WOW SPG 系统设计

2015-01-04
陆金谭

引言

系统概述

wowspg是一套使用TypeScript编写的单页面前端框架，旨在帮助开发者快速搭建一套高效快速的前端单页面应用。同时配套编译工具wowbuilder，帮助开发者按照传统的开发模式便可快速开发出一个单页面应用。

系统编写背景

直达号平台控制台页面为一套单页面网站应用，由于框架本身基于backbone于两年前编写，代码架构不叫陈旧已经不能很好的满足产品快速迭代的需求。现行代码多人接手，规范不尽完善，系统架构比较复杂，导致代码冗余度较高学习成本很大。同时在多人协作开发单页面应用的的时候，老版框架对代码组织结构不尽完善，导致在上线的时经常出现排队上线，代码合并等问题。为了解决以上代码问题。梳理了多种线上单页应用的共性问题，编写了wowspg单页框架，以帮助开发者快速的搭建高效、规范的单页应用。

术语定义及说明

名称	含义
Block	页面片段，一个页面有n个页面片段组成，页面片段之前存在着继承与依赖等关系
Router	url的路由规则，路由匹配的是url和多个block
Tpl(template)	前端模板
DS(data-source)	模板的数据源，及用来渲染模板的数据，通常为发送异步请求从server取回的数据
DT(data-transfer)	数据转换器，有时从server端取回的数据不合适来直接渲染模板，需要通过数据转换器过滤一遍
Handler	页面逻辑处理器，处理页面逻辑的处理程序

设计概述

需求概述

wowspg单页面架构主要实现以下功能：

1. url 到 页面逻辑的路由管理
2. 页面url跳转的全局控制
3. 页面各功能模块的有效拆分
4. 页面server端数据源的获取
5. 各页面资源的加载和初始化工具（页面css，js，tpl等资源的模块化加载和初始化）
6. 历史记录的管理（历史记录的添加、查询、替换等）
7. 错误管理器（处理路由匹配、页面渲染时的错误）
8. 代码模块化拆分，解决单页面中各个子页面的耦合问题

运行环境概述

wowspg主要运行于主流PC&Mac浏览器中
同时也可运行于Android & IOS 手机浏览器中

条件与限制

单页面的路由支持，在基于webkit或Gecko等支持History API的浏览器内核开发的浏览器（占比约70%以上）中，支持path&hash的路由，同时支持历史记录的缓存操作

