平台

2014年1月

前端框架简介

WLJ(walk lonely javascript)采用纯js脚本设计，依托于Ext免费版本3.3.1为基础，并结合YC.CRM产品v4.5对于Ext的扩展、补丁研发，采用面向对象的理念，对前端对象进行了大量的对象化封装。

首页框架结合METRO-UI理念，对于UI体验上进行的重新设计，并在此过程中，加强了对于性能、易用性等方面的改造；

功能开发框架中，主要对开发人员逻辑开发场景进行分析，结合开发人员中存在的对面向对象的理解不深、对浏览器运行机制的不了解，以及代码调试困难等问题，进行了专项封装，如：代码过程化、片段化、扁平化处理；提供一个可见易用的控制台，以及相应的日志API等。

技术架构简介

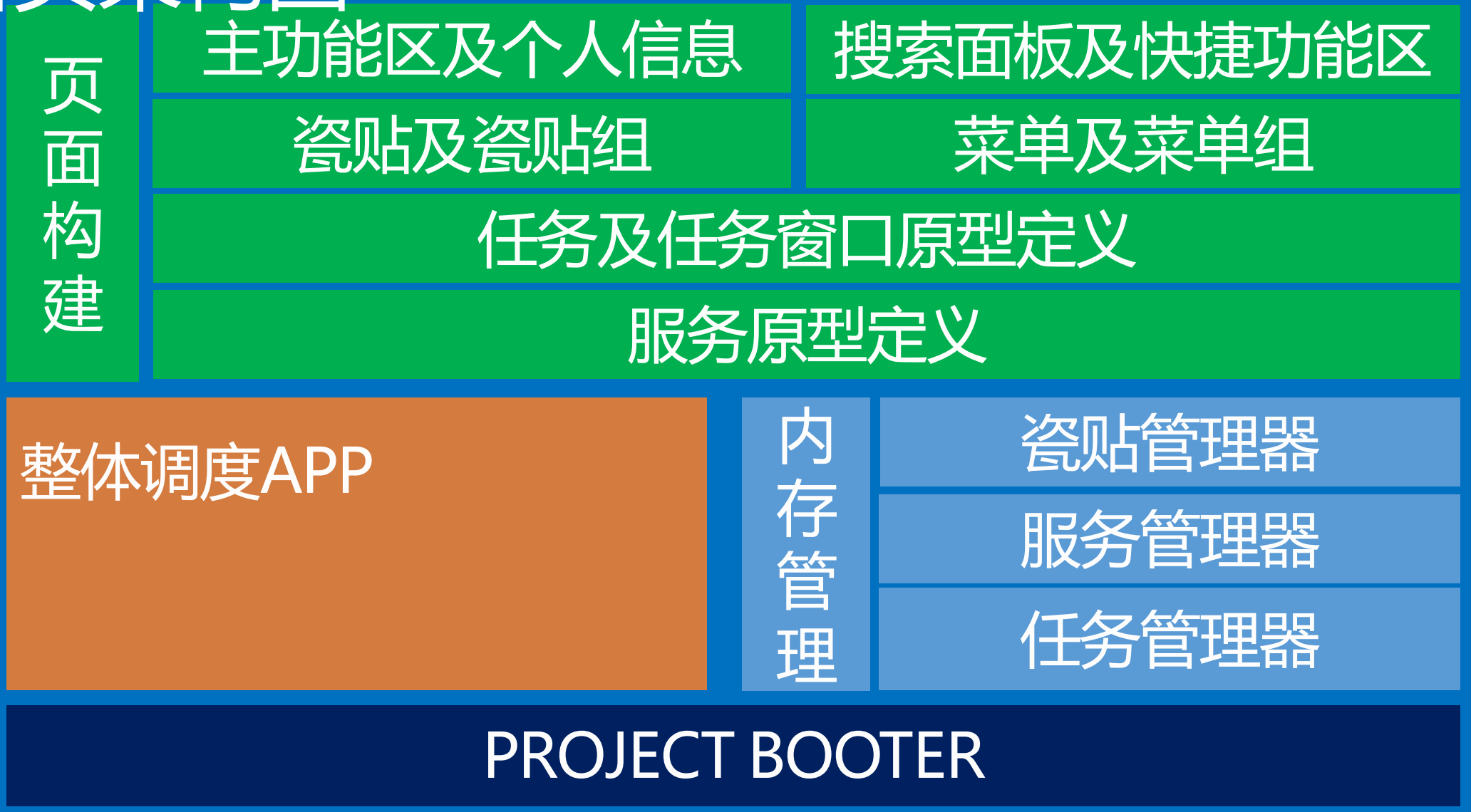
WalkLonely Javascript

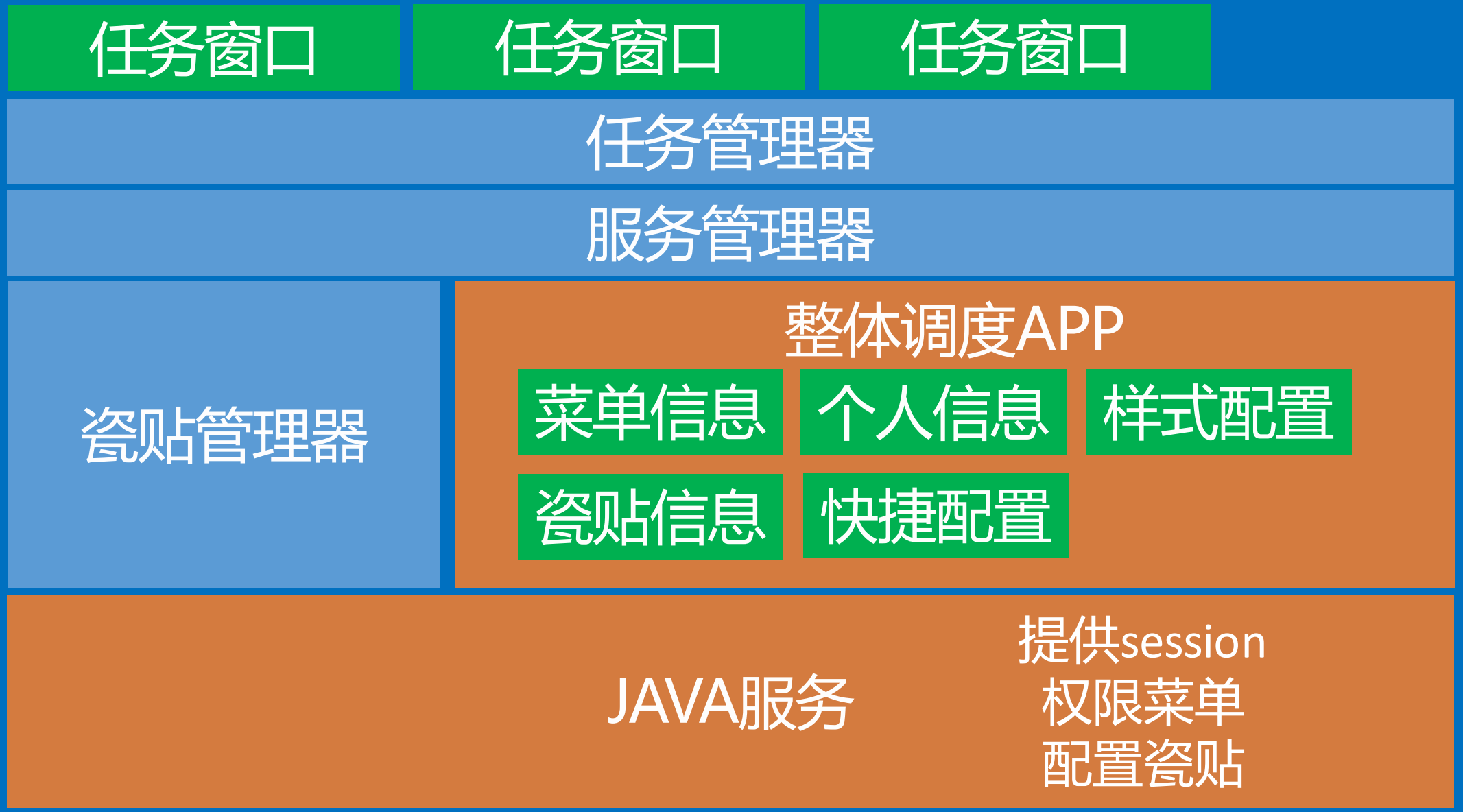
首页瓷贴框架介绍

开发框架

后续工作

首页架构图


页面构建
主功能区及个人信息
搜索面板及快捷功能区
瓷贴及瓷贴组
菜单及菜单组
任务及任务窗口原型定义
服务原型定义
整体调度APP
内存管理
瓷贴管理器
服务管理器
任务管理器
PROJECT BOOTER


任务窗口
任务窗口
任务窗口
任务管理器
服务管理器
瓷贴管理器
整体调度APP
菜单信息
个人信息
样式配置
瓷贴信息
快捷配置
JAVA服务
提供session
权限菜单
配置瓷贴

