

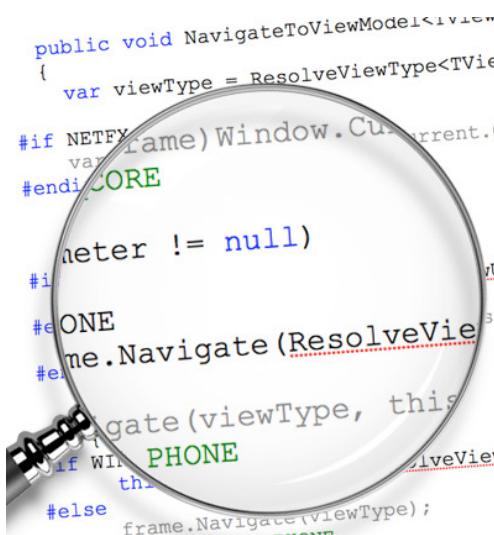
大代码时代的新基建

Query-based 代码数据分析平台实践

目录

- 1. 大规模代码分析的挑战**
- 2. 代码数据分析平台介绍**
 - 代码数据化、标准化 - COREF
 - 代码数据分析语言 - Godel
 - 平台化、产品化 - Sparrow
- 3. 目前落地情况**
- 4. 未来展望**

静态代码分析是指在不实际执行程序的情况下，对代码语义和行为进行分析。



```
public void NavigateToViewModel()
{
    var viewType = ResolveViewType<TViewModel>();
    #if NETFX
    var frame = Window.Current.Content;
    #endif CORE

    if (meter != null)
    {
        #if ONE
        meter.Navigate(ResolveView);
        #else
        meter.Navigate(viewType, this);
        #endif
        if (Windows PHONE)
            frame.NavigateUri(Uri(viewType));
        else
            frame.Navigate(viewType);
    }
    #endif // WINDOWS PHONE
}
```



