**文件依赖管理系统**

# 概述

随着前端系统的日渐复杂，一套系统所引入的脚本文件会越来越多，为了便于开发者管理脚本文件，本文档提供了一套依赖管理系统，按照此系统中定义的规范来管理和使用脚本，会配套提供前端打包发布工具。

依赖管理系统所管理的最小单元为文件，以路径为标识，支持集成所有非自带载入器的第三方框架（即可以直接使用script标签引入页面使用的均可使用此依赖管理系统）

依赖管理系统仅在开发阶段有效，在打包发布后被完全剔除掉，因此不会对线上产品产生任何副作用

老旧项目使用此依赖管理系统及配套的打包发布工具无需对原有代码做任何侵入即可使用

# 依赖库

依赖管理系统为用户提供一个依赖库（define.js），用户在使用依赖管理系统之前需先引入该库。

<script src**=**"/path/to/lib/define.js"></script>

# 依赖接口

以下所有接口均使用NEJ作为名字空间，如define接口使用时用NEJ.define

## define

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 可选，当前文件路径 |
| Array | 可选，依赖的其他文件路径 |
| Function | 可选，当前文件需要执行的脚本 |
| **输出** | Void |  |
| **描述** | 定义脚本文件，并指定当前脚本执行所需的其他依赖文件 | |

应用举例：

NEJ.define('{pro}widget/list.js',[

'{lib}util/ajax/tag.js',

'{lib}util/template/item.js'],

**function()**{

// TODO something

});

## config

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | Array | 可选，依赖映射表 |
| Array | 可选，入口屏蔽文件列表 |
| **输出** | Void |  |
| **描述** | 通过配置来管理依赖，对于历史遗留项目或者使用了非AMD规范的第三方库如需引入依赖管理，可以通过此接口先配置依赖关系表，后期维护可直接使用依赖方式进行  开发阶段会全部依赖，发布只提取页面使用的脚本 | |

应用举例：

被指定的deps.js文件

NEJ.config({

'{pro}a.js':['{pro}b.js','{pro}c.js','{pro}d.js'],

'{pro}b.js':['{pro}c.js','{pro}e.js'],

'{pro}c.js':['{pro}d.js','{pro}e.js','{pro}a.js'],

'{pro}d.js':['{pro}e.js'],

'{pro}f.js':['{pro}a.js']

},['{pro}f.js']);

入口文件

<script src**=**"../../../src/define.js?d=./deps.js&pro=./"></script>

<script src**=**"./f.js"></script>

<script>

define(['{pro}a.js',

'{pro}e.js'],

**function()**{

log('app');

});

</script>

## patch

|  |  |  |
| --- | --- | --- |
|  | **类型** | **描述** |
| **输入** | String | 必须，平台的判断条件，如2=<TR<4 |
| Array | 可选，依赖文件列表 |
| Function | 可选，当前条件下需要执行的脚本 |
| **输出** | Void |  |
| **描述** | 解析平台相关的依赖文件，平台支持TR|WR|GR，没有比较操作符表示支持当前内核所有release版本 | |

引擎标识说明

|  |  |
| --- | --- |
| 标识 | 说明 |
| T | Trident引擎，如IE |
| W | Webkit引擎，如chrome |
| G | Gecko引擎，如firefox |

内置的Trident引擎版本对应的IE版本关系：

|  |  |
| --- | --- |
| Trident版本 | IE版本 |
| 2.0 | 6 |
| 3.0 | 7 |
| 4.0 | 8 |
| 5.0 | 9 |
| 6.0 | 10 |
| 7.0 | 11 |

应用举例：

NEJ.define(['./hack.js'],

**function()**{

// 针对trident平台的处理逻辑

NEJ.patch('TR',**function()**{

// TODO

});

// 针对gecko平台的处理逻辑

NEJ.patch('GR',['./hack.firefox.js'],

**function()**{

// TODO

});

// 针对IE6平台的处理逻辑

NEJ.patch('TR==2.0',['./hack.ie6.js']);

// 针对IE7-IE9的处理逻辑

NEJ.patch('3.0<=TR<=5.0',**function()**{

// TODO

});

});

# 依赖配置

依赖配置使用define.js文件的查询参数进行配置

<script src**=**"/path/to/lib/define.js?a=aa&b=bb"></script>

系统保留单字母命名的参数配置，预置的配置参数如：g、c、d、p，p参数的说明详见[平台配置](#_平台配置)部分

## 预置配置

### g=[true|false]

配置项目中是否存在全局的define函数，默认为false，如果配置为true则项目中必须使用NEJ.define来使用本依赖管理系统，否则如果项目没有定义额外的define函数则可以使用define函数来使用此依赖系统

