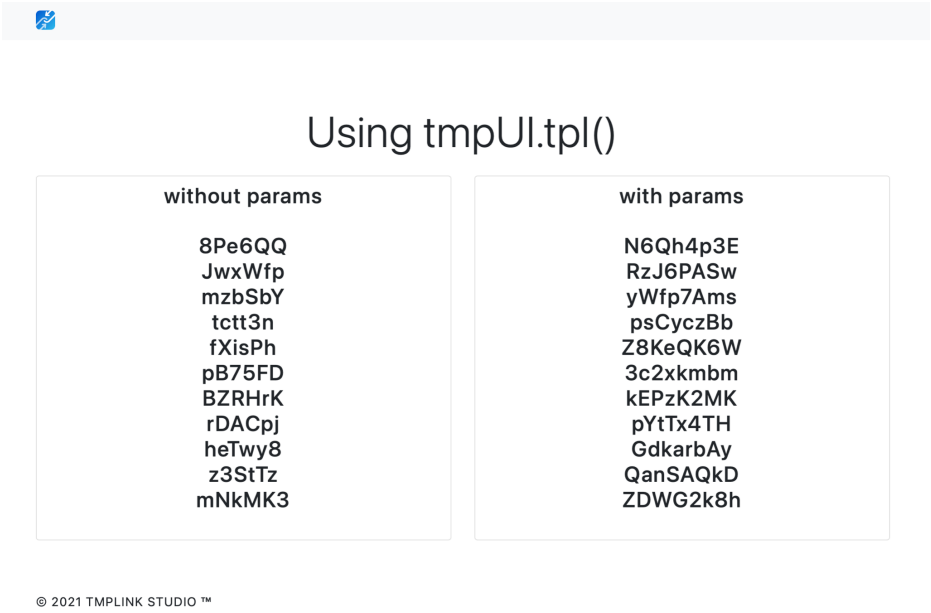
然后，我们需要准备准备一份 JavaScript 文件，用于在页面加载完成时执行模板解析的操作。这份文件的代码如下。

```
app.ready(()=>{
   //without params
   let html1 = app.tpl('tpl_no_array');
   document.querySelector('#no_array').innerHTML = html1;

   //with params
   let params = [];
   for(let i =0;i <= 10;i++){
       params.push(randomString(8));
   }

   let html2 = app.tpl('tpl_with_object', params);
   document.querySelector('#with_object').innerHTML = html2;
});
```

上述代码可以在 example 3 中找到，运行 example 3 的代码，你应该会得到类似如下的界面截图：




## 5, 进阶：多语言支持

多语言支持几乎是所有模板引擎的难题，我们尽量采取了一种比较折中的方案。多语言中通常面临两个问题：载入时渲染和载入后渲染。

本节所有涉及到的代码，你可以在 `/example/4` 中获取。




## 载入时渲染

**首先，我**们来解决第一个问题：载入时渲染。

载入时渲染在这里说的是，页面在加载完成之后，页面就绪时的时间线。我们通过 `i18n` 标签**来**给指定的DOM对象写入内容，比如这样：

```
<div i18n="index_title">.</div>
```

请注意，标签中的 `"."` **是必**须的字符，如果没有它，将不进行渲染。 `index_title` **是**对应**的**语言文件中的语言标记（稍后会说明语言文件的格式）。

载入时渲染时页面加载完成之后自动进行的。

## 载入后渲染

**第二个**问题：载入后渲染。**很多**时候，页面内容是在载入之后进行请求的，比如用户的账单信息。**那么如何**处理这些模板的语言渲染问题呢？

**很**简单，调用一次 `tmpUI.languageBuild()` 。这个函数可以对页面进行一次渲染。

## 实时语言更换

为界面更换语言时，同样也可以采用此做法。大多数模板更换语言时需要刷新页面，而 `tmpUI` **不需要，只需要运行一次** `tmpUI.languageSet(目标语言的代码)` 。

## 语言文件

语言文件是一份 `JSON` **格式的文件，其中以** `"语言标记":"内容"` 为格式组织。比如上方提**到的** `i18n="index_title"` ，则需要在语言文件中包含 `"index_title":"这是首页标题"` **才能**够正确在目标位置填充内容。

**要正确**应用多语言配置，你还需要在 `tmpUI` **初始化**时的参数中设置好多语言的的相关参数。

这里是一段示例代码：

```
"language":{
   "en":"./lang/en.json",
   "cn":"./lang/cn.json",
},
```

**其中，** `en` **与** `cn` **是**语言代码，后面的值则是对应的语言文件路径。

## 自动侦测与默认设定

`tmpUI` **的**语言引擎会在页面初始化时自动侦测浏览器的语言，并自动设置对应语言代码的文件，**以下是内置的**语言代码清单。

| 语言代码 | 目标语言 |
| --- | --- |
| cn | 简体中文 |
| hk | 繁体中文 |
| us | 英语 |
| jp | 日本语 |
| ru | 俄语 |
| kr | 韩语 |
| ms | 马来语 |
| de | 德语 |
| fr | 法语 |