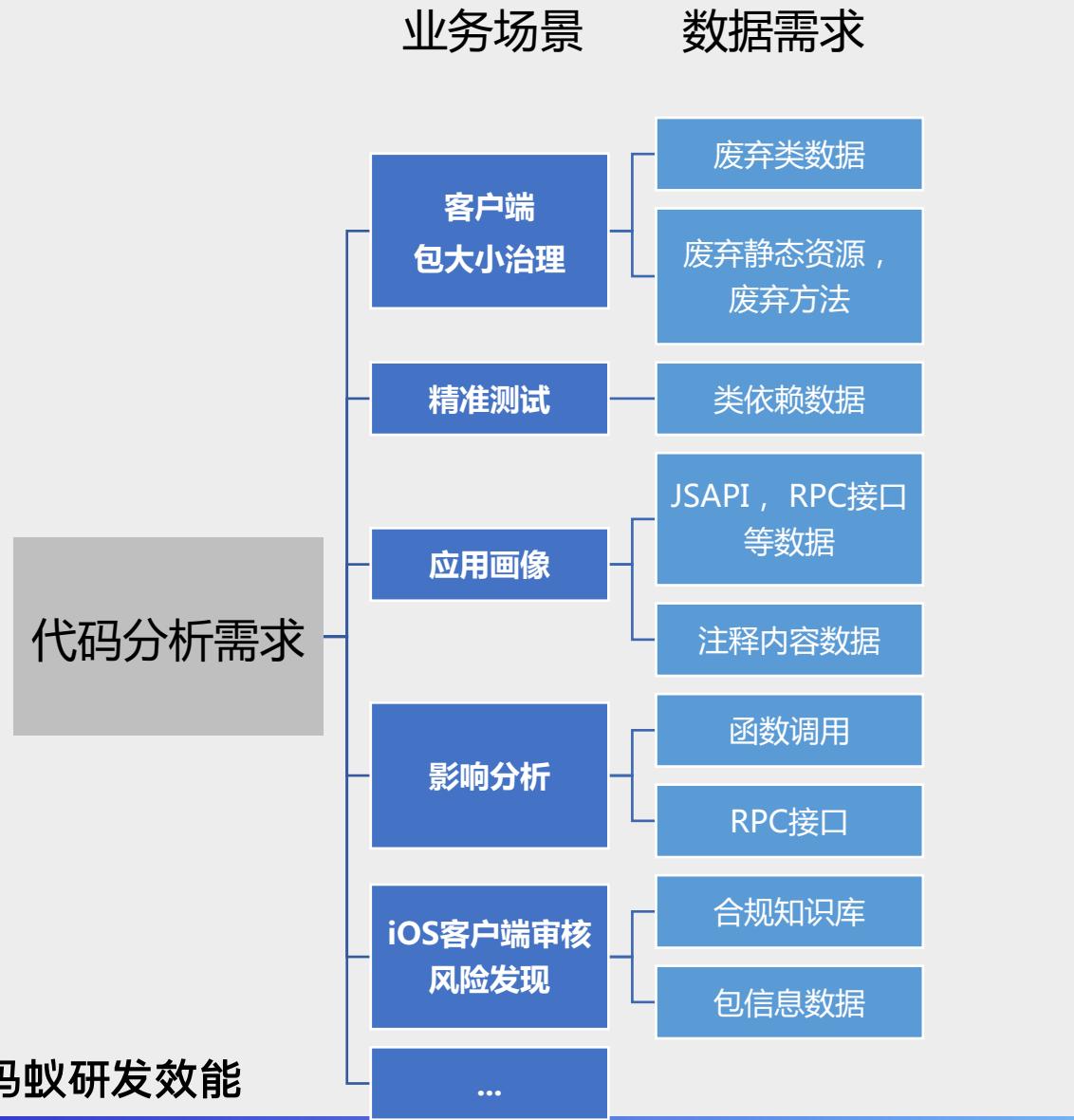
安全生产

主动防御：检查编码规范违反、代码缺陷、安全漏洞、恶意或违规行为

被动防御：编写专属编码规则，防范事故二次发生

研发辅助

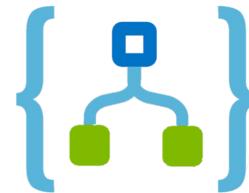
语言服务、代码度量和重构、架构理解、性能优化



需求碎片化



需求定制化



轻语义分析



需要大规模分析 多种语言代码



代码分析的技术挑战

代码分析 = 语言种类 × 算法种类 × 场景配置 × 规则种类

典型数量

10

8

20

5000

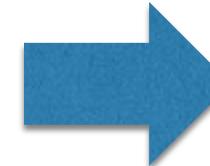


大量的碎片化的
分析需求
+
巨大的计算量

VS.

