无线端面相数据设计与可视化编程语言DSON

meili-inc

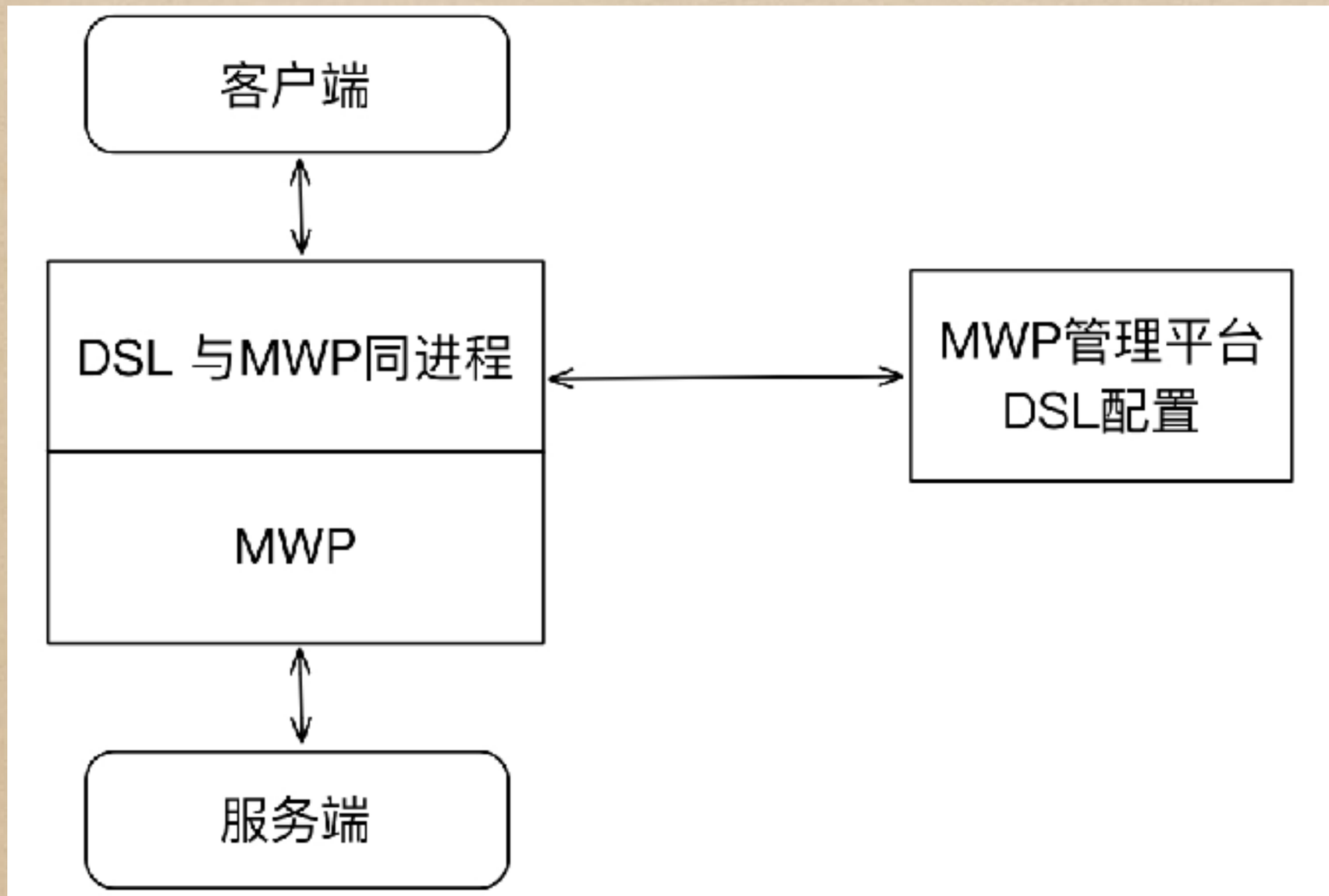
吴邪（刘昱杰）

目录

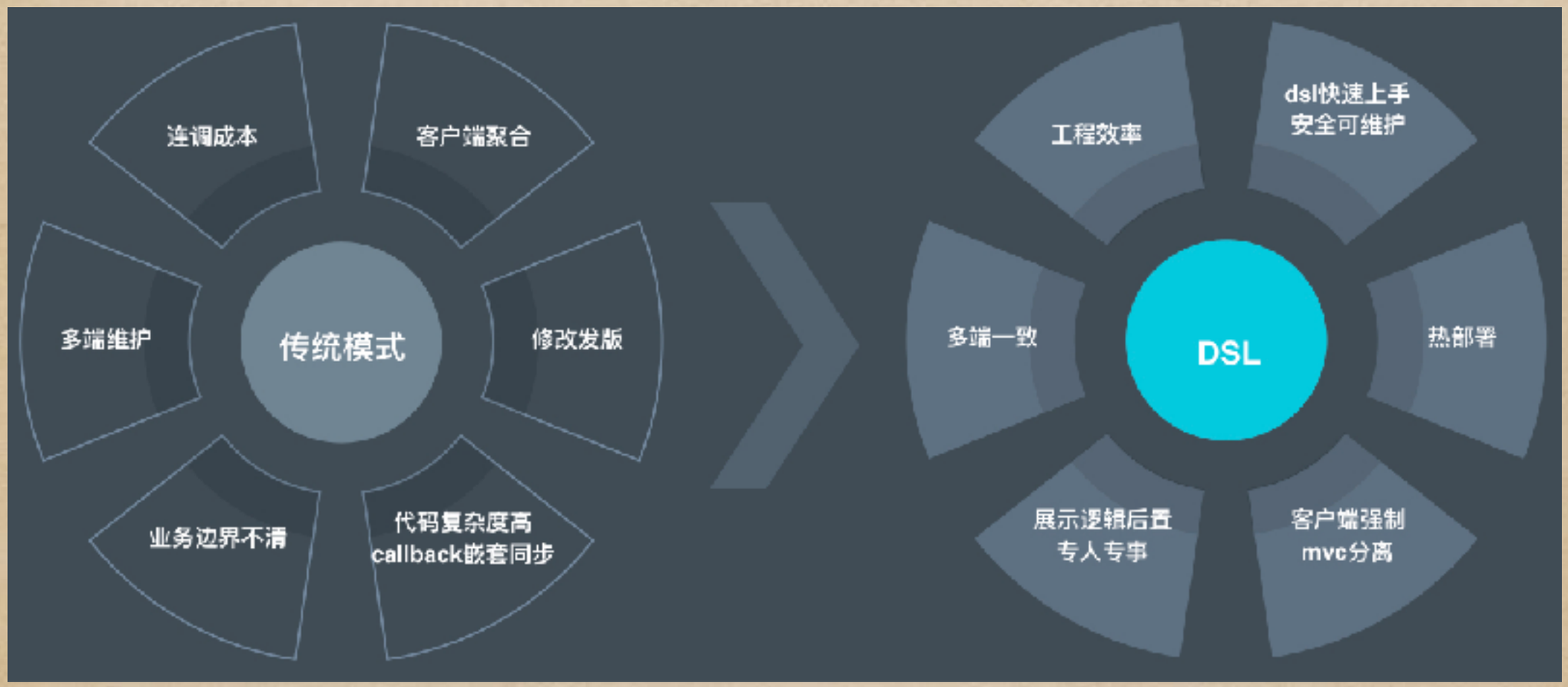
MWP－DSL是啥玩意儿?

- MWP是啥?
- DSL层本质，无线端面向数据设计与开发，针对业务数据的无线端前后端分离的方案。
- 嵌入在MWP中，提供一套DSL（DSON），针对无线端（Android，IOS，H5）和展现层相关的业务数据（多个数据源）的组装、拼接和转换

Dsl是啥玩意儿?



