**Matrix管理平台技术分析报告**

系统解耦的强度决定了系统架构的高度

## 1、项目背景

为适应公司业务和规模的不断扩大以及不久的将来会出现的大量产品需求，在现有后台无法满足的情况下，尝试逐步重构现有后台业务，将各个子后台系统整合，由Matrix平台统一约束。

公司现有管理后台存在诸多问题罗列如下：

1. 分页相关问题

A：长整型数据与现有分页插件不兼容问题

B：类Guns系统物理分页参数不统一问题

C：运管系统物理分页效率低下

D：查询后分页定位不准确问题

E：类Guns系统表格右侧操作按钮权限无法控制问题

F：列表页页面处理复杂，入手难度高，有学习成本

G：分页参数侵入性高，没有封装

1. 现有项目代码封装性低，项目间代码可复用性差
2. 对第三方框架过度依赖，缺少深度封装，可维护性差，灵活性低
3. 现有后台管理系统缓存使用占比极低
4. 现有管理系统消息提示几乎无法实现(列表页表格插件固有问题)
5. 项目缺少规范化的开发文档-导致代码编写可读性低，问题定位不方便
6. 项目碎片化问题
7. 运管系统ORM框架灵活性问题
8. 运管系统的RPC接口侵入问题，没有进行RPC接口数据转换
9. 运管系统代码规范性问题(js脚本凌乱，html属性使用错误)
10. 类Guns系统权限分离问题(现有多套权限，多套数据库)
11. 数据交互格式不统一问题
12. 没有抽离出统一的管理平台

根据上述问题，建议逐步替换现有后台管理项目。

## 2、新系统优势

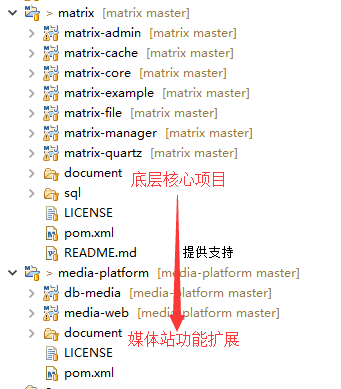
matrix管理平台于两年前重构自某知名电视购物平台，在原有平台基础上加强代码封装性，提高项目间代码解耦水平，降低代码编写难度，剥离非业务代码在服务中的侵入，抽离出一套成熟、稳定、易维护、高吞吐量、高可扩展性的商务解决方案。依托于先进的项目结构设计，对Spring-mvc进行了深度封装，对垂直体系架构进行了横向扩展，极大提高了项目的灵活性。非常适合中大型互联网公司构建灵活多变的大型项目。

matrix平台缓存使用占比高于98%，在双十一高并发压力下，数据库读写操作低于千分之0.15。在20台分布式服务器集群环境下，线上并发峰值10万+，吞吐量4万/秒。

新系统优势如下：

1. leader- slaves模式，leader控制所有子项目(web|job|api|role)
2. 完善的权限系统，不依赖第三方
3. 项目国际化处理 – 以配置文件的方式进行消息处理
4. 项目关键信息配置文件化，不侵入代码
5. 开发者快速入门，利于团队人员更新、迭代；代码结构规范、统一
6. 独立的文件服务器 – 后期可扩展到CDN
7. api平台化管理。支持热熔断降低线上风险|支持跨域配置, 脱离jsonp对代码侵入
8. 定时任务集成化管理，支持分布式定时任务
9. 缓存集成化封装，缓存加载与具体业务代码分离
10. 支持分布式锁
11. 无侵入的物理分页及前后端解决方案
12. 弹窗再分页处理、查询按钮与分页定位
13. 页面使用JSP+CSS+JQuery，无第三方插件，入手难度小
14. 成熟、稳定、易维护

leader- slaves模式图如下：



## 3、matrix管理平台整体设计

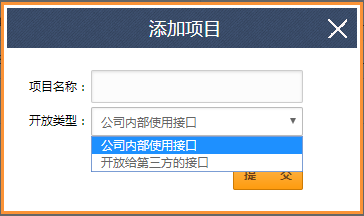
约定优于配置|业务分离|操作缓存|分布式|傻瓜式开发

### 3.1 api平台化管理

matrix系统完成了对接口的平台化管理。其中包括如下四项：api所属项目、跨域白名单、api信息树和请求者信息。如下图所示：



#### 3.1.1 api所属项目



该模块定义了api应该被分配到哪个项目下使用。比如open-api、安卓客户端、IOS客户端、IPad客户端、微信客户端、PC网页等等，将每一个api有序的区分开；当公司发展很快的时候，可能同时展开多个孵化项目，有上百人的开发队伍，每个小团队开发一个app，如果不进行有序化管理，接口开发人员很难快速、准确的定位到自己项目组的api，如果在加上新人接手离职人员的工作，更容易改错其他项目组的代码；同时也会加大运维人员的维护成本。