有限的研发资源

<10 研发投入
数百台机器



- 维护多套分析引擎
- 依赖几十个不同工具
- 标准化只做到流程级

技术难复用

给OBJC设计的分析引擎
无法用来分析JS项目

技术难迁移

做C++ Parser的编译器工
程师，不懂如何写规则

可维护性差

不同组件的能力，性能，可
靠性，参差不齐

重复开发

会写Java分析算法的同学，
Python算法又得写一套

难以开放

数以千计的分析方法和规
则，工作强依赖专家经验，
难以复制

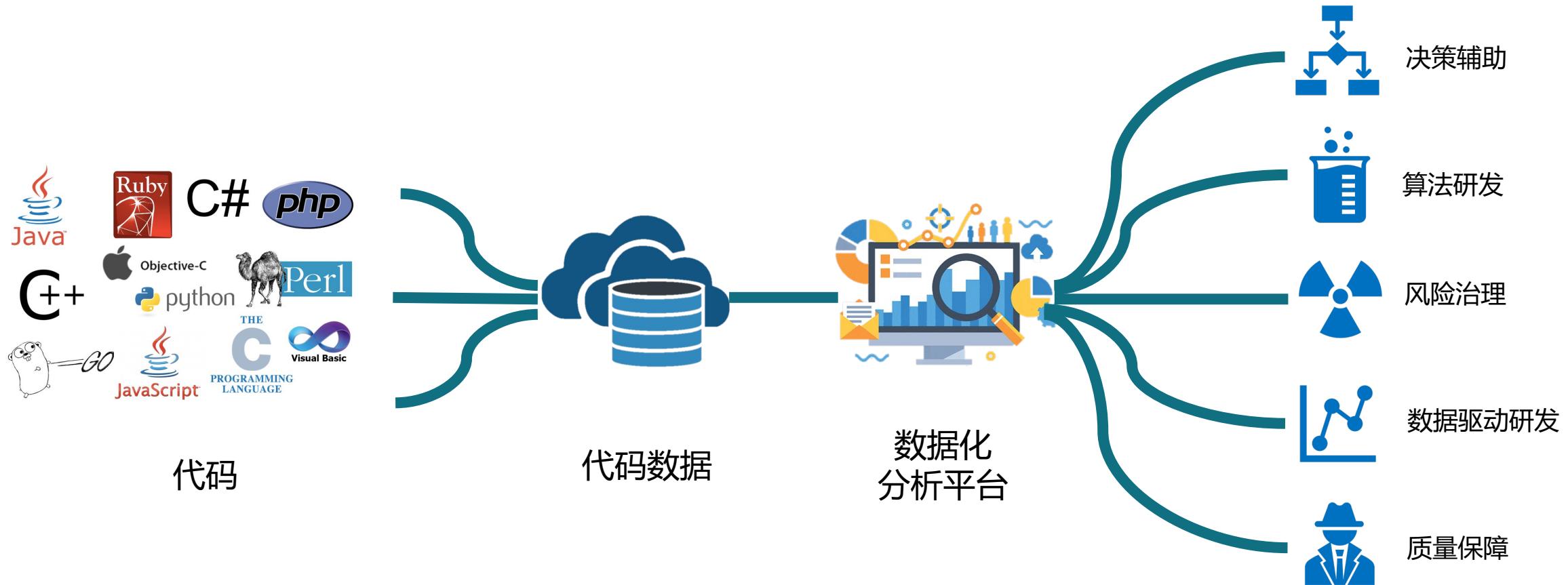
不够通用

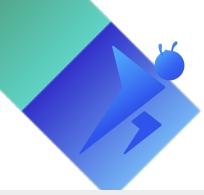
为代码度量设计的工具
无法用于抽取代码数据

用做数据库的思路去做代码分析平台

- 理念： 简单、一致、标准、开放
- 方法： 数据化，标准化、平台化

BigCode时代的新基建 -- 代码数据化分析平台





Sparrow —— QL-Based 代码数据分析



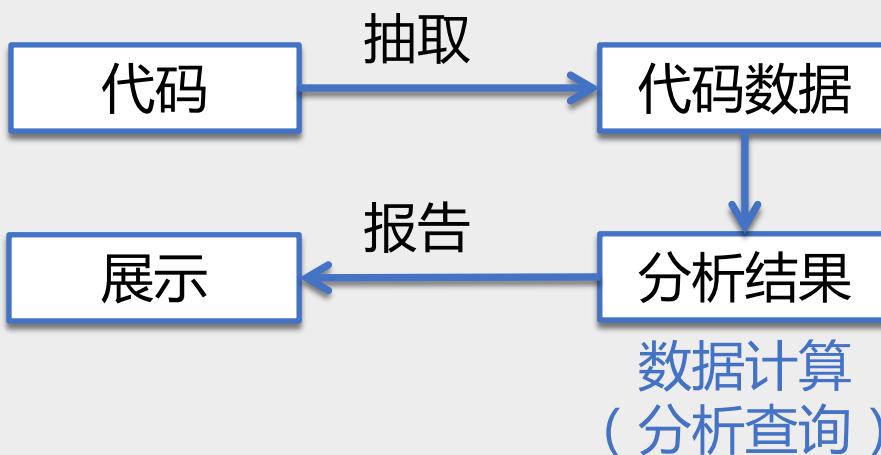
愿景

- 人人都是代码分析师

使用方式

- 使用声明式的DSL – Gödel语言，来描述你想要什么。剩下的交给Query引擎和Sparrow系统

Sparrow 代码分析流程



样例：DRMAttribute查询

DRMAttribute查询

查询目的

查询DRM的attribute信息，不同版本会有不同，提取字段（3.0的drm）或者方法名（1.0的drm）
示例代码如下：

```
1 @DResource(id = "com.alipay.govern.greeting")
2 public class GreetingConfig {
3
4     /**
5      * 定义一个 drm 属性，drm 属性 必须有对应的 setter、getter 方法，建议用 idea 自动生成的方法
6      * 另外 drm 属性不能是静态属性，也就是不能是 static 属性，否则失效
7     */
8 }
```

Query

3.0版本查询

```
1 import coref.java.*
2
3 //输出带有DAttribute注解的字段的名字
4 predicate output(string name){
5     exists(Field f){
6         f.getAnnotation().getName() == "DAttribute"
7         and
8         name == f.getName()
9     }
10 }
```

Results

查询结果

0
com.alipay.zoneclient.internal.DefaultRouteCoordinator.zoneInfo:String
com.alipay.zoneclient.internal.DefaultRouteCoordinator.elasticRuleVersion:String
com.alipay.zoneclient.internal.DefaultRouteCoordinator.zoneColor:String
com.alipay.zoneclient.internal.DefaultRouteCoordinator.rpcRouteByGroupWeight:String
com.alipay.zoneclient.internal.DefaultRouteCoordinator.grayKey:String
com.alipay.zoneclient.internal.DefaultRouteCoordinator.idcDomainSwitch:String
com.alipay.zoneclient.internal.DefaultRouteCoordinator.idcDomainScope:String

举几个例子

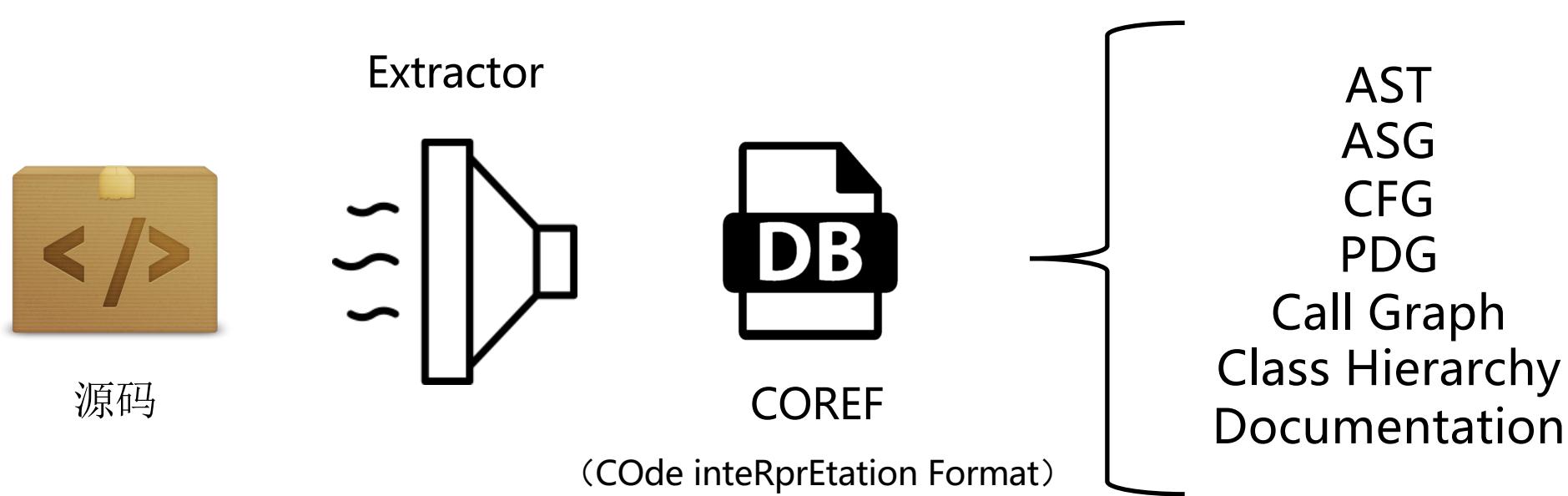


怎么建设？

第一步



代码数据化，标准化



COREF设计目标

- **Goals of COREF**

- 直接支持的代码索引/跳转/搜索。
- 轻量级的程序分析。
- 转换生成更高阶的格式(如LLVM IR)。

- **Non-goals of COREF**

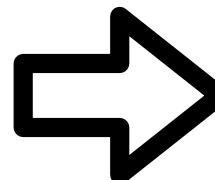
- 不是一个language-agnostic的数据格式，定义的时候会考虑不同语言带来的影响。
- COREF不是一个具体的平台/应用/实现，它是一个**格式定义**。
- 编译构建 (aka. COREF不是UNCOL)。
- 不直接支持**Heavy-Weight程序分析** (e.g., path-sensitive analysis)