传统模式问题与DSL目标



业界相关工作

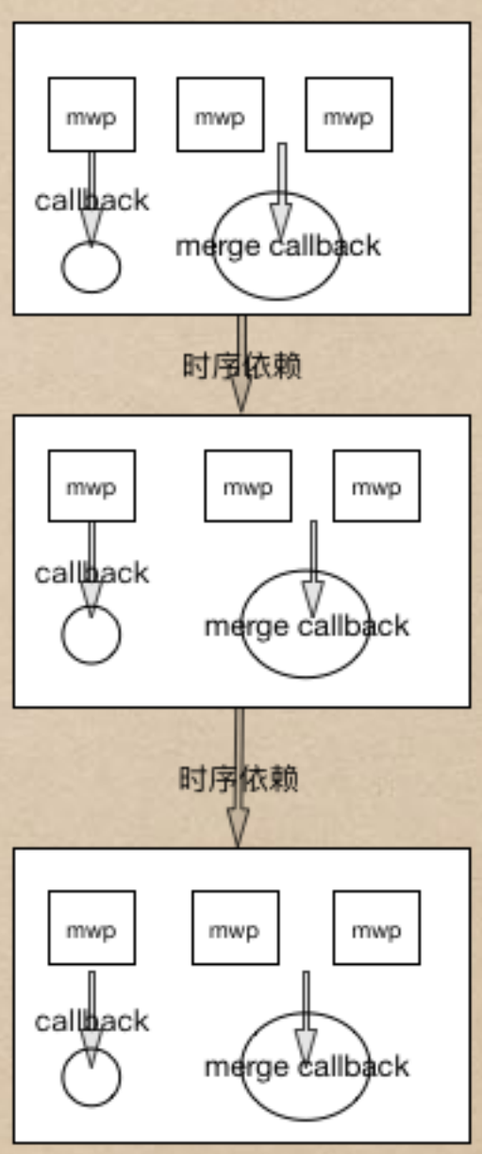
- fackbook GraphQL 一种新的查询数据的方式，重点是在客户端如何查，类似于sql（hive），表达能力有限基本是对接口的filter,而我们的业务需要的是map reduce（有复杂逻辑和循环），同时需要对服务端做大量适配和改造。
- ali midway nodejs 作为中间层，nodejs的学习运维成本，不是真正面向数据。

DSL的挑战-业务

- 真实的业务场景较为复杂，会出现任意N个MWP接口随机组合和callback情况(通用的业务模型)
- DSL层具体提供的能力，既受限又够用（静态代码编译，白名单、黑名单保证安全，DSON)
- DSL层易用性（数据可视化语言DSON，所见即所得，从客户端同学需要的具体json着手，不需要重新学习一门语言，不需要管并发，内存管理，锁等等)

方案-DSL业务模型

- N个接口任意情况组合
- M个flush到客户端（M>1 bigpipe后续支持）
- T个独立callback(包含错误，完全业务方定制)
- 三种基本原子组合（独立、merge,时序依赖）



挑战~性能、稳定性和安全

- 轻量级MWP转发->多MWP组合并运算带来的系统压力
- MWP特点（高稳定性、高QPS、低rt，任何一点小问题会被放大）（经过双十一的验证）
- 开放给业务方编辑带来的安全和性能隐患