api所属项目模块主要配合【api信息树】模块来使用

#### 3.1.2 api信息树



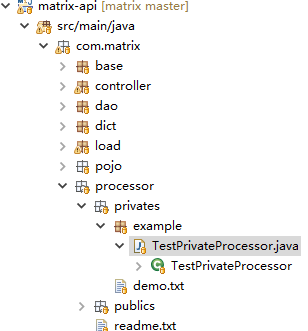
api信息树模块会将平台中所有注册的api以树形结构来进行集中展示。清晰的展示出每个api所从属的项目。root作为根节点，一级节点(open-api等)则为【api所属项目】中定义的内容。其下面每个具体的api记录当出发单击事件的时候，会作出如下展示：



其中【业务处理实现】public.order.OpenApiOrderInfo为接口开发这所建立的java类路径的一部分。其java类完整路径应为：

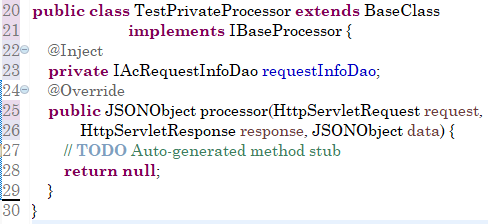
*com.matrix.processor*. public.order.OpenApiOrderInfo

系统在调用接口的时候会默认加上前缀com.matrix.processor，然后在调用这个类。如下图所示，为一个测试类：



系统规约：所有的api接口业务处理类都要定义在*com.matrix.processor*包下。这么做的目的是系统架构在设计之初利用了java中的路径同源性，也就是说，在其他的子项目中，只要是按照包路径的约定来命名，并且同时被一个web项目引入，都可以巧妙的通过java反射来调用。这也是matrix平台极为灵活的一个特性，将项目间耦合度通过反射和同源性完全解开，做到只关心数据结构。

具体到每一个对应api的处理类，其示例结构如下图所示：



针对这个类，说明如下：

1. BaseClass 强有力的核心类

com.matrix.base.BaseClass.java是matrix-core项目中的一个核心类，该类是对SpringMvc框架和J2EE Servlet2.0规范的深度封装体。他提供了如下功能：

1. 动态资源注入；
2. 获取项目关键配置信息缓存值 public String getConfig(String key)
3. 获取项目国际化配置文件信息缓存值public String getInfo(long infoCode, Object... parms)

其中，动态资源注入功能为整个项目提供了灵活的Spring Bean的获取功能，通过**@Inject**标签可以注入系统中任意的资源，比如：Service、Dao等等。更加出色的表现是当遇到Rpc接口数据转换的时候，可以将Rpc接口的侵入性降到最低。

getConfig(String key)，此方法能够获取指定目录下的配置文件信息，使用方式如下：



在resources/META-INF/matrix/config/config.matrix-file.properties文件中配置如下：



getInfo()方法则用于项目国际化，不做多解释。

1. IBaseProcessor

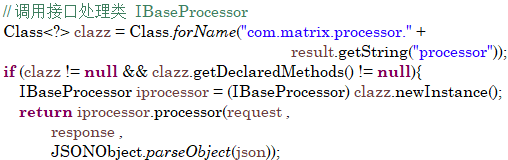
开放接口顶层抽象接口|所有开放给第三方的接口以及公司内部的api接口全部需要实现这个类的方法。他只有一个接口方法：

public JSONObject processor(HttpServletRequest request,

HttpServletResponse response , JSONObject data);