<script src**=**"/path/to/lib/define.js?g=true"></script>

### c=[gbk|utf-8|…]

配置项目编码格式，依赖载入的脚本编码格式会设置为此参数配置的信息，打包输出也为参考此参数进行输出，默认为utf-8，如果配置为其他编码格式则需保证项目的相关文件的编码必须统一为此配置的编码格式

<script src**=**"/path/to/lib/define.js?c=gbk"></script>

### d=/patch/to/dep.js

配置第三方库或者遗留项目文件的依赖关系，具体使用可参看[NEJ.config](#_config_1)接口的说明

<script src**=**"/path/to/lib/define.js?d=./dep.js"></script>

## 路径配置

文件定义时指定的路径可以通过路径配置的变量来指定前缀，如路径配置时指定了变量A，则文件定义时可以直接使用”{A}”来表示A指定的路径前缀。



### lib

框架路径配置名称为”lib”，文件定义和依赖时可直接使用”{lib}”来表示框架的路径。此配置信息通过外联的define.js路径自动解析。

### pro

项目脚本根路径配置名称为“pro”，文件定义和依赖时可直接使用“{pro}”来表示脚本根路径，此配置信息可以通过外联的define.js路径中的查询串进行配置。如果没有自定义配置则默认相对于引入define.js的文件所在目录的“../javascript/”路径。

### patch [deprecated]

框架补丁路径配置名称为“patch”，文件定义和依赖时可直接使用”{patch}”来表示补丁的路径。此配置信息通过外联的define.js路径中的查询串中的p参数解析。

不建议使用，参见[platform](#_platform)

### platform

控件依赖补丁名称为“platform”，只用于文件依赖，使用“{platform}xxx.js”来表示控件依赖的补丁文件，会解析为依赖xxx.js和xxx.patch.js两个文件。xxx.js为W3C/ES规范实现方式，提供所有标准平台支持的公用部分，xxx.patch.js通过NEJ.patch接口提供不同平台对这些接口的特有实现逻辑

### native

框架混合模式路径配置名称为“native”，文件定义和依赖时可直接使用”{native}”来表示混合native适配的路径。此配置信息通过外联的define.js路径中的查询串中的p参数解析。

### 自定义路径

其他自定义路径可以通过引入define.js文件时作为查询参数输入进行配置，如自定义com路径，则可以通过以下方式进行配置（配置路径中以”./”、”../”起始的相对路径相对于当前地址栏路径）

<script src**=**"/lib/define.js?com=../js/"></script>

## 平台配置

平台参数在开发及打包过程中都会使用，框架支持平台参数的配置通过define.js路径上查询串中的p参数输入。

平台配置信息，此配置又分两类基本配置：补丁配置和混合配置，因为混合模式下使用的浏览器引擎固定，因此当配置中出现混合类型的配置时忽略补丁配置的值。

如果在引入依赖定义库时未指定平台信息则表示系统需对全平台浏览器支持。

### 补丁配置

主要用来修正浏览器平台对接口及控件的支持，按照目前浏览器引擎划分，参数值由一个或者多个平台标识组成，标识支持如下所示：

|  |  |
| --- | --- |
| 标识 | 说明 |
| gk | 以gecko为核心的浏览器平台，如firefox等 |
| wk | 以webkit为核心的浏览器平台，如chrome、safari等 |
| td | 以trident为核心的浏览器平台，如IE、360、maxthon等 |
| td-0 | 以trident为核心的浏览器平台，且引擎内核版本大于等于3.0，即IE>=7 |
| td-1 | 以trident为核心的浏览器平台，且引擎内核版本大于等于7.0，即IE>=10 |

对于此类值的配置自动生成的patch路径值为{lib}patched/目录下的一个或者多个路径，因为为非混合模式，因此native值不存在，实际引入的框架和应用中都不允许出现{native}的路径。



### 混合配置

主要用于混合开发模式下对native接口的适配，按照native平台划分，参数值由一个标识组成，多个标识则以识别的第一个标识为准，标识支持如下所示：

|  |  |
| --- | --- |
| 标识 | 说明 |
| cef | 基于cef框架混合应用，主要针对桌面应用 |
| ios | ios平台混合应用，如iphone应用、ipod应用、ipad应用等 |
| win | windows phone平台混合应用 |
| android | android平台混合应用 |