Abstract Syntax Tree (AST)

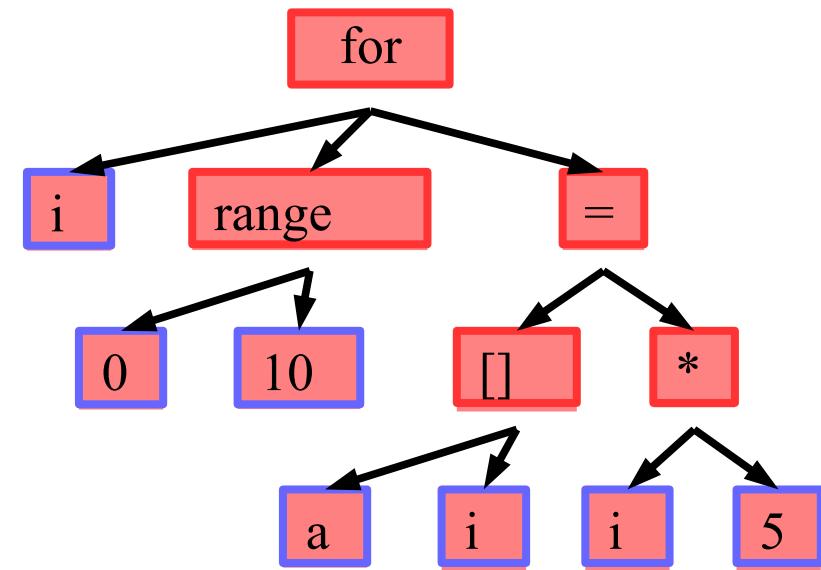
- Tokens
 - 源代码根据词法规则转化为一组tokens
- 抽象语法树(AST)
 - 满足语法规则的Tokens组合结果
 - 上下文无关文法
 - 剔除了非必要的节点，如comment，parenthesis，空格¹

Source Code

```
for i in range(0, 10):  
    a[i] = i * 5
```



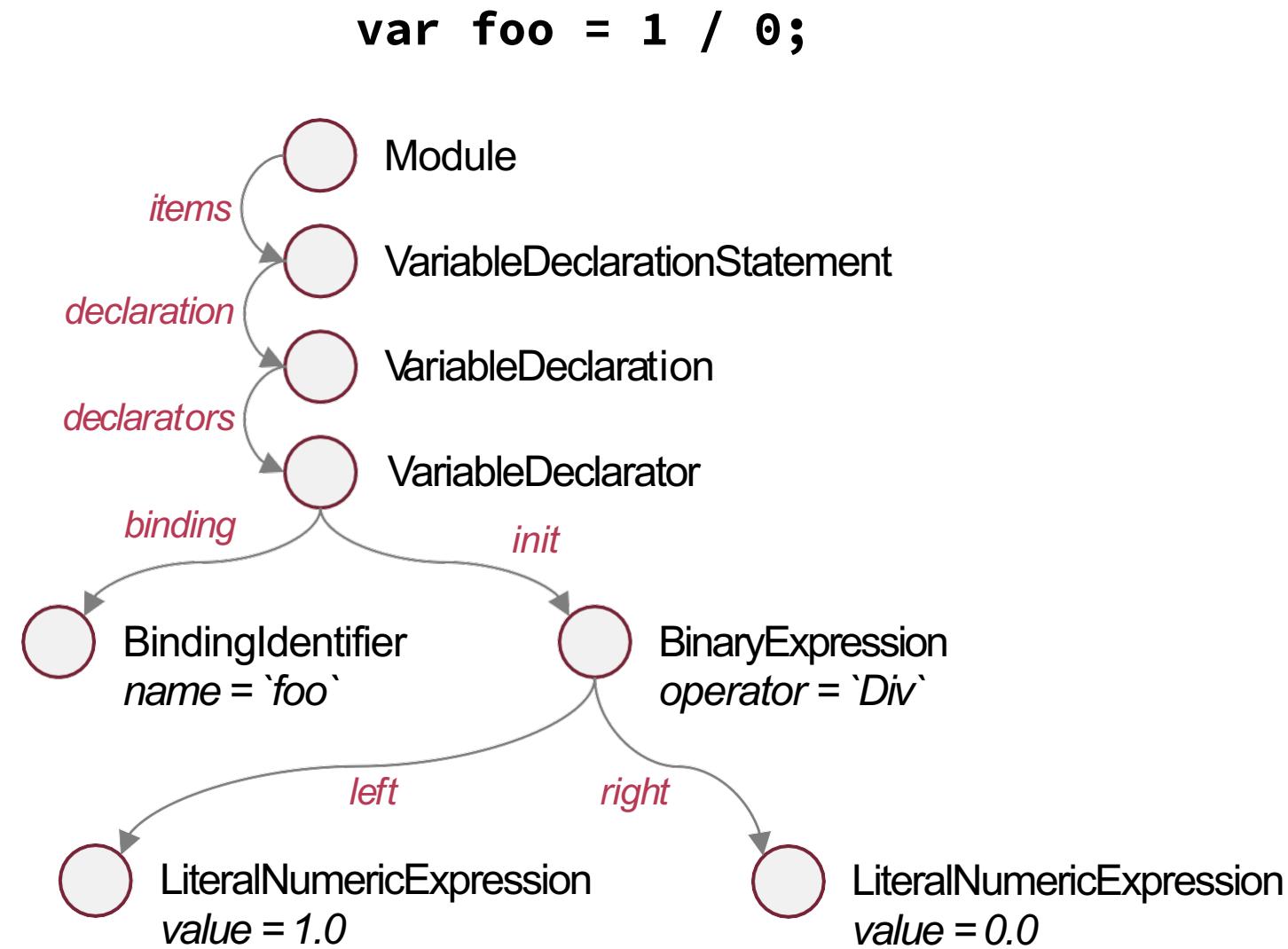
Abstract Syntax Tree (AST)