每个业务对应的实现类均在此处进行详细逻辑处理，data是完整的JSONObject消息请求对象，包含所有信息。

1. 接口调用方式

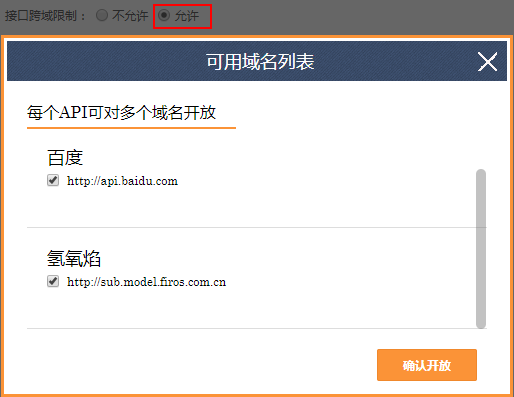


缓存、反射与面向接口变成相结合，完成对接口的调用。

#### 3.1.3 跨域白名单

展示平台接口允许的跨域信息，增强安全性，去除Jsonp对系统api的侵入性，让开发人员更专注于接口开发，省去无用代码。接口的跨域特性源于浏览器沙箱的一种保护措施，这种安全措施不允许浏览器上通过JavaScript脚本的方式调用异域接口(*但安卓和IOS客户端可以通过自己HttpClient客户端来调用开放出的接口*)。但往往在特定业务场景下，会用到不少跨域调用的场景，传统的方式是通过Jsonp侵入到api代码中来实现一个指定域名的调用，这种方式会冗余与服务无关的代码，同时只能指定一个特定域名；如果域名更换或者又有其他域名也要使用该接口，则需要停止服务器部署的项目，修改代码再从新发布项目，并且增加相同功能的api接口代码，代码重复性高，灵活性差。

matrix完美的解决了这个问题。在系统中随时定义允许跨域的域名列表，然后在【api信息树】中为一个指定的API接口开放多个可跨域域名。如下图点击【允许】单选框为一个API勾选多个可跨域的信任列表。



确认开放后，跨域信息会保存到接口对应的缓存中，在生产线上达到由平台控制，随时开启随时关闭的能力。

#### 3.1.4 接口熔断

与跨域白名单所提供的灵活性相似，接口熔断这个功能则进一步提升了系统安全性。它不同于ip白名单列表，指定一些ip禁止访问服务器；接口熔断的目的在于阻断所有尝试访问该接口的人。在特定业务场景下，我方接口的开发人员因失误，出现了危及公司财务、荣誉甚至系统稳定的代码行为，往往我们没有时间去重新部署服务器以及在第一时间找到问题代码所在，更不可能因为一个接口出现问题就立刻关闭所有的接口服务，让产品无法使用。

此时接口熔断技术便开始施展威力。在API树形结构列表中找到对应的接口，选择立刻熔断。



这个接口就会立刻作废，不会再让危害更进一步扩大。开发人员找到错误后，修改完代码再到凌晨进行发布即可。

#### 3.1.5 请求者信息

此处维护了所有对API的请求者信息，如下图：



系统API采用非对称加密的方式进行接口调用。每一个请求者都会对应一个缓存信息，其缓存结构如下：

key: xd-ApiRequestKey-133C9C129D53

value:

{

"key": "133C9C129D53", // 接口调用方公钥

"value": "6DFA608D49324E47A5D69A13523BDFDA", // 接口调用方私钥

"flag": 1, // 接口调用方状态：启用1 禁用 0

"atype": "private",

// 接口调用方分类。private:开放给公司内部接口请求者|

// public:公开，即注册给第三方的接口请求者

"organization": "IOS乘客端",

"createUserId": 1,

"createTime": 1512119626795,

"updateUserId": 1,

"updateTime": 1512119627160,

"id": 1,

"list": [] // 如果atype=public，此处应有对其开放的接口访问标识列表

}

### 3.2 系统权限设计

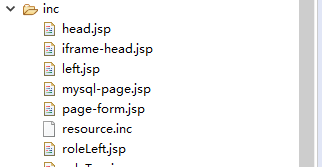
权限系统已经独立，在设计上采用leader- slaves模式，leader控制所有子项目比如下图中的【媒体站】。



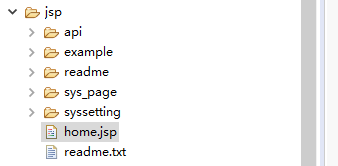
leader作为其他web系统的管理者，对其提供权限控制。该模块包含系统用户列表、系统角色列表和系统功能列表。系统角色关联着系统功能列表，系统用户则可以关联多个系统角色，系统权限所有的数据结构都已经缓存化，并在对应的Init类中有缓存结构标识的注释。