页面构建原型

主功能区及个人信息

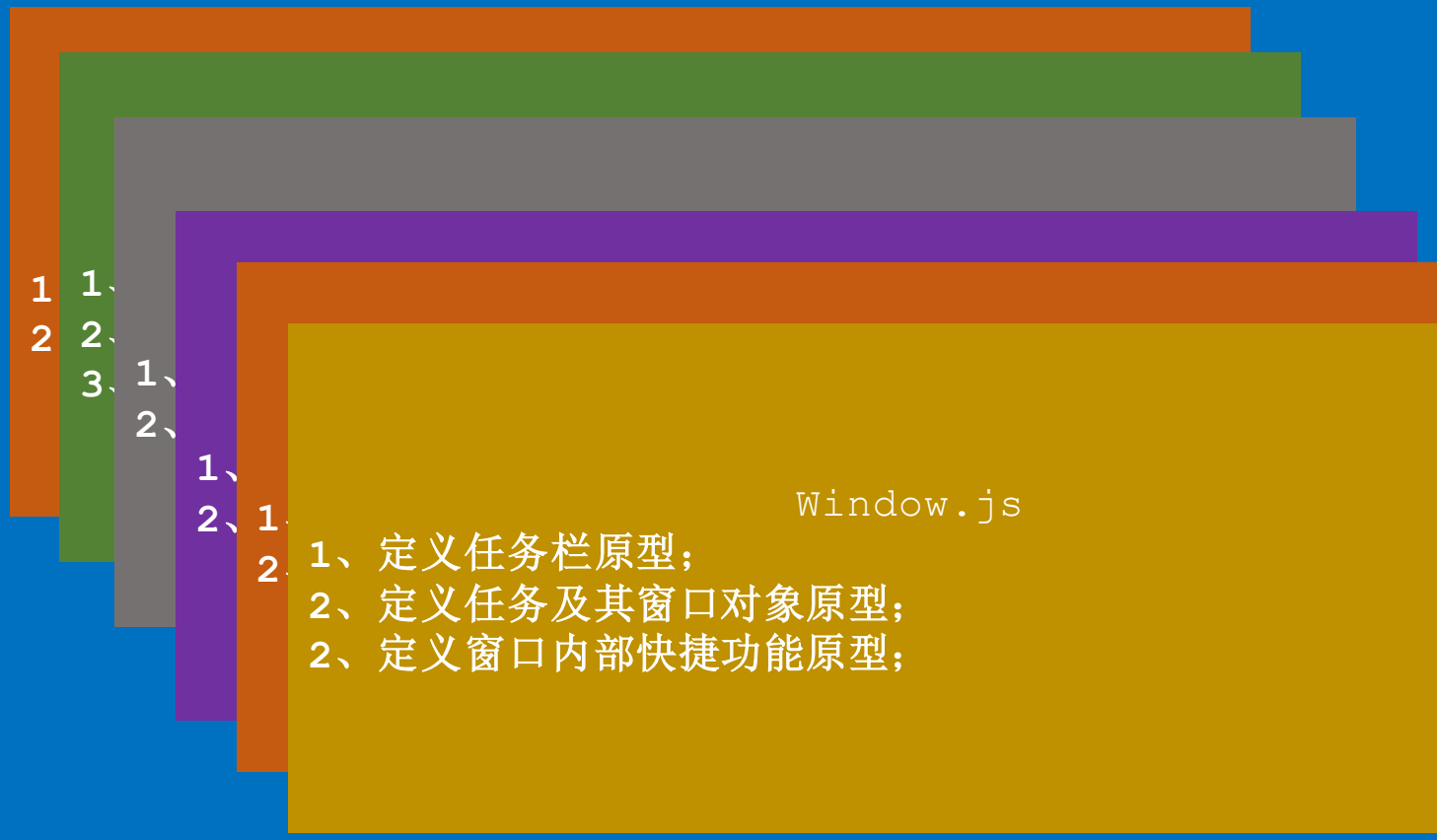
瓷贴及瓷贴组

搜索面板及快捷功能区

菜单及菜单组

服务原型定义

任务及窗口原型定义



内存管理

瓷贴管理器（Wlj.TileMgr）

渲染队列

1、以同步队列的方式逐个请求渲染代码并渲染内容；
2、渲染队列结束后，自动启动数据队列；
3、动态创建、销毁瓷贴同步销毁队列数据；

数据队列

1、数据队列配置，由动态瓷贴代码动态添加；
2、数据队列以同步方式逐个请求；
3、数据配置在受控对象销毁时，同步销毁；

内存管理

服务管理器（Wlj.ServiceMgr）

- ❖可提供独立功能的一个页面URL被定义为一个服务
- ❖以菜单ID（resId）为key
- ❖服务列表在首页初始化之前构建；展示窗口在第一次点击时构建
- ❖服务对象负责任务对象构建；
- ❖所有新建人物窗口的操作，均通过服务管理器查找到相关服务的方式打开任务窗口；