抽象语义图 Abstract Semantic Graph (ASG)



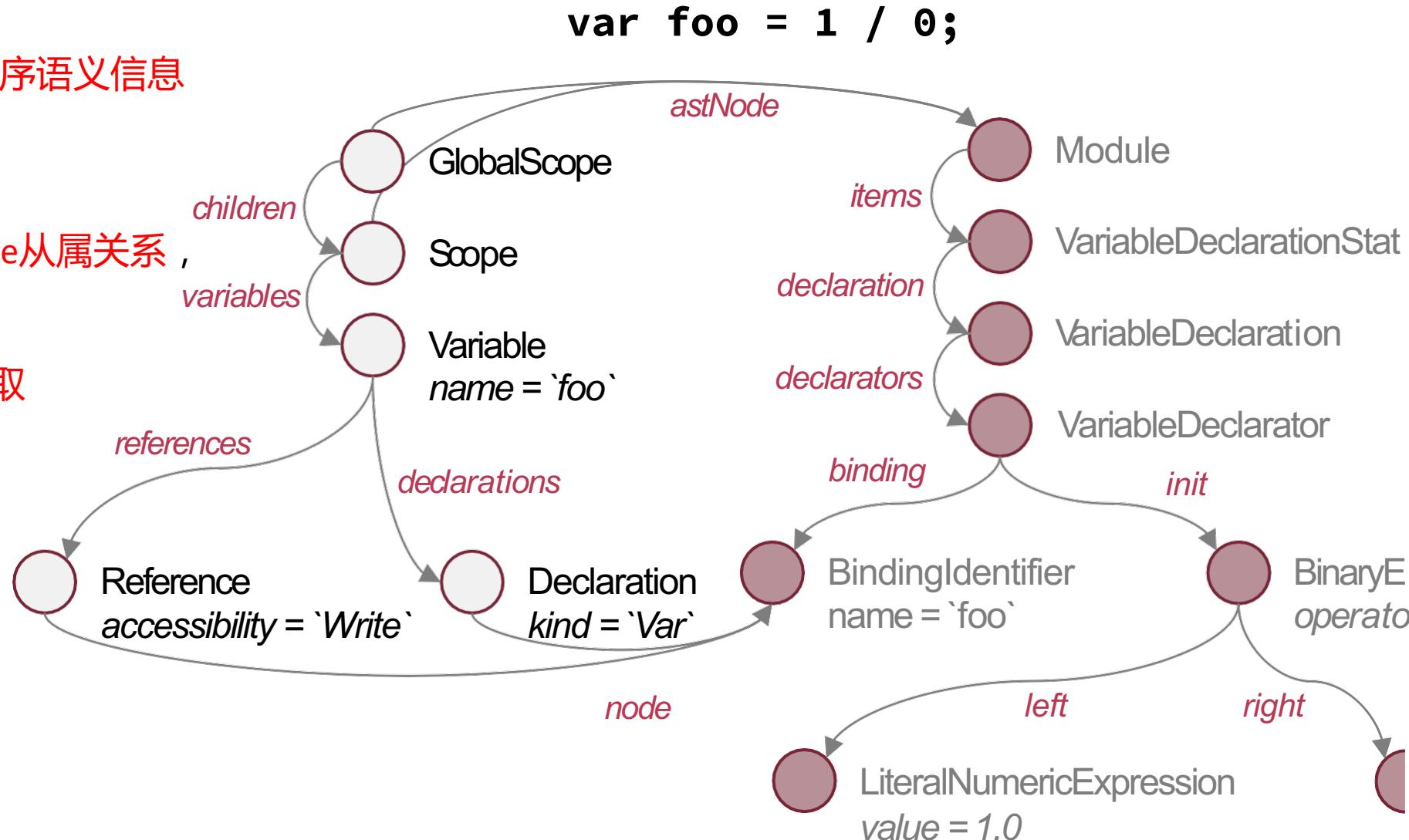
- 是高于AST的程序抽象，表示程序语义信息
- v.s AST 增加
 - 点：Scope
 - 边：引用/定义关系，Scope从属关系，Scope间关系
- 注：还包括公共子树的抽取



抽象语义图 Abstract Semantic Graph (ASG)