其对应的项目为matrix-manager。对于拦截器，则使用的是SpringMvc自带的拦截器：HandlerInterceptorAdapter，并在其基础上进行了封装，对我们自己的业务进行支持。

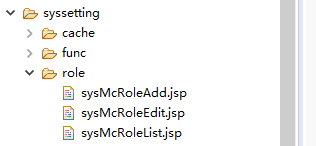
整体页面结构采用JSP分割的方式来渲染。在webapp/inc文件夹下是设计好的分割文件；



主要的展示区域则采用iframe这种主流页面嵌入式方式；在webapp/jsp/home.jsp中定义了嵌入iframe的地方。



syssetting 文件夹下是系统权限、角色相关设置页面。



### 3.3 文件服务器

文件服务器作为后台管理系统的一个通用组件，已经在matrix中独立成为一个子项目，通过maven的依赖引用可以为其他web项目提供文件服务支持，同时也可以独立部署为一个文件服务器，供全局调用。

matrix-file提供以下常规格式的文件服务：jpeg,jpg,png,bmp,docx,doc,xlsx,xls,pdf,

html,zip,rar,ppt,pptx；其他类型的文件会被服务器拦截。文件上传完成后，会返回如下消息体给接口调用者：

{

"status": "success", 状态：success or error

"msg": "文件上传完成",

"original": "example.png", 图片原标题

"title": " bf92f1576b23470a948dbdcb8feba788.png", 图片新标题

"size": "62091", 文件大小

"save": " image/29c10/bf92f1576b23470a948dbdcb8feba788.png ",

用于保存到数据库

"type": "image", 文件类型

"url": " <http://192.168.1.34:8080/matrix-file/image/29c10/>

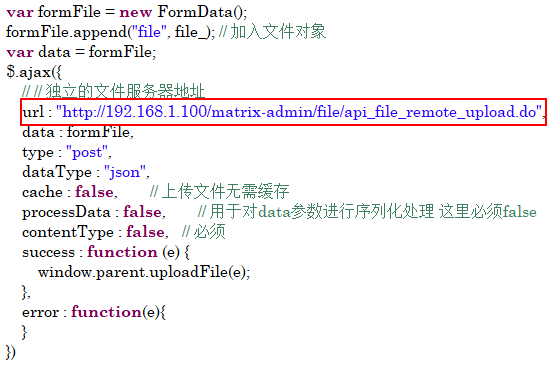
bf92f1576b23470a948dbdcb8feba788.png ", 可访问路径

"height": "247" 图片高

"width": "163", 图片宽

}

该接口多数情况下会在HTML中进行调用，ajax的请求代码如下：



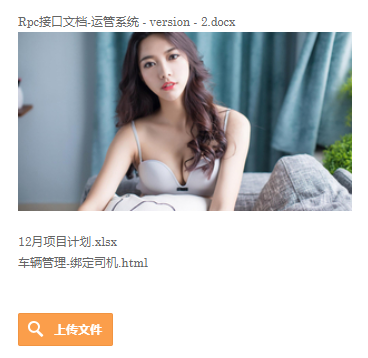
这是一个典型的跨域访问。他的调用方式是在需要的地方嵌入一个iframe标签来调用：