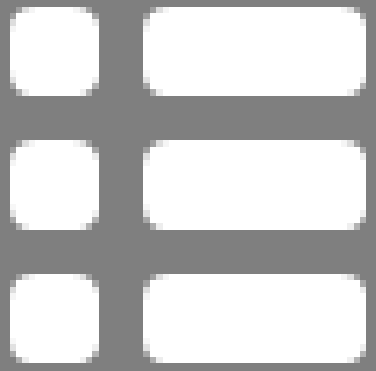
内存管理

任务管理器（Wlj.TaskMgr）

- ❖任务窗口由服务对象创建
- ❖任务对象包括窗口对象、frame对象、以及快捷操作栏；
- ❖任务对象在构建过程中会在任务管理器中登记注册；
- ❖任务管理器提供一个可视当前任务列表界面，在界面中可关闭任务

技术架构简介

WalkLonely Javascript



首页瓷贴框架介绍

开发框架

后续工作

面临问题及分析

问题表现

- 代码量大
- BUG频出
- 复用率低
- 上手门槛高

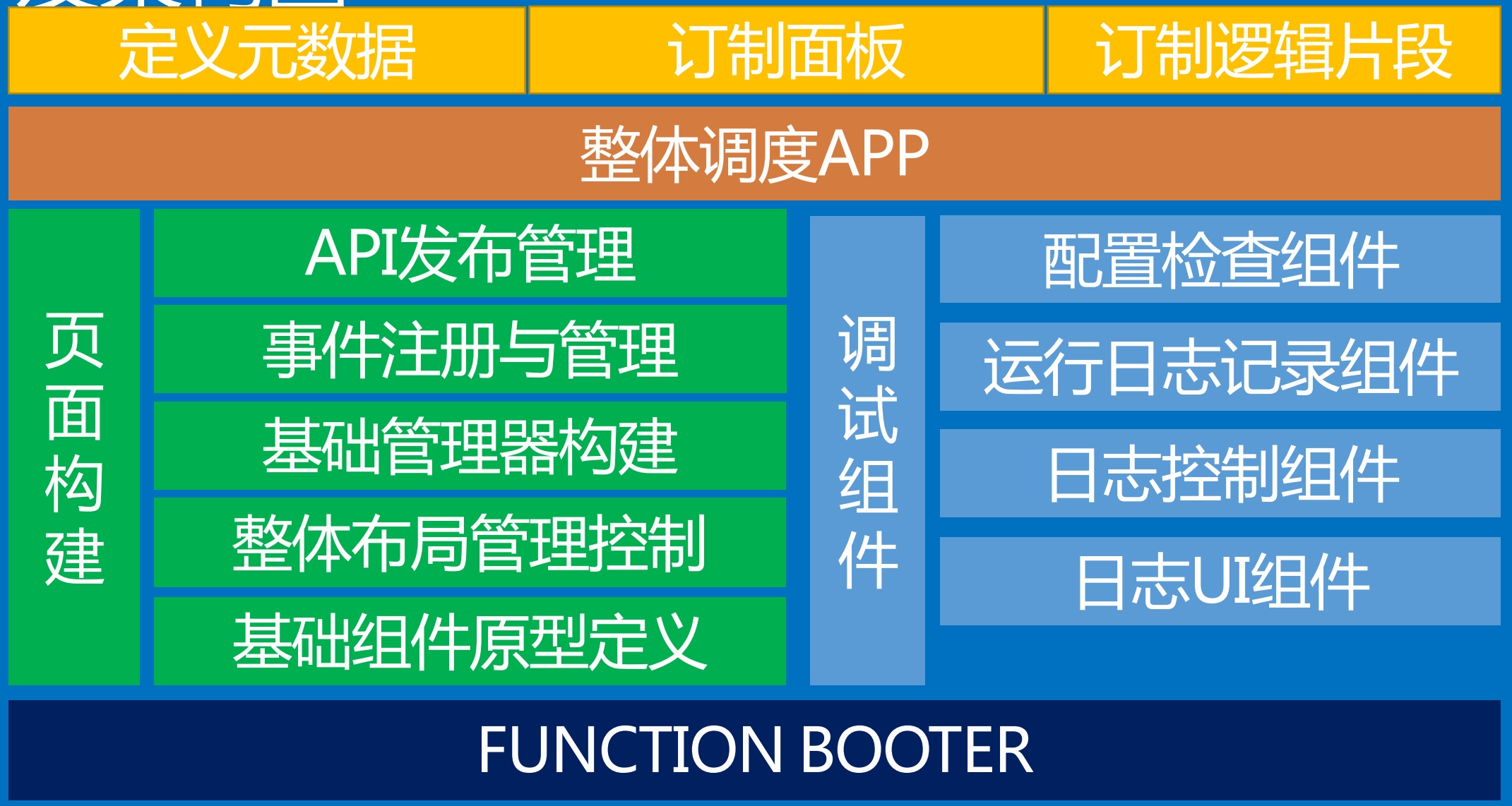
问题原因

1、面向过程和面向对象
2、JS、浏览器等机制的理解有难度
3、前端代码有框架，无架构
4、调试成本高

解决方案

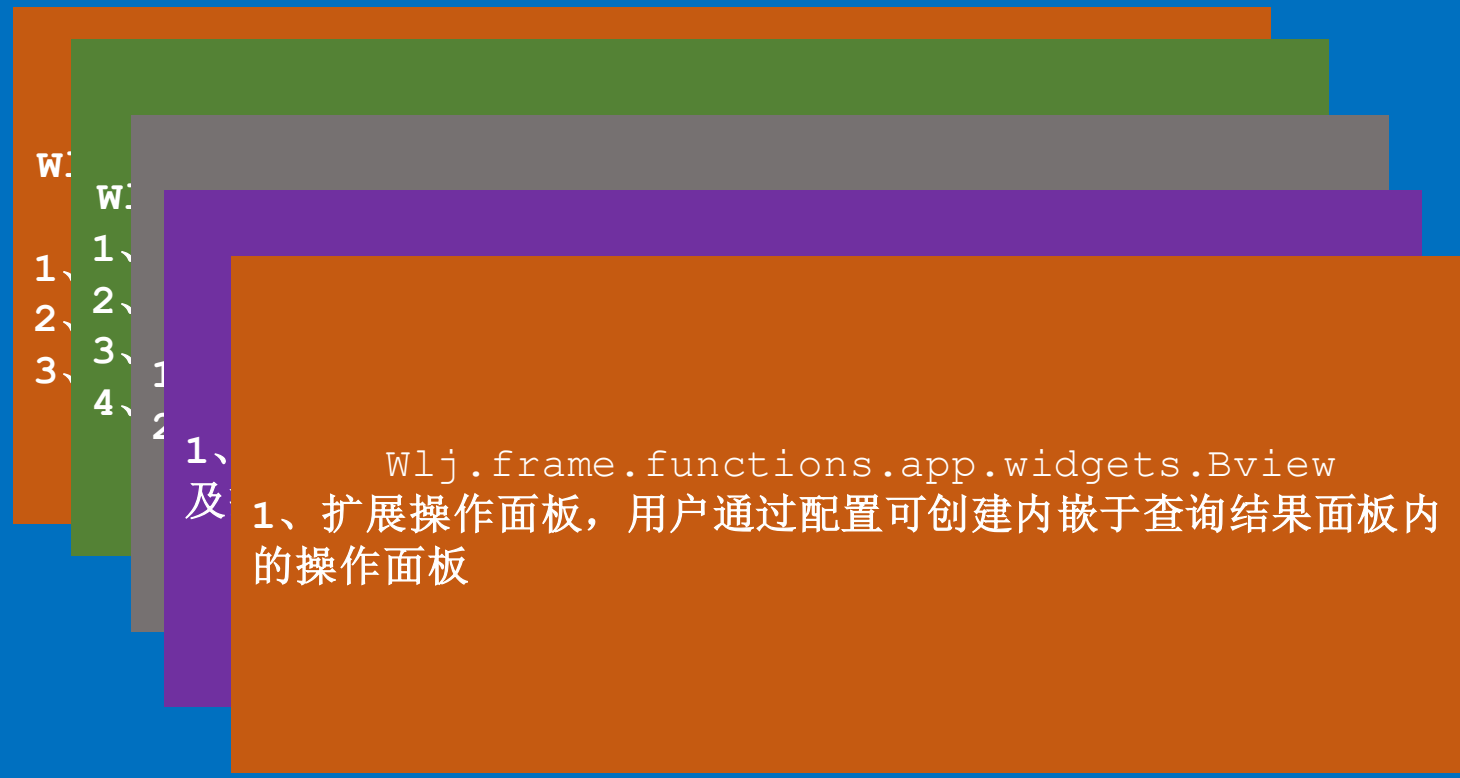
- 业务代码过程化、结构化
- 订制配置项以及逻辑原型
- 提供业务代码校验机制
- 构建统一作用域的API方法
- 提供日志API和控制台