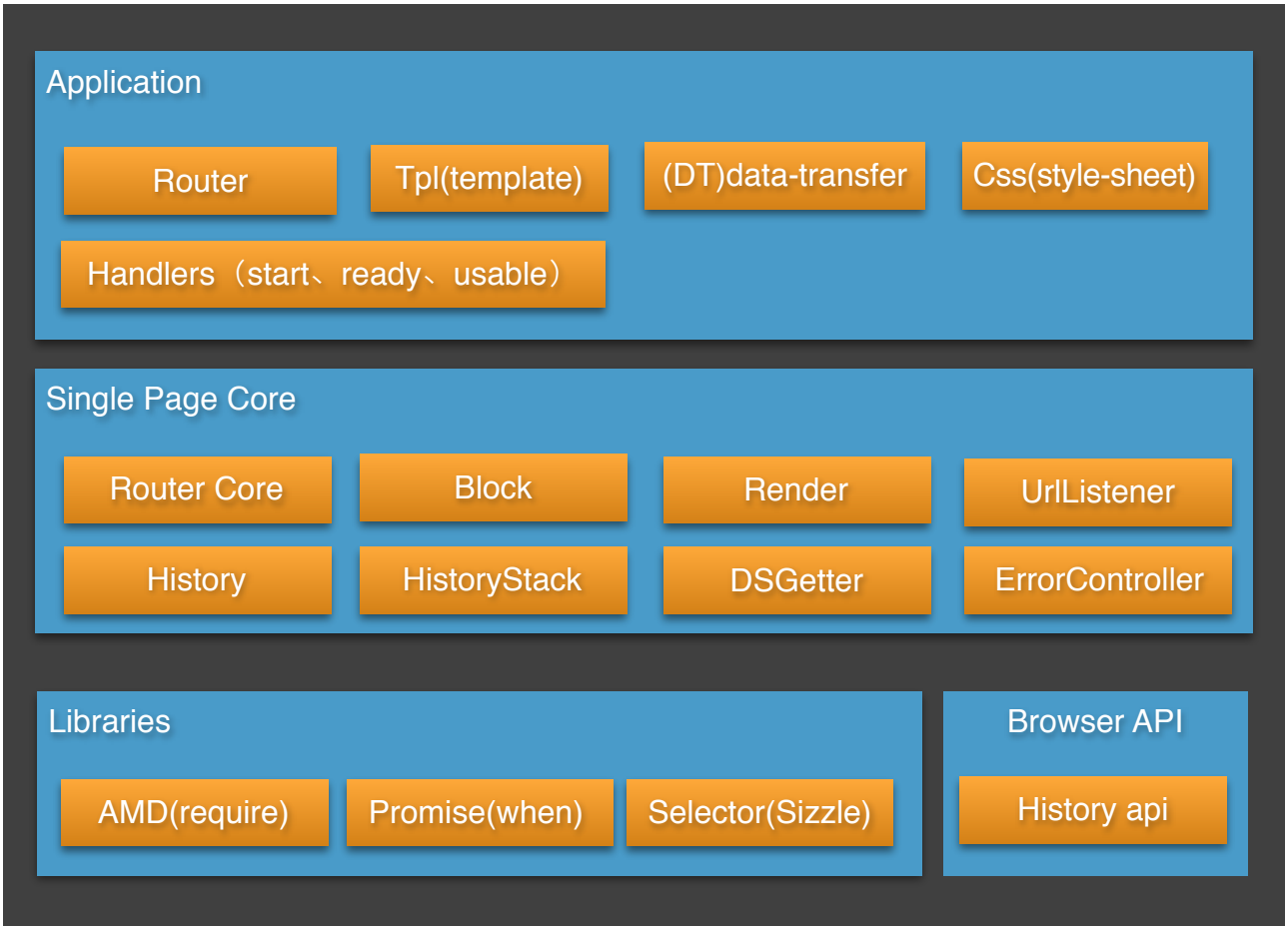
其他浏览器（如，IE6/7/8）下，只能支持通过hash进行路由

开发工具&环境

wowspg使用TypeScript进行开发，需要发布时将TypeScript编译成Javascirpt。
开发时建议使用WebStorm或者visual studio 进行开发，便于TypeScript的调试工作。