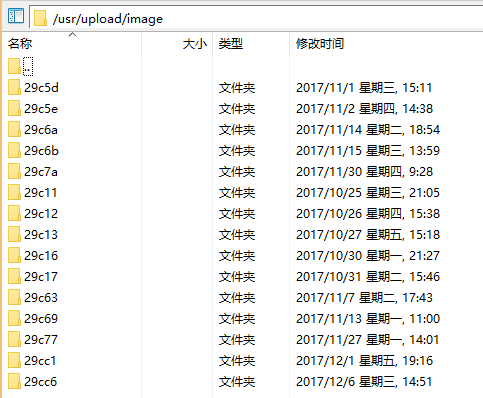
其配置后的效果图如下：

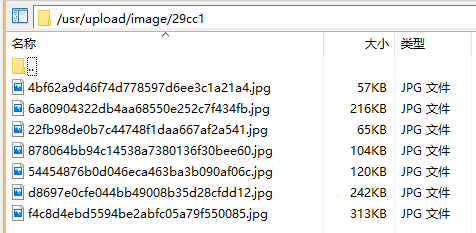


上传文件后的示例如下：



关于上传文件的具体使用方式，在开发者快速入门中有详细的代码展示。在linux服务器上的文件夹展示如下：



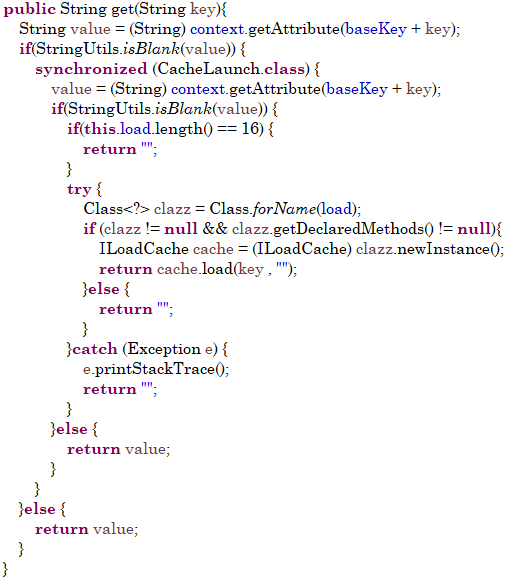


子文件夹的命名是以时间戳做Hex转码后的值。

### 3.4 缓存集成与封装

matrix平台最出色的一个地方就是对缓存的使用，整个系统都在操作缓存，只有在添加或者修改的时候会涉及到操作数据库。在高并发环境下，应对海量的互联网请求，缓存的响应速度在1ms上下，效率几乎是操作数据库的4、5百倍，因为数据库在高并发的请求下，性能低下是不争的事实。

得益于matrix-cache项目对缓存的高强度封装和系统在整体架构上的前瞻性设计，代码在对缓存上的操作极为方便和简单，同时做到了缓存初始化的代码(*这部分代码在查询数据库，组织缓存数据信息，与具体的服务无关*)与具体业务代码分离，将耦合性降到最低。同时，在取缓存时底层已做封装，可以有效防止 缓存击穿 [1 ]，其代码如下：

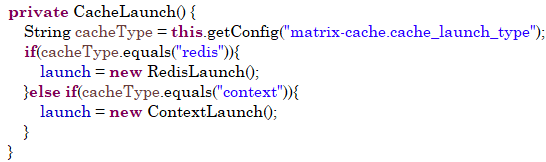


#### 3.4.1 缓存具体使用步骤

首先在类中加入如下代码：

private IBaseLaunch<ICacheFactory> launch = CacheLaunch.getInstance().Launch();

这段代码可以对开发者屏蔽缓存类型，不用开发者关心用的Redis还是Ehcache，或者将来出现的其他更优秀的缓存框架。假如将来出现了其他的缓存框架我们需要替换，从而获得更高的代码执行效率，我们业务中的代码可以在不修改的情况下，无损的切换到新的缓存中。目前matrix-cache支持redis 和 ServletContext两种类型的缓存。其部分代码如下：



在具体使用的地方，代码如下，设置缓存：

launch.loadDictCache(DCacheEnum.ApiDomain , null).set(key , value);

获取缓存：

String value =

launch.loadDictCache(DCacheEnum.ApiRequester , "InitApiRequester").get(key);

一行代码解决所有问题。在取缓存时“InitApiRequester”这句话是缓存封装的精华之一，这段代码做到了**缓存加载与具体业务代码分离**。InitApiRequester是一个java类，他存在于一个约定的路径下：com.matrix.load。当取缓存为空的时候，会到com.matrix.load包下加载InitApiRequester.java类来捞取缓存信息，从而将无关代码与业务代码分离。

Init为类命名的约定前缀，ApiRequester则是枚举中约定的类型。这些类会继承BaseClass.java，同时实现ILoadCache接口。

*[缓存击穿]:*

*这里指查询一个缓存数据，由于缓存是不命中时出于容错考虑，从存储层查不到数据则返回空，这将导致这个查询的数据每次请求都要到存储层去查询，在流量大时，数据库服务器可能就会挂掉，从而导致整个系统服务崩溃。*

### 3.5 开发者快速入门

matrix系统最为人性化的一点。在互联网公司，由于各种原因人员变动很频繁，但是产品需求永远在追加。这就需要系统的开发成本极低才能让新加入的人员快速入手项目。matrix提供了傻瓜式的开发样例，在开发示例中有极为丰富的脚手架代码，从JSP页面层一直到mybatis的xml配置文件，我们倡导一个原则：**复制即可得**。在业务层的开发中，一段一段的敲代码效率是最低的。