对于此类值的配置自动生成的patch路径值为{lib}patched/目录下对应的引擎目录，native值为{lib}native/目录下的具体配置目录。



# 文件定义

文件定义使用[define](#_define)接口，define支持以下几种形式

## 完整形式

带三个参数依次为：当前文件路径，依赖文件列表，文件执行函数

NEJ.define('{pro}widget/list.js',[

'{lib}util/ajax/tag.js',

'{lib}util/template/item.js'],

**function()**{

// TODO something

});

## 无文件路径

带两个参数依次为：依赖文件列表，文件执行函数

## 无依赖列表

NEJ.define(['{lib}util/ajax/tag.js',

'{lib}util/template/item.js'],

**function()**{

// TODO something

});

带两个参数依次为：当前文件路径，文件执行函数

NEJ.define('{pro}widget/list.js',

**function()**{

// TODO something

});

## 仅执行函数

带一个文件执行函数的参数，多用于没有依赖的入口文件，这类文件等价于直接定义一个函数执行，一般也可以不使用依赖系统

NEJ.define(**function()**{

// TODO something

});

此方式等价于定义函数后直接执行

(**function()**{

// TODO something

})();

## 仅依赖列表

带一个依赖文件列表参数，一般只用于归集依赖列表，将常用的依赖集合使用一个文件标识

NEJ.define([

'{lib}util/ajax/tag.js',

'{lib}util/template/item.js'

]);

# 循环依赖

当依赖链出现了环时我们就认为出现了循环依赖，依赖管理系统对于循环依赖的处理分两种情况处理



* 强依赖：避免代码出现强依赖，执行过程会出异常
* 弱依赖：允许出现弱依赖，可以正常处理

## 强依赖

在define接口中输入的执行函数里直接调用了依赖列表中的其他文件中的API的情况



## 弱依赖

在define接口中输入的执行函数里没有直接调用依赖列表中的其他文件中的API，所有对其他文件的API的调用均在当前文件提供的API中调用的情况



# 使用范例

## Html文件

<!DOCTYPE html>

<html>

<head>

<title>test</title>

<meta charset**=**"utf-8"/>

<link rel**=**"stylesheet" href**=**"./css/page.css"/>

</head>

<body>

<div><!-- content --></div>

<script src**=**"/lib/path/define.js?pro=../js/"></script>

<script src**=**"./javascript/page.js"></script>

</body>

</html>

## JavaScript文件

define(['{lib}util/template/tpl.js',

'{lib}util/dispatcher/dispatcher.2.js'],

**function()**{

**var** \_ = NEJ.P;

/\* start up dispatcher \*/

\_('nej.e').\_$startup({

rules:{

rewrite:{

'404':'/m/blog/list/',

'/m/blog/list/':'/m/blog/',

'/m/setting/account/':'/m/setting/'

}

},

modules:{

'/?/tab/':'module/tab/index.html',

'/?/blog/tab/':'module/blog/index.html',

'/m':{

module:'module/layout/system/index.html',

composite:{

tab:'/?/tab/'

}

},

'/m/blog':{

module:'module/layout/blog/index.html',

composite:{

tab:'/?/blog/tab/'

}

}

}

});

});

# FAQ

## Q#1

Q： 哪些项目可以引入此依赖管理系统？

A： 凡项目中脚本是直接通过script标签引入到页面中的均可直接使用此依赖管理系统

## Q#2

Q： 如何解决存在依赖关系的第三方库的依赖引入？比如jquery-ui-0,jquery-ui-1都依赖了jquery，jquery-ui-2依赖了jquery-ui-1，如果采用原始的页面编写方式则会采用这种形式引入页面，如果使用该依赖系统该如何引入使用？

<script src**=**"jquery.js"></script>

<script src**=**"jquery-ui-0.js"></script>

<script src**=**"jquery-ui-1.js"></script>

<script src**=**"jquery-ui-2.js"></script>

<script>

// your code

</script>

A： 此种情况可以采用NEJ.config进行依赖文件的配置，然后在define.js路径后面配置d参数来载入依赖配置信息，具体操作步骤如下：

1. 配置文件（a.js）中定义第三方库文件的依赖关系

NEJ.config({

'{pro}jquery-ui-0.js':'{pro}jquery.js',

'{pro}jquery-ui-1.js':'{pro}jquery.js',

'{pro}jquery-ui-2.js':'{pro}jquery-ui-1.js'

});

1. define.js文件增加配置参数d=a.js

<script src**=**"/lib/define.js?d=./a.js"></script>

1. 使用依赖文件

NEJ.define(['{pro}jquery-ui-0.js',

'{pro}jquery-ui-2.js'],

**function()**{

// TODO

});