帮助方法

1、自定义的帮助方法：行内helper， 最终输出只是字符串的helper

常见使用场景：数据格式化，数据处理

在app.js中注册

1、导入 const hbs = require('hbs');//导入hbs库，扩展handlebars

2、注册 hbs.registerHelper('**addOne**', function (num) {

return ++num;

})

3、使用 在hbs页面中 {{**addOne** xxx}}

当有多个注册要提取到一个目录的注册方法

1)、在项目里创建一个新目录**helper** 在新目录下创建一个**js**文件用于保存自定义的方法

导入：**const hbs = require('hbs');//导入hbs库，扩展handlebars**

注册：**hbs.registerHelper('addOne', function (num) {**

return num+99;

})

在**app.js**里引入注册信息：**const helper = require('./helper');//注册hbs帮助方法**

moment 日期工具类**moment.js**，日期获取，格式化

npm i moment -S 安装

2、块级的helper,最终输出是html的helper 主要作用做代码的搬家迁移

1、导入 **const hbs = require('hbs');//导入hbs库，扩展handlebars**

2、注册 **const blocks = {};//代码块缓存对象**

hbs.registerHelper('extend', function (name, context) {

// context 是上下文，保存有用方法和数据，最后一个参数永远是**context**

```
    let block = blocks[name]; // block用来存放代码块
    if (!block) {
        block = blocks[name] = [];
    }
    // 变异指令中代码块并放入block
    block.push(context.fn(this));
    // 与context.fn()配对还有一个方法 context.inverse()
})

hbs.registerHelper('block', function (name) {
    const val = (blocks[name] || []).join('\n')
    blocks[name] = [];//清空缓存
    return val;
})
```

```
const blocks = {};//代码块缓存对象
hbs.registerHelper('extend', function (name, context) {
    // context 是上下文，保存有用方法和数据，最后一个参数永远是context
    let block = blocks[name]; // block用来存放代码块
    if (!block) {
        block = blocks[name] = [];
    }
    // 变异指令中代码块并放入block
    block.push(context.fn(this));
    // 与context.fn()配对还有一个方法    context.inverse()
})

hbs.registerHelper('block', function (name) {
    const val = (blocks[name] || []).join('\n')
    blocks[name] = [];//清空缓存
    return val;
})
```

3、使用

保存代码到缓存：//在A页面将要搬家的代码

```
{{#extend 'jquery'}}
<script>
  $(function () {
  })
</script>
{{/extend}}
```

在B页面读取到A页面的缓存代码

```
<!--在B页面搬进A的代码jquery-->
<script src="/javascripts/jquery.js"></script>
{{{block 'jquery'}}}
```

3、N多帮助方法库 https://github.com/helpers/handlebars-helpers

```
npm install --save handlebars-helpers      安装helpers库
const helpers = require('handlebars-helpers');//helpers库
// 只导入一部分,并且和我们的handlbars实例挂钩，对需要使用的引入
helpers.comparison({handlebars: hbs.handlebars})
使用：参见文档
```

部分视图（ **partial**），对公共功能组件的抽取

1、引入 const hbs = require('hbs'); //导入hbs库，扩展handlebars

引入 const path = require('path'); // 处理路径相关

2、注册 说明视图的所在目录

```
hbs.registerPartials(path.join(__dirname, '../views/partials'));
```

3、在views目录下新建一个partials目录，将公共的视图代码抽取出来，放入partials目录下新建一个hbs的页面文件里

views新建目录partials，新建nav.hbs页面文件

将公共视图代码放入新的hbs文件，在需要展示视图的hbs文件中通过{{> nav}} {{> 文件名}}

4、动态 partial （通过获取的数据来确定要显示的视图内容）

注册： name参数是一个变量，通过这个变量的值来显示输出视图

```
hbs.registerHelper('whichPartial', function (name) {
        return name;
 })
```

使用：**{{> (whichPartial navName)}}**

5、块级部分视图 partial-block

作用**1**、错误处理－－－－>没有被注册也能执行

{{#> ooxx}}

出现错误时，您会看到这句话

{{/ooxx}}

作用**2**、**<!--**使用块级**partial**传递内容，相当于代码搬家**-->**

定义：在**partials**目录下建一个**win**的**hbs**页面文件

```
<div class="win">
        <div class="win-title">title</div>
          {{> @partial-block}}
</div>
```

使用：

{{#> win}}

** **这是窗口里面的内容。。。。。。。。。。。。。。。** -->**

{{/win}}

作用**3**、布局组件

定义：在**partials**目录下建一个**layout**的**hbs**页面文件

```
<div class="top">
    {{> top}}
</div>
```

使用：

{{#> layout}}

{{#*inline 'top'}}

这是**top**

{{/inline}}

{{/layout}}