开发架构图


定义元数据
订制面板
订制逻辑片段
整体调度APP
页面构建
API发布管理
事件注册与管理
基础管理器构建
整体布局管理控制
基础组件原型定义
调试组件
配置检查组件
运行日志记录组件
日志控制组件
日志UI组件
FUNCTION BOOTER

基础组件原型定义

- 页面查询条件面板类
- 页面查询结果列表容器
- 数据行渲染对象
- 系统操作面板基类
- 扩展操作面板基类



整体布局管理控制

- 页面整体默认采用border布局；
- 查询条件和查询结果面板占据中部；
- 查询条件根据所需条件字段行数自动计算高度；
- 查条件列数据根据面板宽度计算；
- 上下左右面板各有默认配置，开发人员可自定义；

基础管理器构建

数据字典管理器

树形结构管理器

TreeManager

1、树形结构管理器为单态对象；
2、树形结构管理器负责树形面板管理及其数据源管理；
3、树形结构管理器在第一次调用时初始化；

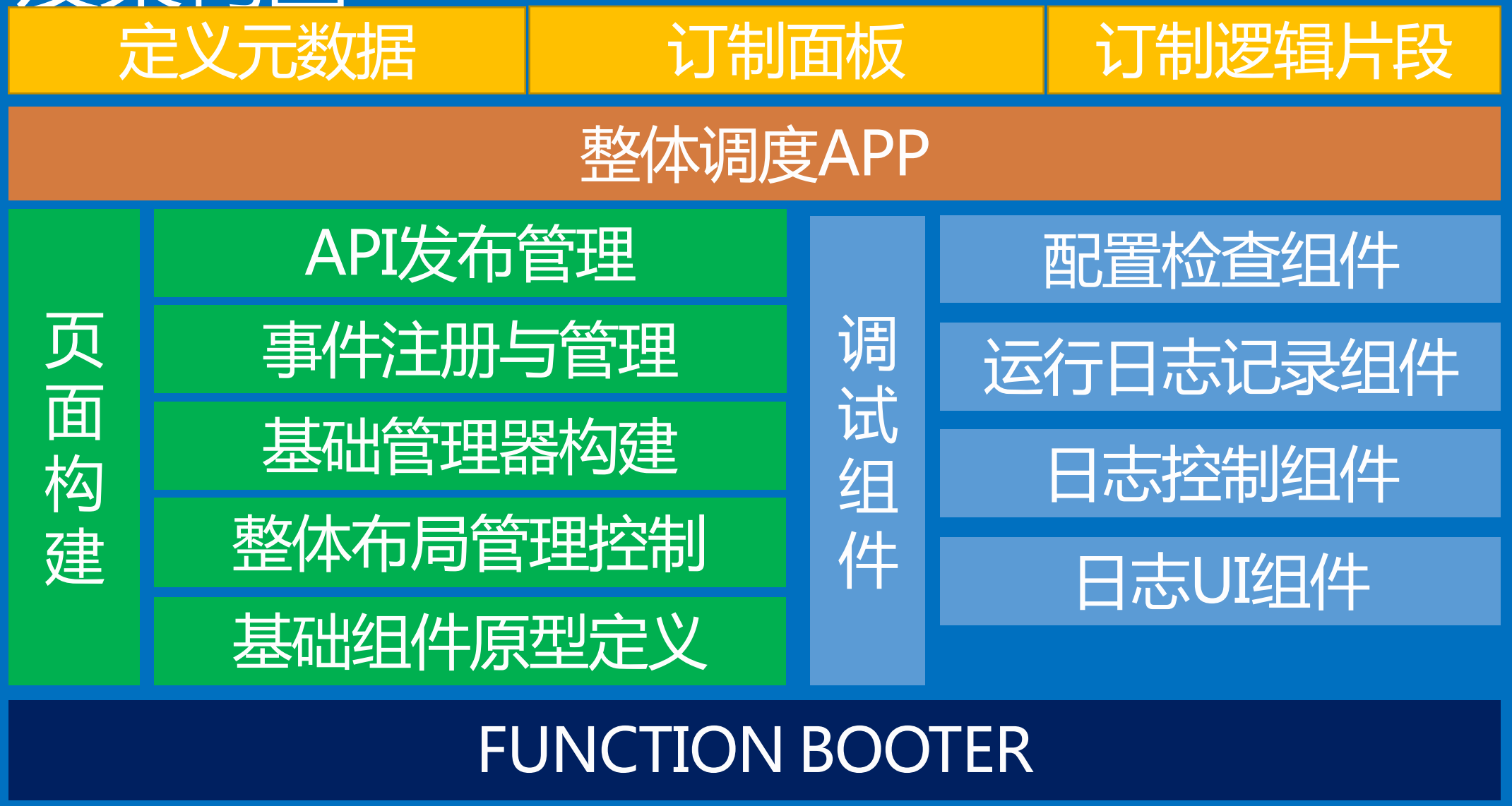
事件注册与管理

- ❖所有事件监听添加在APP对象上，并由内部组件事件触发
- ❖系统可用事件入口函数可见于Wlj-frame-function-header.js文件总，listners对象；
- ❖所有事件触发、调用日志可见于控制台；
- ❖目前，系统提供47个可用事件；18个待扩展事件；

API发布管理

- ❖API方法和配置项分别定义于Wlj-frame-function-api.js和Wlj-frame-function-header.js文件；
- ❖API方法均由APP对象发布，其作用域为window对象，可直接调用；
- ❖配置项原型均有默认配置，以过程化代码进行自定义；

开发架构图


定义元数据
订制面板
订制逻辑片段
整体调度APP
页面构建
API发布管理
事件注册与管理
基础管理器构建
整体布局管理控制
基础组件原型定义
调试组件
配置检查组件
运行日志记录组件
日志控制组件
日志UI组件
FUNCTION BOOTER