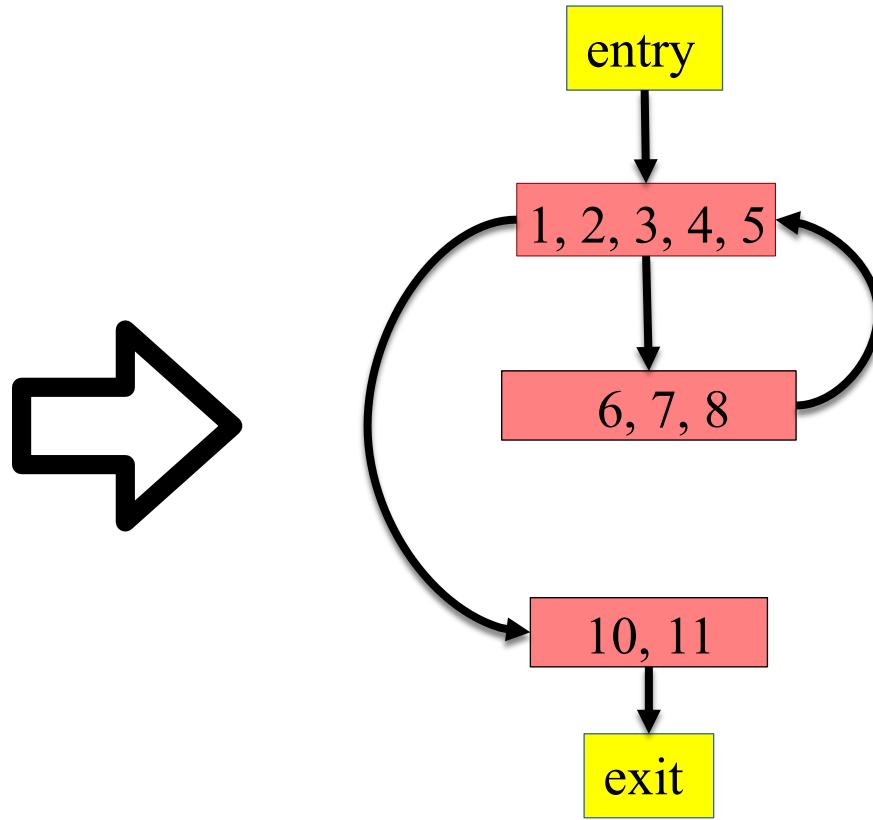
- 是高于AST的程序抽象，表示程序语义信息
- v.s AST 增加
 - 点：Scope
 - 边：引用/定义关系，Scope从属关系，Scope间关系
- 注：还包括公共子树的抽取



控制流图 Control Flow Graph (CFG)

- 表示程序语句间的可达关系。
- 表示程序的所有可能通路。
- v.s. AST 增加
 - 点：entry, exit, basicblock
 - 边：可达关系，控制谓词

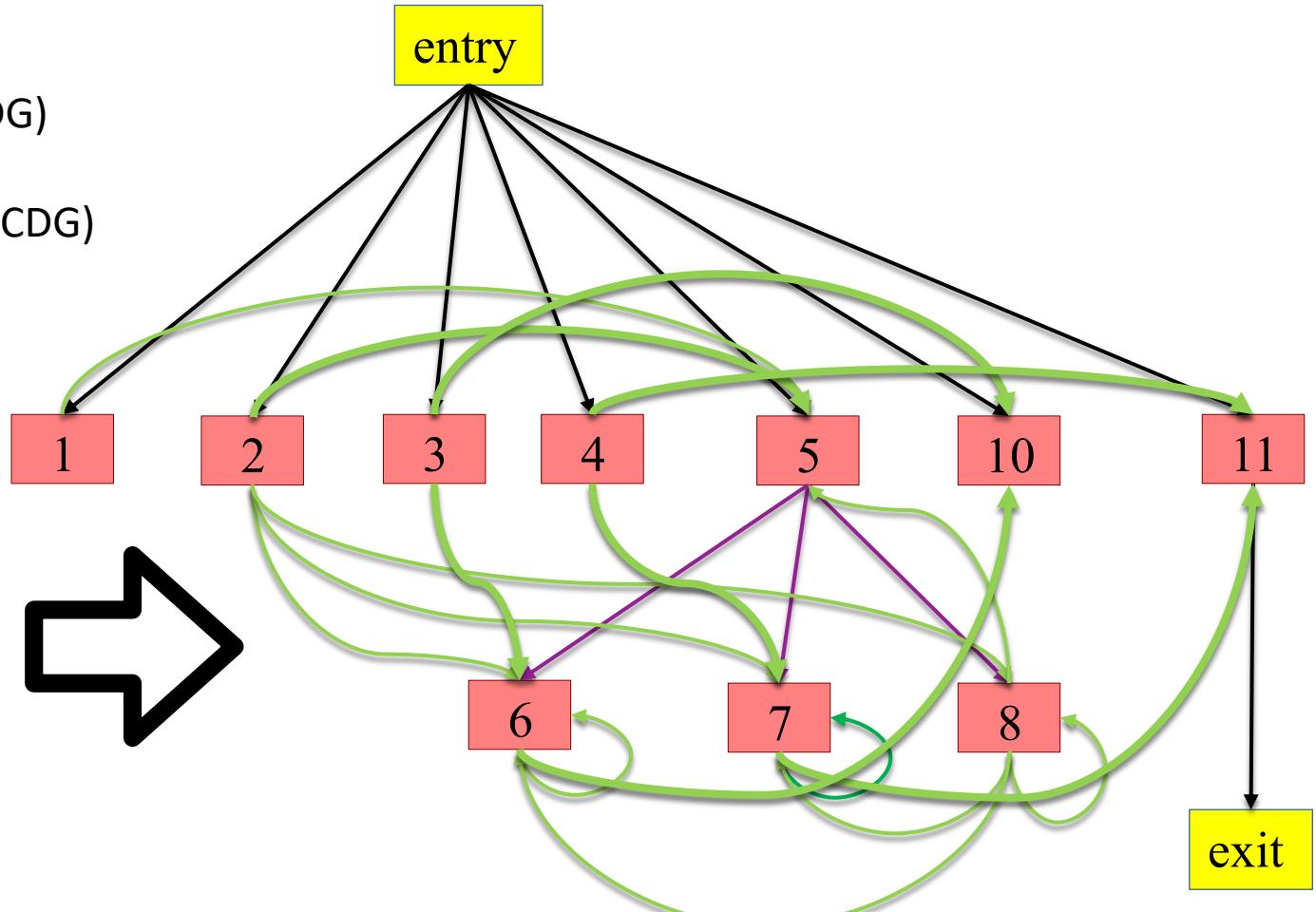
```
1. read(data);
2. i = 1;
3. sum = 0;
4. dev = 1;
5. while(i<=data){
6.     sum = sum + i;
7.     dev = dev * i;
8.     i++;
9. }
10. print(sum)
11. print(dev)
```