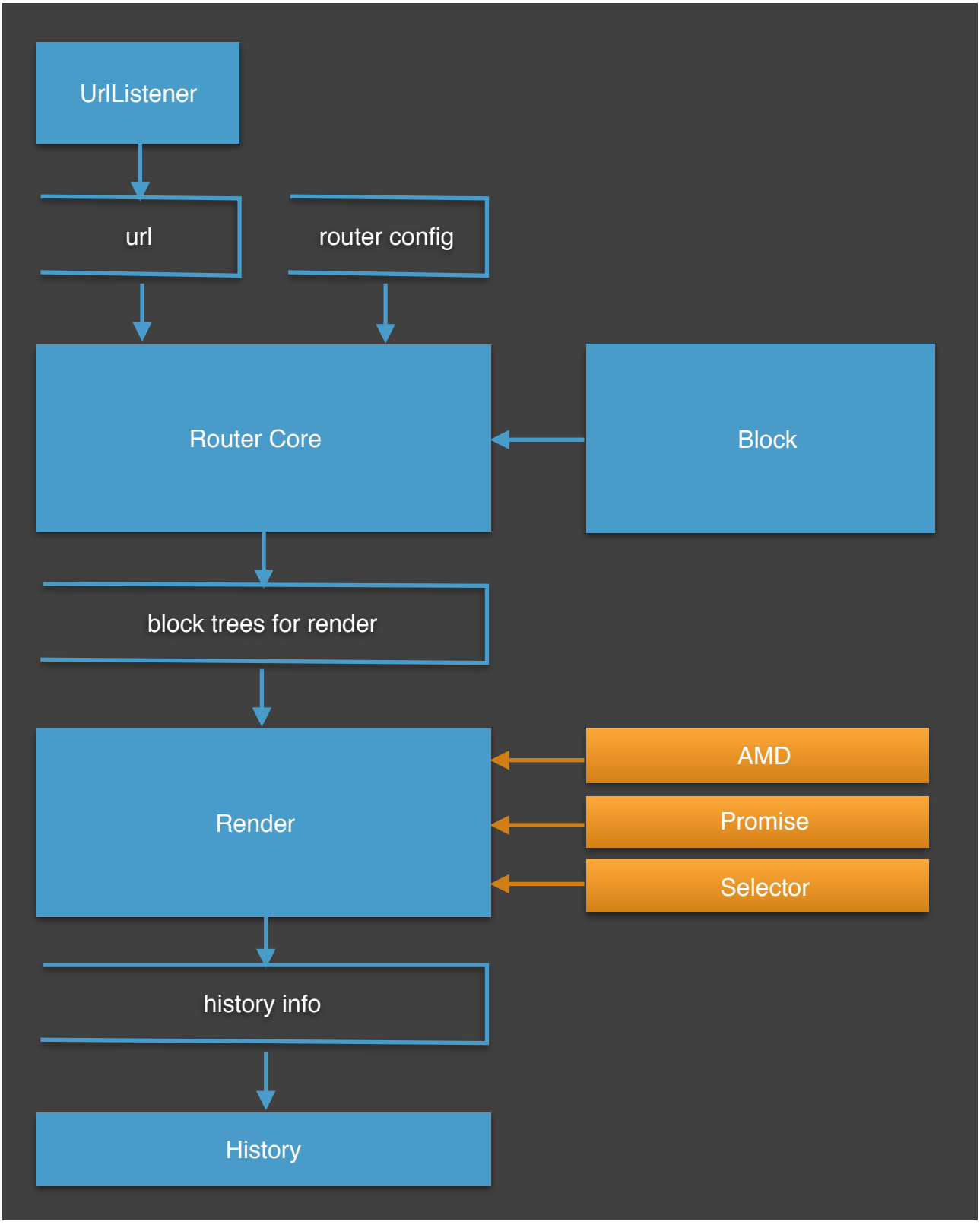
系统详细设计

单页应用基础架构



1. 应用程序接口（Application）：开发者进行Block开发时需要实现的接口定义。主要包含：路由配置、Block 的前端模板、数据转换器、模板样式文件、模板逻辑处理器（其中包含block初始化即要执行的逻辑处理、block初始化完成时需要执行的处理器和block完全加载完是执行的处理器）
2. 单页核心程序（single page core）：单页面架构的核心架构。主要包含：前端路由核心处理器、Block初始化程序、Block渲染器、页面更换的监听程序、页面历史缓存程序、历史记录处理器、DS处理器、错误处理器。
3. 核心库（Libraries）：功单页面的核心程序调用的核心库。主要包含：AMD加载器（如：require），Promise处理器（如：when）、Css选择器支持（如：Sizzle or JQuery）。核心库可制定为实现AMD、Promise、Selector的其他基础库代替。
4. 浏览器接口（Browser API）：单页核心架构主要使用的浏览器API，其中主要包括：History API。