方案－性能

1.全异步化

- rxjava

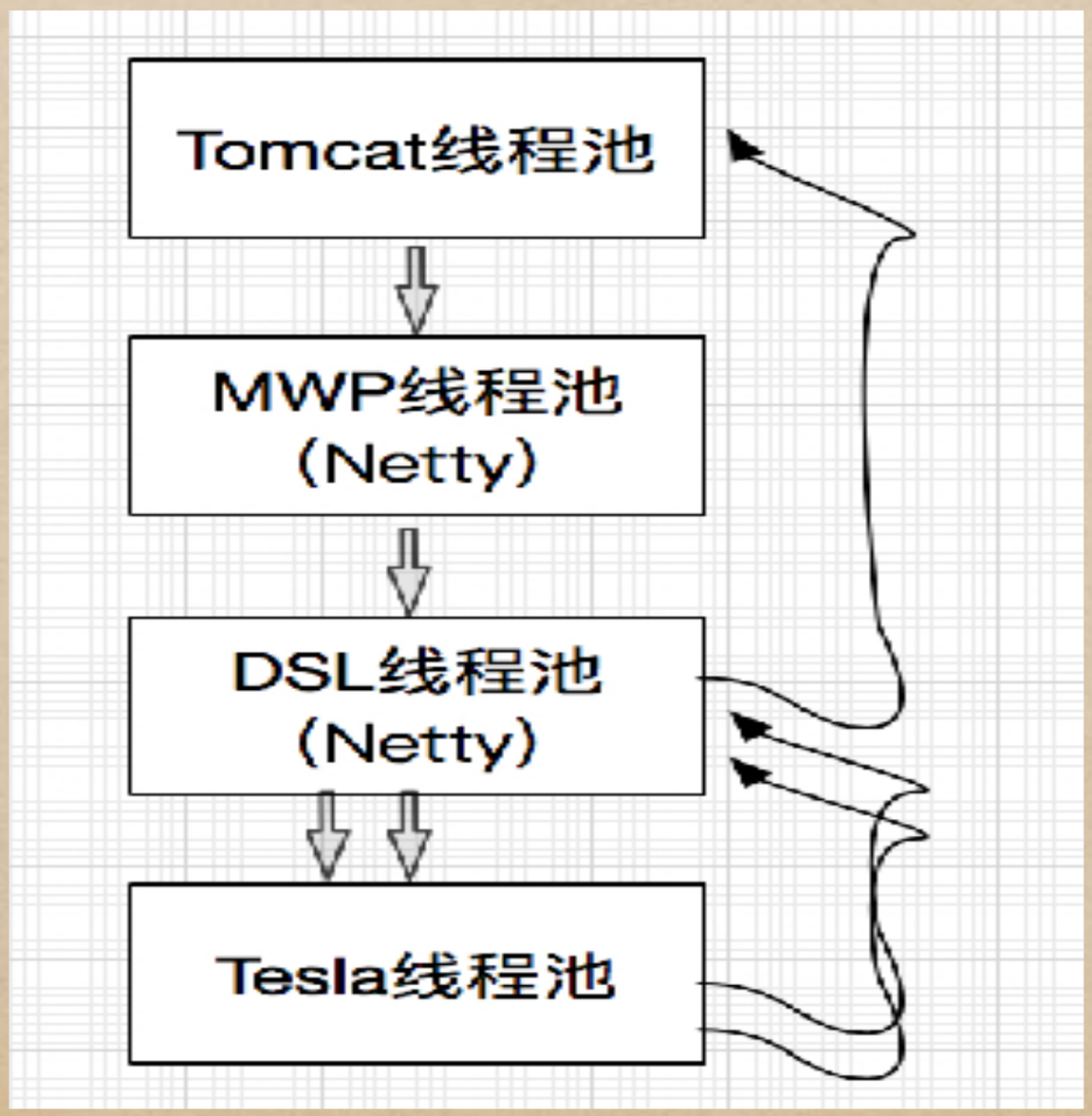
2.线程模型与调度

- netty event loop
- 线程隔离
- 多callback仍然交给触发线程，避免并发和线程拷贝

3.高性能groovy集成

- 执行效率（静态编译与原生java同样效率）
- jvm调优（GC、perm）
- dson中间代码优化

DSL线程模型



高性能groovy集成

- 采用groovy作为目标语言的原因

1. 对JAVA集成友好
2. 部署运维方便（运行在JVM上不需要其他特殊环境）
3. 经过一系列的优化性能上和原生JAVA接近，同时经过优化可以解决GC相关问题
4. 学习接入成本低，相关语法糖适合dsl