语言引擎会将设置好的语言代码，存储到 `localStorage` 中，键名为 `tmpUI_language`。在下次访问时自动调用。

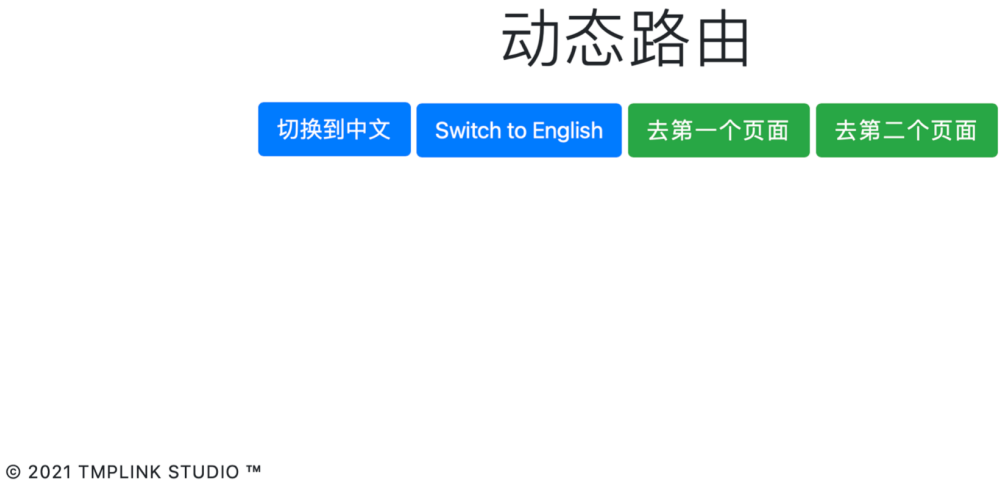
如果浏览器侦测到的语言，其语言代码没有在程序配置中进行配置，那么会将语言代码更改至 `languageDefault` 设定的值，这个值在没有配置的情况下，默认是 `en` 。

## 6, 进阶：动态路由

有些时候，我们需要处理添加大量不同样式的页面（甚至这些页面可以是后端服务器动态生成的），如果要为每个页面都编写路由和资源组，那么 index.html 中关于路由配置(path)的部分可能会很大，因此我们需要一种更为灵活的方案，动态路由就是为此而生。

动态路由的配置名是 DynamicRouter，比如你可以设置 "DynamicRouter":"/route" ，这样，tmpUI 就会将项目目录下的 /route 目录作为动态路由配置文件的存放目录。

在访问一个未在 path 中配置的路由时，比如 /test.html ，如果启用了动态路由配置，tmpUI 会尝试到 route 目录中寻找 /route/test.html.json 这个文件，如果能找到它，就读取其中的资源组配置，并加载资源组构建页面。




相关的示例代码，你可以到 /exampl/5/ 中浏览。

在这个例子中，你可以看到我们的 index.html 中的配置参数是这样的：

```
var app = new tmpUI({
   "googleAnalytics": "G-4DGYKM9EHS",
   "siteRoot": "/example/5/",
   "loadingIcon": "/tpl/img/logo.png",
   "loadingPage": true,
   "dynamicRouter": "route",
   "loadingProgress": true,
   "languageDefault": "en",
   "language": {
       "en": "./lang/en.json",
       "cn": "./lang/cn.json",
   },
   "preload": {
       "bootstrap": {
           "/tpl/plugin/bootstrap4/jquery.min.js": { "type": "js", "reload": false },
           "/tpl/plugin/bootstrap4/bootstrap.bundle.min.js": { "type": "js", "reload":
false },
           "/tpl/plugin/bootstrap4/bootstrap.min.css": { "type": "css", "reload":
false },
           "/tpl/css/reset.css": { "type": "css", "reload": false },
       }
   },
   "path": {
       "/": {
           "title": "tmpUI App!",
           "preload": ["bootstrap"],
           "body": {
               "./tpl/index.html": { "type": "html", "target": { "type": "body" } },
               "./tpl/header.html": { "type": "html", "target": { "type": "id", "val":
"tpl_header" } },
               "./tpl/footer.html": { "type": "html", "target": { "type": "id", "val":
"tpl_footer" } }
           }
       }
   }
});
```

**你可以**观察到这段配置代码中，`path` **的部分只配置了** `/` ，动态路由部分被配置为 `route`，**由于我**们的示例代码在子目录中，因此还需要配置 `siteRoot` 为 `/example/5/` ，这样才**能确保**动态路由的目录能被正确找到。

# 7, 处理资源路径和链接

## 转化链接为单页应用路径

由于 tmpUI 实质上是一个单页应用，因此我们需要单独处理页面内的链接。

通过给 <a> 标签加入 tmpui-app="true" ，在页面初始化时，tmpUI 会查找设置了此属性的 a 标签，并将链接转换成适当的应用内路由。

比如 <a tmpui-app="true" href="/login">

在页面初始化之后，会自动转换为 <a href="?tmpui_page=/login">

为什么要这么做？为了更好的兼容性。

尽管在内部路由系统中，使用 # 也可以直接拦截所有的 URL 并处理其路由，但是如果访客是从外部链接进入，则无法在服务器端日志中得到记录。因此我们采用查询符号来将这些请求作为 GET 参数。

而在某些程序中，带有 # 的链接，在渲染时（或者复制链接时），可能不会处理 # 及其之后内容。

此外，这种形式的链接，无需在服务器端进行任何配置（比如 Nginx 的重写配置）。

## 子目录下的静态资源路径处理

对于图片等资源来说，可以通过标签设置 tmpui-root="true" 来应用额外的路径重写规则。这通常是在设置了 siteRoot 参数时的配套设置。

比如，当设置了 siteRoot 为 /subdir/ 时，<img src="a.jpg" tmpui-root="true"> 将会被扩展为 <img src="/subdir/a.jpg" />

请注意，siteRoot 配置的路径，后面必须跟上 /。如果在资源组中配置的地址中包含 /，则不会应用 siteRoot ，因为 tmpUI 会认为你想要配置的是绝对路径，这与 URL 的路径规则是基本一致的。