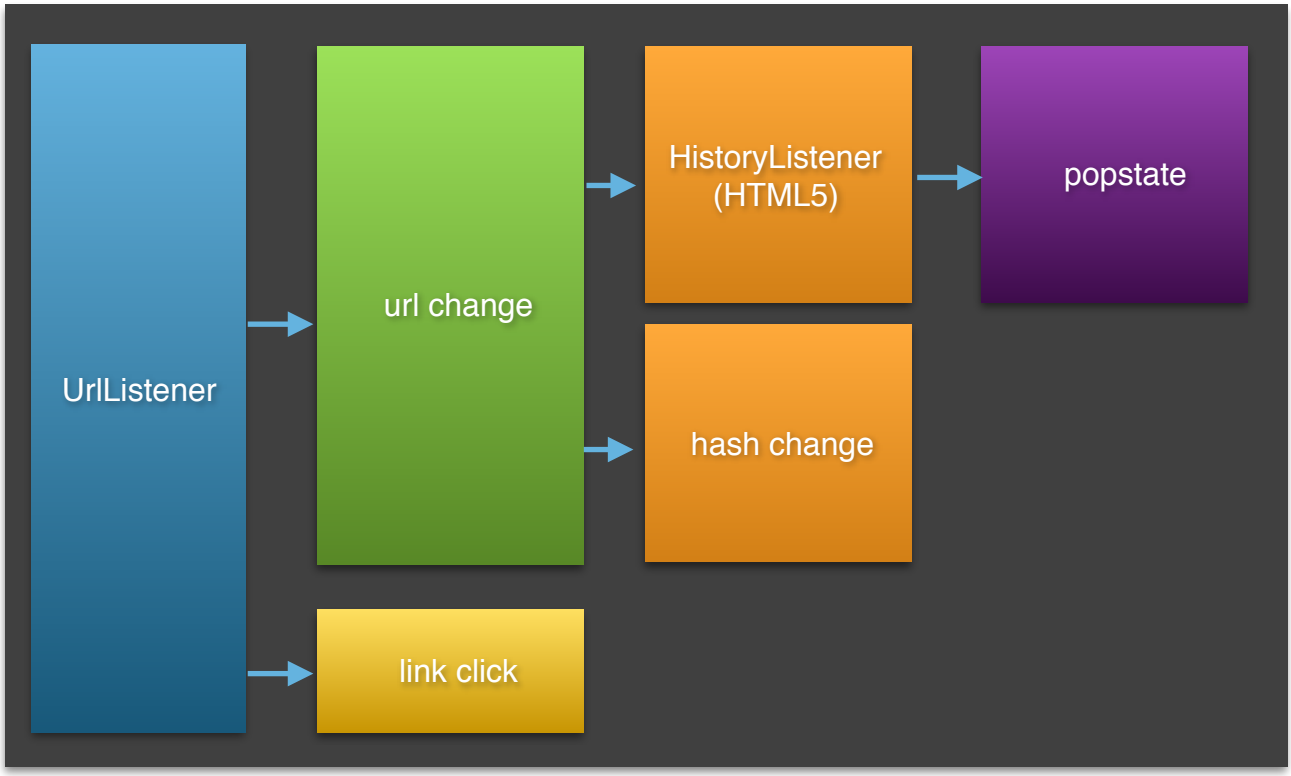
单页核心程序子系统关系



子系统设计

UrlListener Url 监听器设计

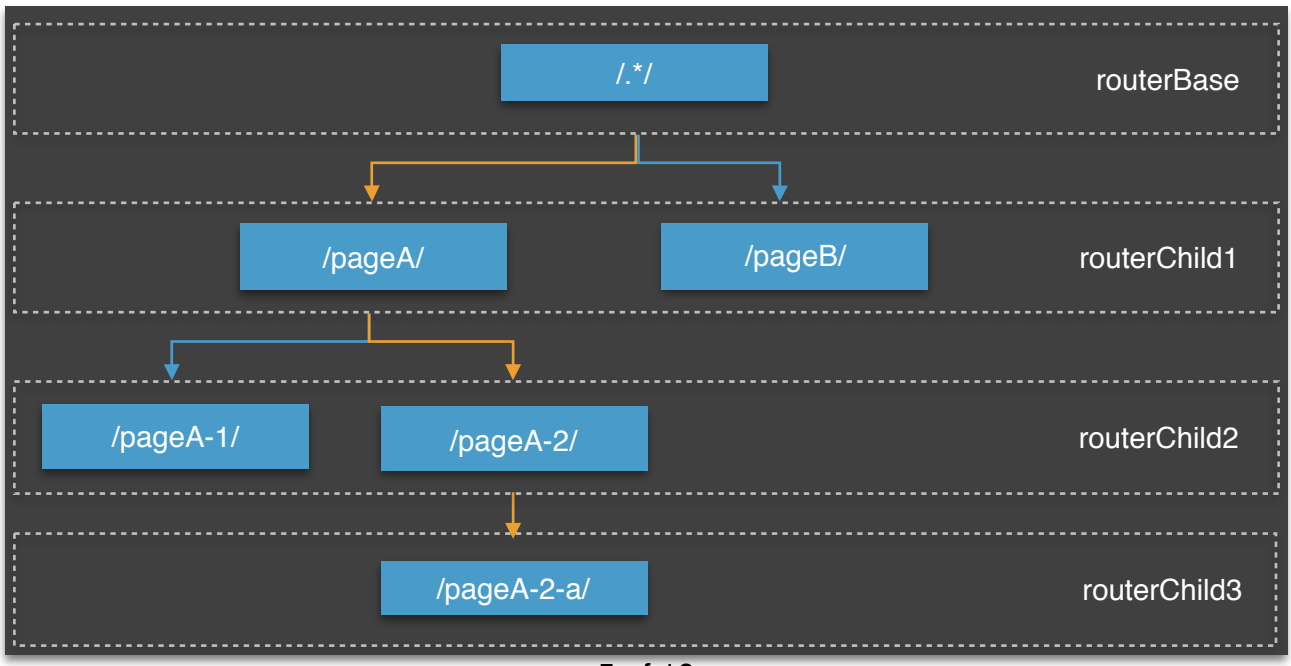
UrlListener主要工作是监听页面跳转行为，处理需要render的url，调起页面渲染行为。



1. UrlListener主要包含对于url改变的监听和链接点击的监听
2. url change为处理历史记录的变更（浏览器后退、前进），在支持HTML5 History API的浏览器环境下，使用history API，监听popstate时间。非H5浏览器环境，使用hash进行单页控制
3. link click事件，为a标签点击事件监听，当链接没有禁止跳转，则将点击链接进行路由匹配。