程序依赖图 Program Dependency Graph (PDG)

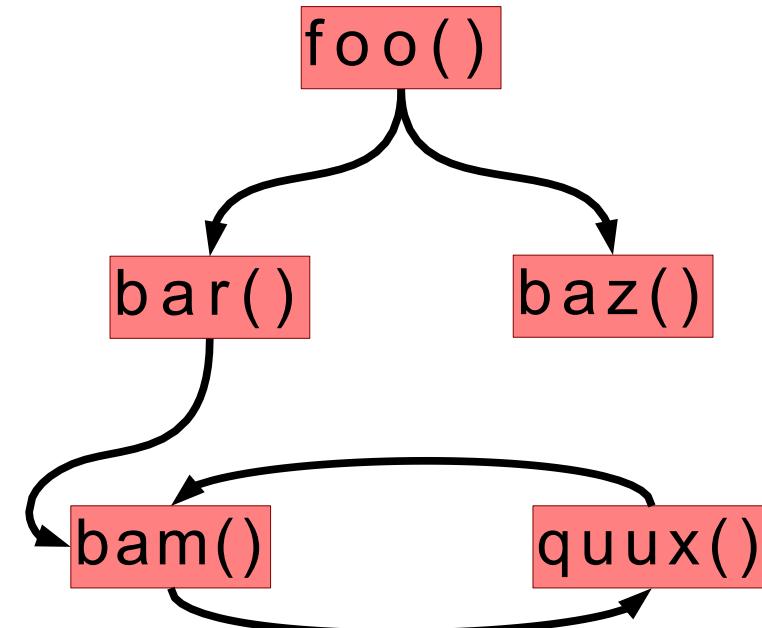
- 表示程序语句之间的依赖关系。
- 分类：
 - 数据依赖图 (Data Dependency Graph, DDG)
 - 语句间数据依赖关系
 - 控制依赖图 (Control Dependency Graph, CDG)
 - 语句间控制依赖关系

```
1. read(data);
2. i = 1;
3. sum = 0;
4. dev = 1;
5. while(i<=data){
6.     sum = sum + i;
7.     dev = dev * i;
8.     i++;
9. }
10. print(sum)
11. print(dev)
```



过程调用图 Call Graph (CG)

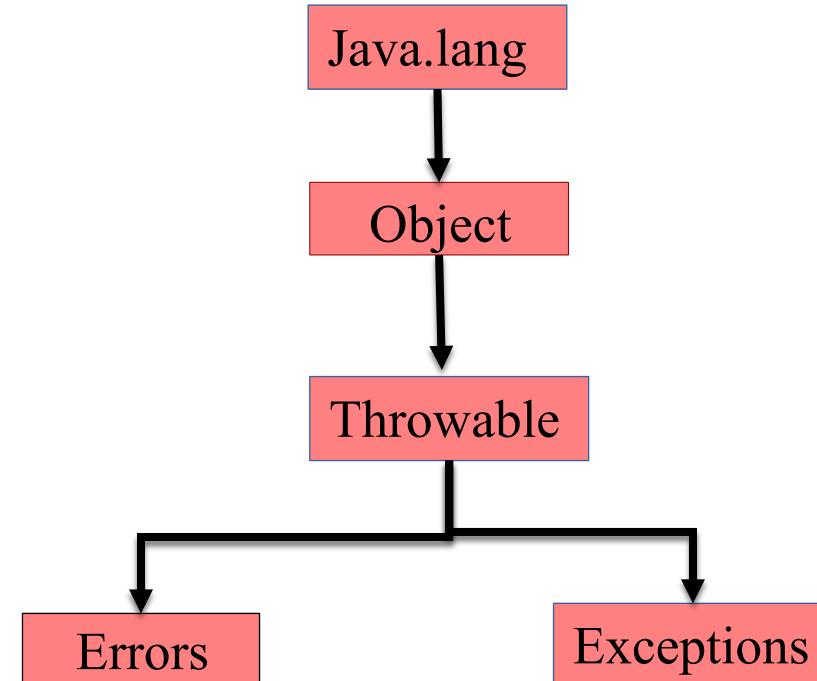
- 过程间（过程/函数/方法）调用关系的图
- 点：过程/函数/方法
- 边：所有可能的调用关系
- v.s AST 增加
 - 点：**无¹**
 - 边：**无**