高性能groovy集成

1. 三种集成方法比较与优化

	GroovyScriptEngine	GroovyClassLoader	Java原生
无优化	30983ms	29363ms	608ms
InvokeDynamic	\	11267ms	\
CompileStatic	4020ms	1009ms	\

2. 两种优化方案获得和原生java同样的执行效率（50倍，CompileStatic，InvokeDynamic指令）

3. GroovyClassLoader隔离（避免class unloading失败）

4. jvm调优（调大CodeCache参数，开启classunloading -XX:+UseConcMarkSweepGC -XX:+CMSClassUnloadingEnabled -Dgroovy.use.classvalue=true）对于jdk7+,groovy2.4以上版本 Dgroovy.use.classvalue=true，打开classvalue优化

方案－稳定性

- 线程隔离
- DSL接口隔离与容错
- 系统资源可控（cpu和内存可控，对于循环封装只能对Collection的iterator，避免死循环造成CPU的压力；限制用户自己new出对象，数据操作完全基于http接口返回的数据，防止误用导致的内存问题；内存的压力和之前的MWP比主要来源于timeout，3s超时，接口rt监控与接口级别控制）
- DSL层开关
- 灰度发布与灰度切流量

方案－安全

- 能力受限（通过DSON SDK,静态代码扫描，白名单和黑名单机制，明确确定用户能做什么，不能做什么）
- DSL接口权限管理（可控，溯源）

数据可视化语言DSON

- 基于json的数据可视化自定义语言DSON（包含逻辑判断和基于返回list的循环等一系列特性）

为什么要有DSON

- 降低接入MWP DSL门槛，用户用最小的成本（学习、使用、调试、维护）将自己的业务需求（对N个MWP接口的组装、拼接和转换）变成具体可以运行的代码
- 所见即所得，从客户端同学的需求和思维习惯出发，数据驱动设计，我需要的就是一个最终的可以被客户端使用的json格式的数据，所以就从这个json出发，直接去思考如何通过Mwp接口返回的数据构造我需要的数据。
- 受限表达，安全性（能力受限，性能可控）
- 写代码方式的门槛：

1. 熟悉代码的具体语法成本较高
2. 需要知道业务逻辑如何与DSL框架发生联系
3. 1和2带来的学习成本、编码、调试和维护问题，比如代码拼写错误等

DSON设计

```
{
 "#set($ml)": "$payloadMap['mwp.application.api@1']",
 "flushMap": {
   "flushkey": {
     "ret": "$ml['ret']",
     "data": {
       "MRArray": [{
         "#map($ml['data']['feature'],Map)": {
           "id": "$node['id']",
           "counter": {
             "#set($templist1)": "[1,2,3,4]",
             "#reduce($templist1, int, 0)": {
               "#set($my)": "_DInteger.valueOf($init)",
               "#if($init+1)": "$my+_DInteger.valueOf($node)",
               "#else": "$my"
             }
           }
         }
       }]
     }
   }
 },
 "parameterMap" : {},
 "header" : {}
}
```

```
{
  "flushMap" : {
    "flushkey" : {
      "ret" : "SUCCESS",
      "data" : {
        "MRArray" : [ {
          "id" : 1,
          "counter" : 6
        },
        {
          "id" : 2,
          "counter" : 6
        },{
          "id" : 3,
          "counter" : 6
        },{
          "id" : 4,
          "counter" : 6
        }]
      }
    }
  },
  "parameterMap" : {},
  "header" : {}
}
```

DSON编译器设计

- 基于groovy的内部DSL（叶子节点内部表达式）与外部DSL结合（总体DSON结构构建抽象语法树），最终目的是将DSON翻译成可执行的groovy代码。
- 通过抽象语法树，将DSON自身语法和目标语言生成实现分离，可以各自独立做演进，一方面方便DSON语法扩展，另一方面，方便翻译成groovy之外的其他目标语言