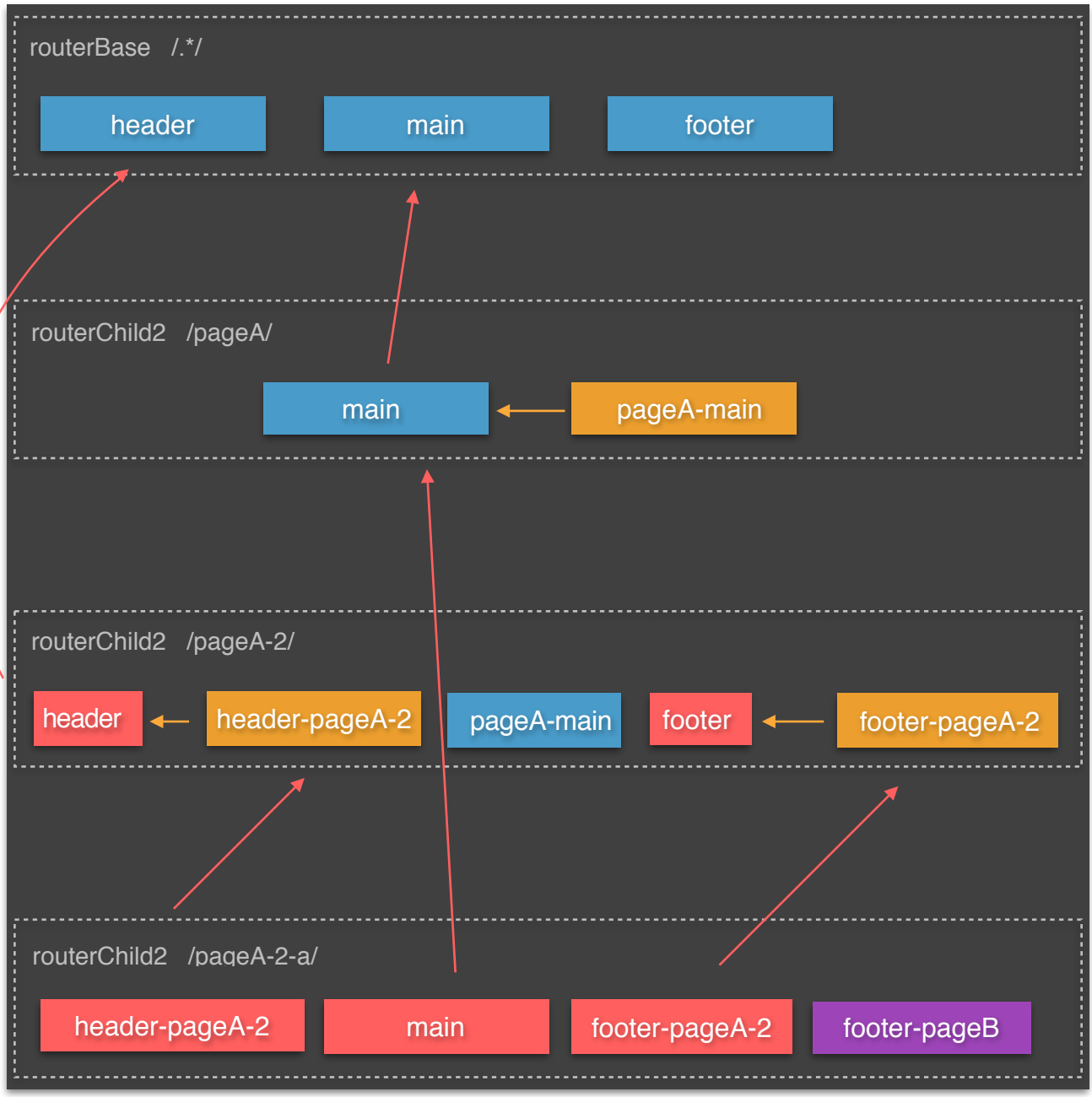
Router Core 路由处理器

路由处理器，主要的工作是，将需要渲染的url进行router匹配，获取要进行render的Block的信息，路由结构是一个树形拓扑结构，示例(/pageA/pageA-2/pageA-2-a)如下：



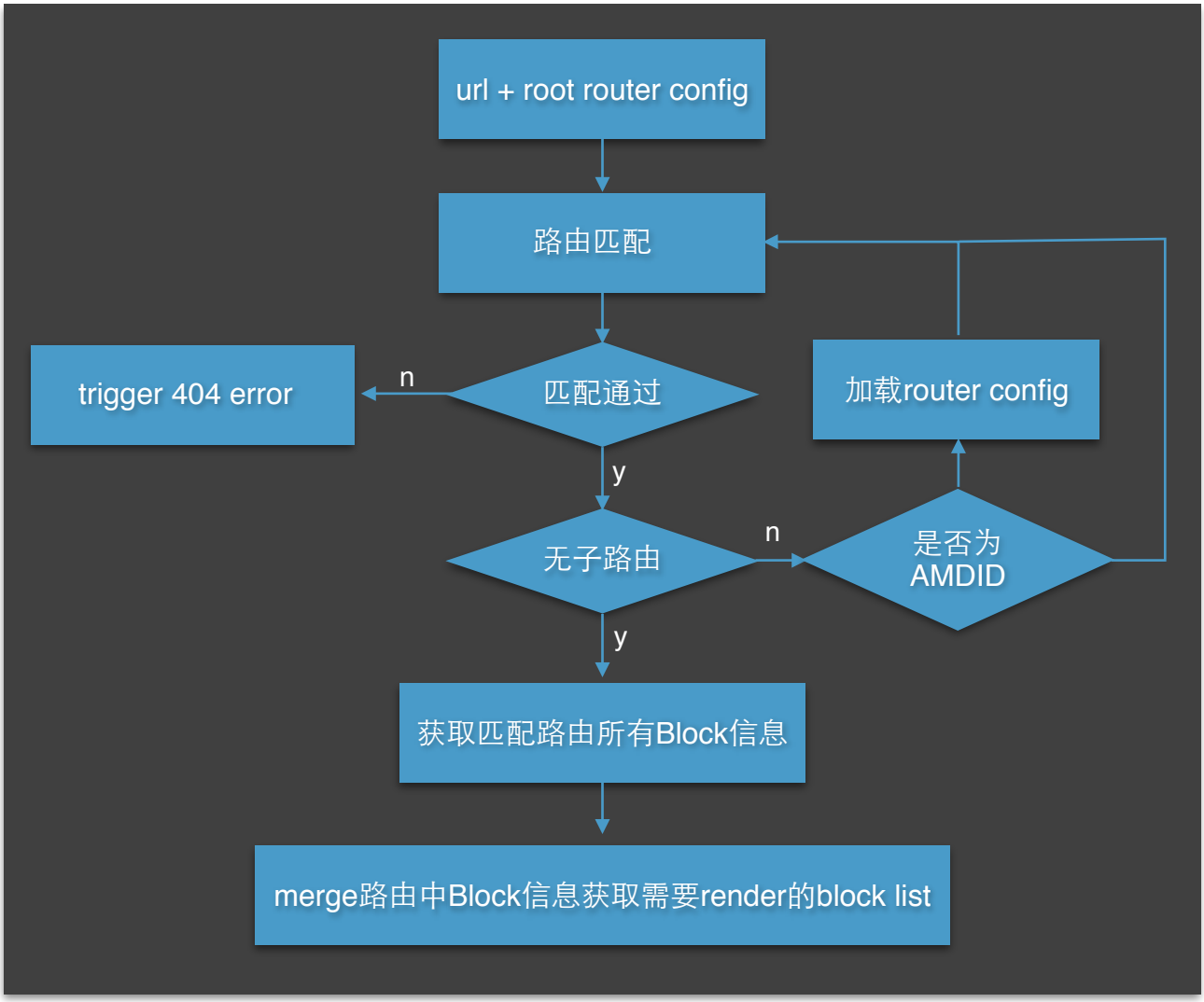
1. 路由规则受用正则匹配url
2. 上一级路由配置指向下一级路由配置，下一级路由配置可以直接写在配置文件，也可以指定一个amdID，在路由匹配阶段会异步加载下一级router config
3. 每一级路由配置同时配置了router下block的基本信息，每一个router 对应n个block。