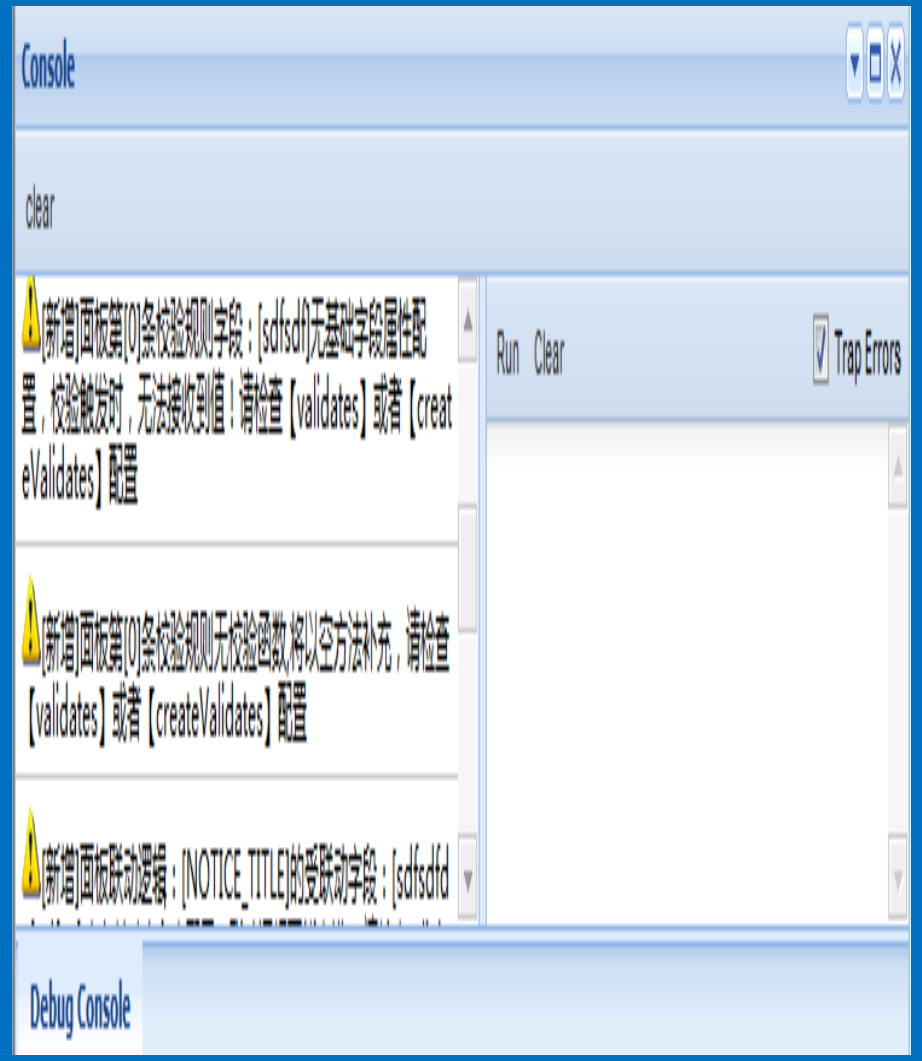
调试组件

调试组件：提供一个可见于页面的控制台组件，开发人员可在开发过程中根据级别开启相应日志显示功能辅助代码开发和调试。

- 配置检查组件
- 运行日志记录组件
- 日志控制组件
- 日志UI组件

日志UI组件

UI组件：
1、定义于debug.js文件；
2、UI组件主要包括一个控制台，和脚本运行窗口；
3、UI组件构建并发布日志调用API，包括：Ext.log、Ext.warn、Ext.error，开发人员可在自己的逻辑function内部调用。



日志控制组件

控制组件：
1、定义于Wlj-frame-function-error.js文件；
2、控制组件主要控制各级别日志记录是否开启。包括：ERROR、WARN、INFO三个级别；

```
Wlj.error.debugerConfig = {
    ERROR : false,
    WARN : true,
    INFO : false,
    logLineLimit : 100,
    initLogTool : function(){
        if(!Wlj.error.debugerConfig.ERROR){
            Ext.error = Ext.emptyFn;
        }
        if(!Wlj.error.debugerConfig.WARN){
            Ext.warn = Ext.emptyFn;
        }
        if(!Wlj.error.debugerConfig.INFO){
            Ext.log = Ext.emptyFn;
        }
    }
};

JDEBUG = Wlj.error.debugerConfig;
```

日志开关关闭时，日志API的内部逻辑会被清除。

日志组件加载级别最高，业务逻辑代码任何时机均可调用。

运行日志记录组件

运行日志：
1、运行日志主要包括：事件触发和调用日志，组件内部逻辑日志；
2、事件触发日志由APP统一处理；
3、组件内部逻辑日志由各个组件各自调用日志API记录；

```
Wlj.frame.functions.app.App.prototype.fireEvent = function(){
    var eventName = arguments[0];
    var eventResult = true;
    if(!eventName){
        return eventResult;
    }