1. 词法分析（DSON格式解析、系统关键词识别）
2. 语法分析（静态DSON语法规则校验）
3. 语义分析与中间代码生成（抽象语法树构建）
4. 中间代码优化（基于抽象语法树的代码优化提升执行效率，主要是通过编译时类型确定，中间对象生成来优化执行效率）
5. 目标代码生成（跟进特点规则生成目标groovy代码）

DSON扩展性

- 提供udf功能，一方面解耦用户复杂需求，另一方面方便用户自身代码和配置重用与沉淀。

DSL周边体系

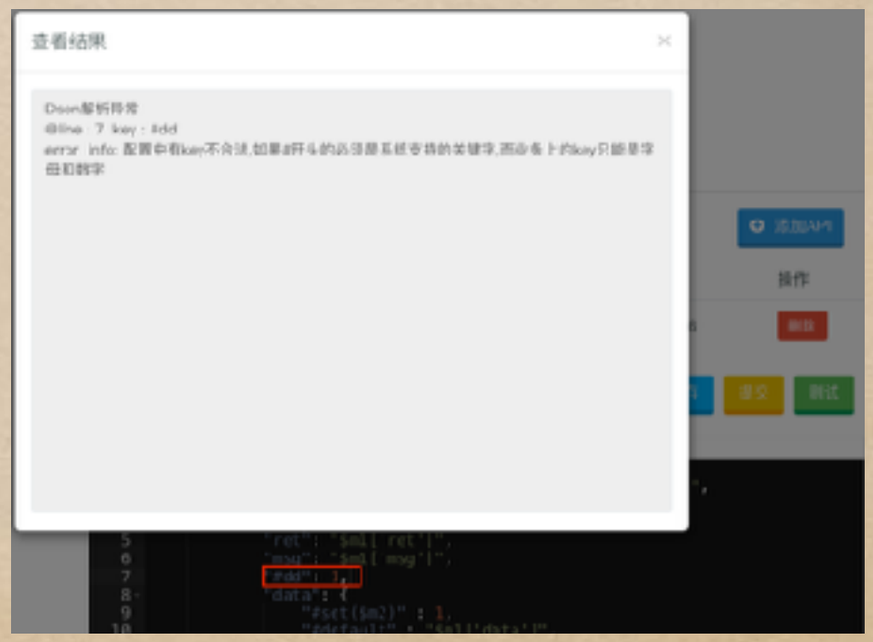
1. dsl管理后台
2. web ide(dson在线编辑器)

- 自动生成脚手架配置
- 代码自动提示与补全（$,#,_,:,工具类）

3. debug与错误提示

- dson静态语法校验与错误提示（精确到某行某Key）
- 运行时runtime异常提示（精确到某行）
- 编译时异常（精确到某行）
- debug

4. 两种测试方法（mock数据单侧DSL，连MWP整个接口测试）
5. 权限管理（编辑权限，DSL接口锁定，DSL编辑记录查询）
6. 线上运维工具（管理后台业务方直接获取错误日志）



DSL周边体系

7. 监控（对接sentry，接口级别qps rt相关）

8. 报警 （借助sentry，接口级别自定义报警）

9. 自动化测试daily run（业务方同学配置接口级别多个自动化 test case，daily run检测接口业务变更）

DSL适合什么样的业务场景

- 有多个接口任意组合操作的操作需求
- 接口返回数据有操作，并且后续有扩展需求
- 多终端适配
- 前台业务需要合并后端多个业务方的数据源，避免后端投入和联调成本
- DSL不适合什么场景（缓存，CDN，比如麦田；操作过于复杂的业务要下层到service层，比如需要处理并发；多个MWP接口各自独立，且对返回后的数据无需处理，有自己的降级策略和强弱依赖关系）
- 衡量是否适合接入，用户体验、性能、开发效率

DSL后续

- bigpipe
- 容器化部署与微服务

Thank you