上面router对应如下Block结构：



1. 每一层router描述所属页面的block配置
2. 所有block必须从属于父级或祖先router暴露的block，类似于smarty模板的继承，否则block不会被渲染，实例中的紫色block。
3. 父级router可在现有block下添加新的block供下一级覆盖，示例中的黄色block
4. 如示例所示，url '/pageA/pageA-2/pageA-2-a', 会匹配到四个block：'header-page-2', 'main', 'footer-pageA-2', 'footer-pageB'。页面渲染时最终只会渲染红色的block。

路由处理器执行流程如下：



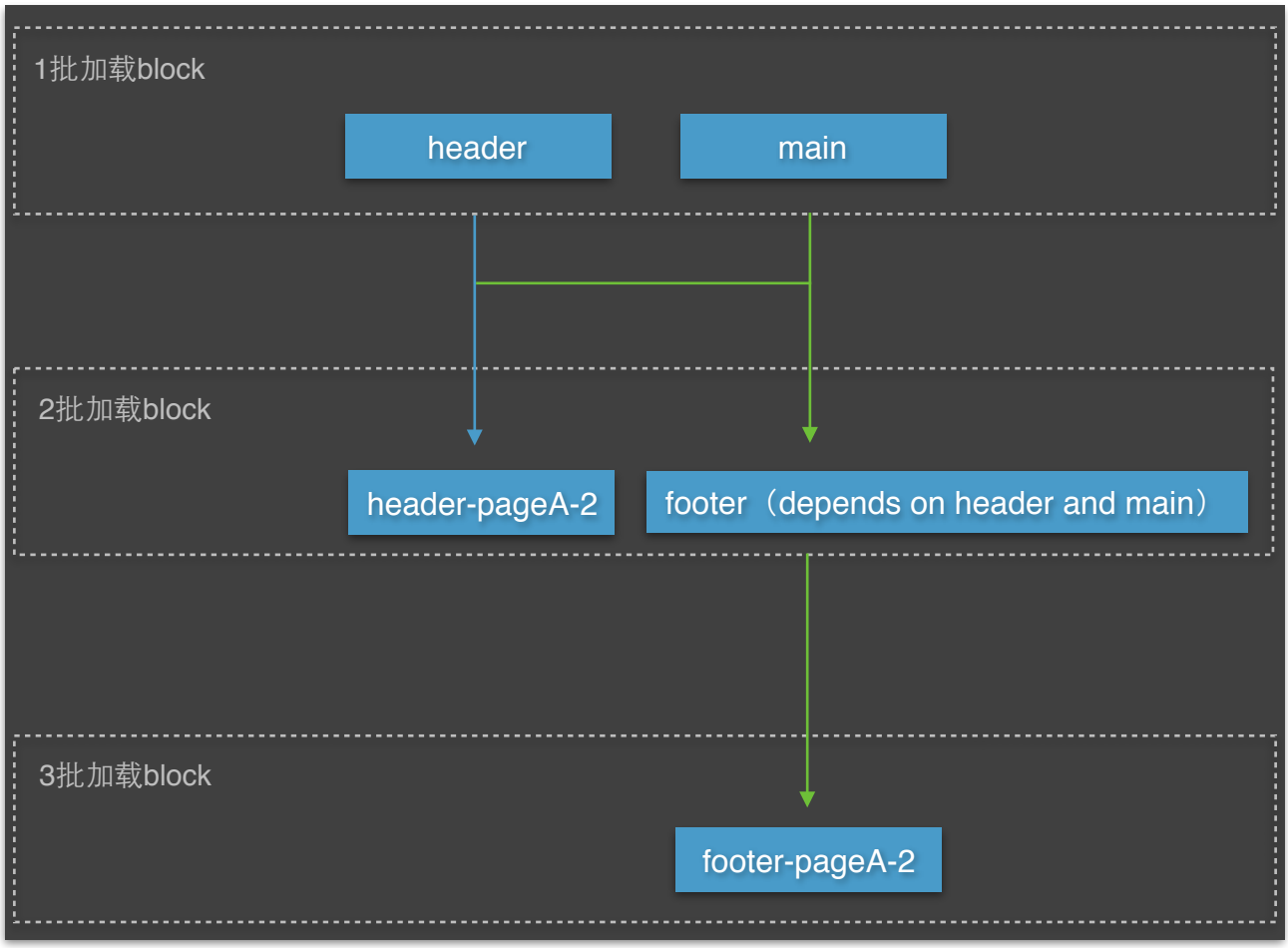
Render子系统设计

该系统主要完成，从Router系统中获取block list信息，并分别调起各Block的渲染操作。是页面渲染的总调度器。