    Ext.log('触发事件：【'+arguments[0]+'】:接收到【'+(arguments.length - 1)+'】个参数！');
    if(!this.hasListener(eventName)){
        Ext.log('【'+arguments[0]+'】未绑定逻辑，事件调用结束！');
        return eventResult;
    }
    try{
        for(var i = 0;i<this.events[eventName].listeners.length;i++){
            Ext.log('fn['+i+']:'+this.events[eventName].listeners[0].fn.toString());
        }
        eventResult = Wlj.frame.functions.app.App.superclass.fireEvent.apply(this,arguments);
    }catch(Werror){
        Ext.error('事件【'+arguments[0]+'】：执行出错。TYPE:【'+Werror.name+'】;MESSAGE:【'+Werror.message+'】!');
        eventResult = false;
    }finally{
        Ext.log('【'+arguments[0]+'】事件结束!');
        return eventResult;
    }
};
```

配置检查组件

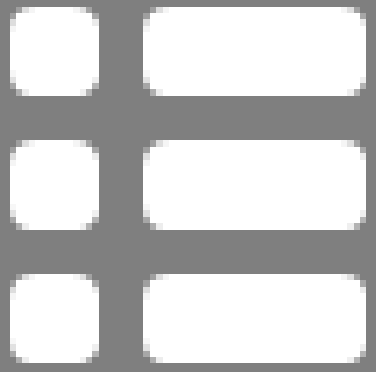
配置检查：
1、定义于Wlj-frame-function-builder.js的过程化代码逻辑；
2、根据菜单参数请求功能点业务逻辑代码；
3、检查关键配置项及其内部结构；
4、根据检查结果构建APP对象；

业务逻辑代码请求包括：标签同步请求模式和Ajax异步请求模式。两种模式会有不同的缓存效果。

关键配置项如配置信息有误，可能叫停APP构建。

技术架构简介

WalkLonely Javascript



首页瓷贴框架介绍

开发框架

后续工作

后续工作

- 环境适应性，主要体现为对多版本浏览器的支持
- 需要落地开发验证框架的业务覆盖度
- 依赖于版本控制的前端缓存版本管控
- UI与体验设计人员的介入
- 基础性能调优
- 知识传播与支持团队培养
- 更多基础设施扩展及完善

结束语

框架的构建与完善，需要大量的落地验证，并形成反馈。产品应该在这样的闭环业务中滚动成长。