Block之间存在继承和依赖两种关系，一个Block只能依赖于同一级router的其他Block。
Block的render顺序取决于这两种关系，规则为：

1. 先渲染父级block，后渲染子集block
2. 当该模块所依赖的所有模块都已完成渲染，才能开始该模块的渲染

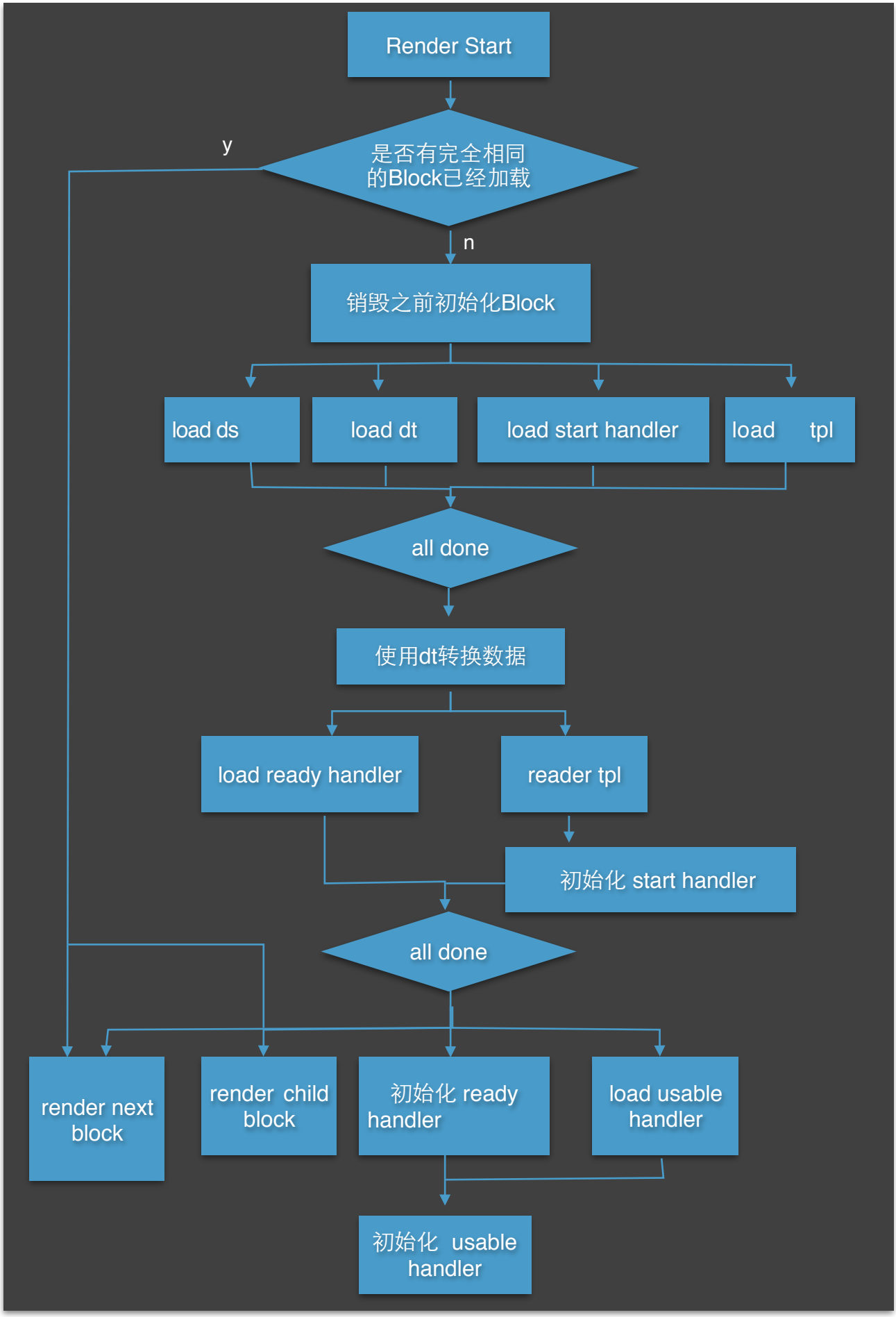
示例如下，假设footer依赖于header和main：



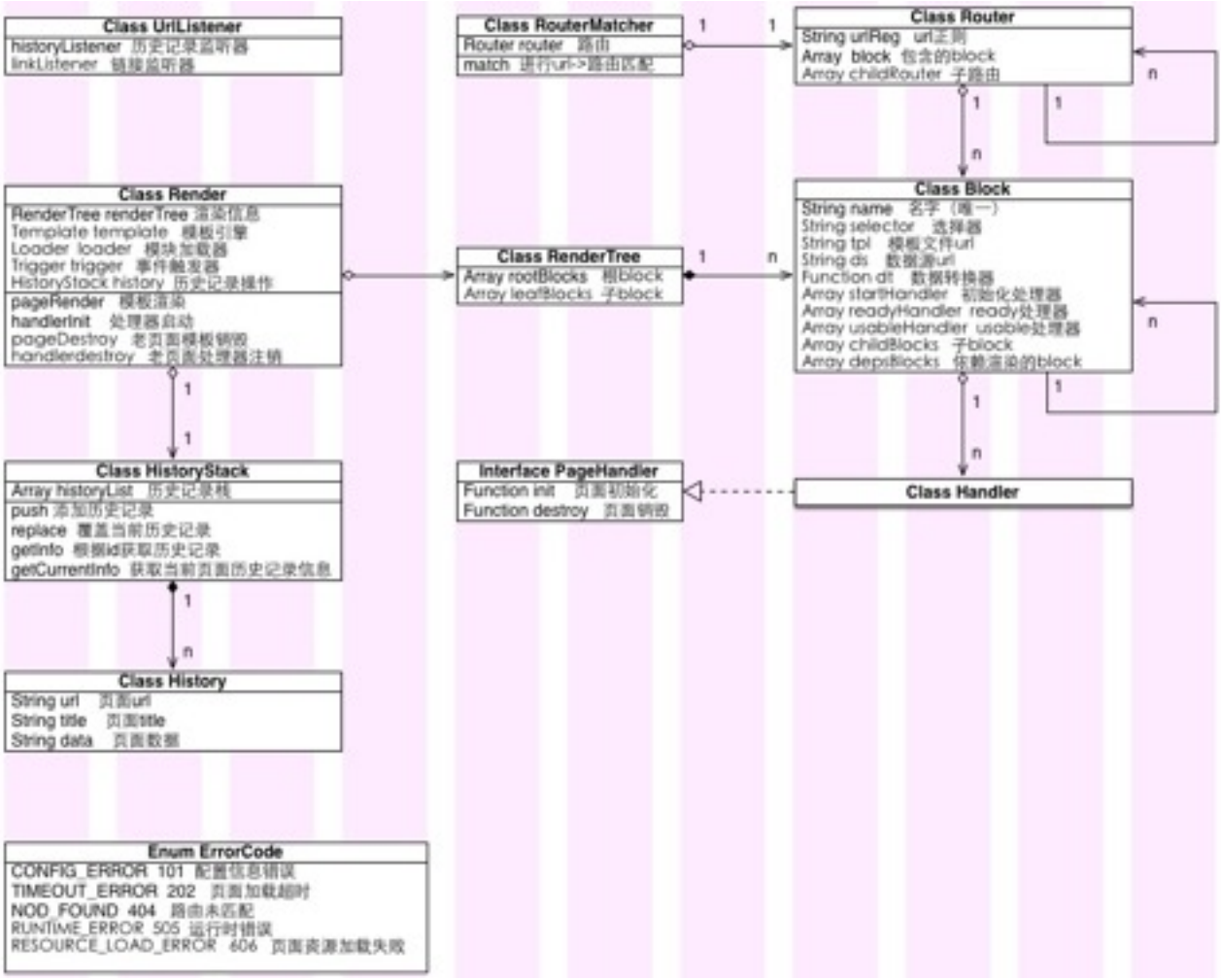
1. 最优先会加载header、main、footer3个block
2. 由于footer依赖于 header和main，所以会等待header 和 main加载完再进行加载
3. 其余部分按照父级优先加载的顺序执行加载

具体进入到每一个Block模块的加载时，将执行以下流程：

1.


Render Start
是否有完全相同
的Block已经加载
y
n
销毁之前初始化Block
load ds
load dt
load start handler
load tpl
all done
使用dt转换数据
load ready handler
reader tpl
初始化 start handler
all done
render next
block
render child
block
初始化 ready
handler
load usable
handler
初始化 usable
handler

系统建模



相关资料:

TypeScript Tutorial: http://www.typescriptlang.org/Tutorial
wowspg git: https://github.com/cloud-fe/wowspg
Require Js: http://requirejs.org/
when.js: https://github.com/cujojs/when/
Sizzle: www.sizzlejs.com
wowbuilder git: https://github.com/cloud-fe/wowbuilder