1：在COREF的设计里面是抽象图

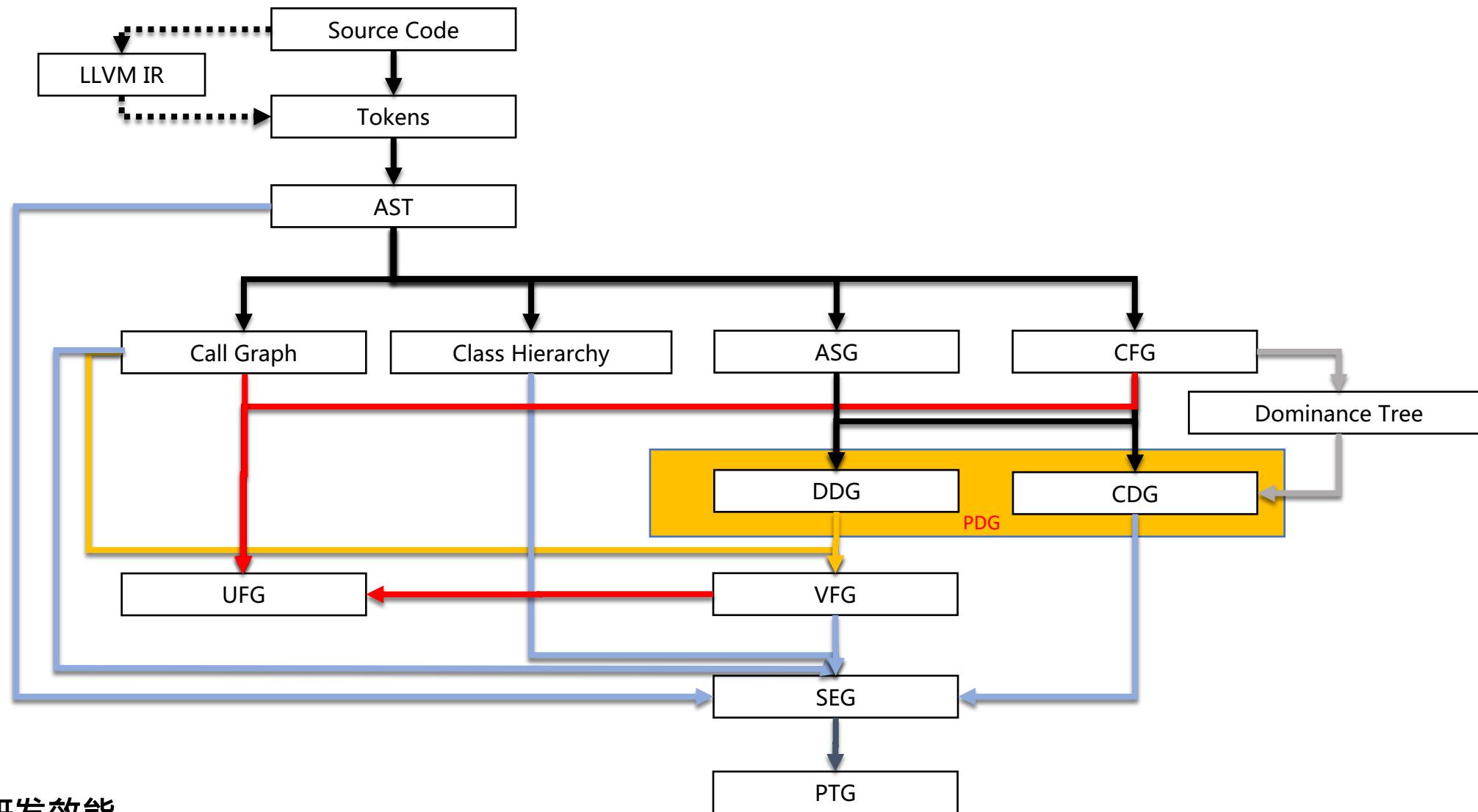
类层次结构 Class Hierarchy

- 由通过继承而彼此关联的类组成的图
 - 点：class类
 - 边：从父类指向子类
- v.s AST 增加
 - 点：**无¹**
 - 边：**无**

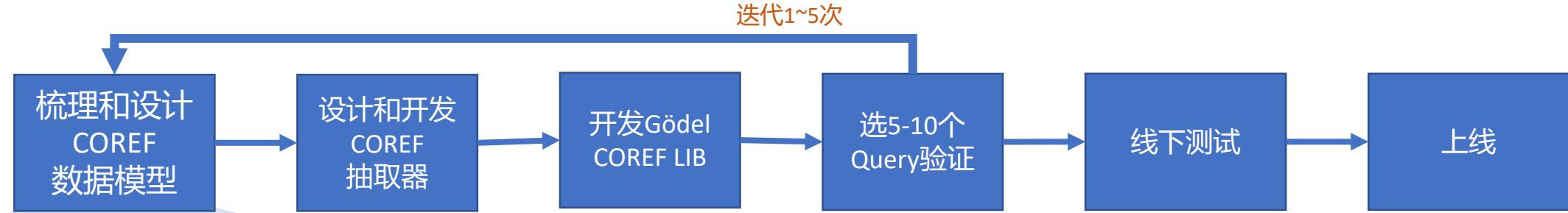


1：在COREF的设计里面是抽象图

程序分析数据转换关系



新语言支持流程



COREF Version 1 一般只支持到AST

COREF = AST + ASG + CFG + PDG +
Call Graph + Class Hierarchy +
Documentation

AST典型数据模型量级

语言种类	AST模型 节点数
Java	133
XML	13
TS/JS	246
OC/C++	397

优点：针对多种需求，一劳永逸

缺点：针对单一需求，比定制工具慢

语言支持
共6种

语言种类	状态	模型节点数
Java	成熟	133
XML	成熟	13
TS/JS	成熟	246
OC/C++	beta	397
Go	beta	173
Python	beta	165
Java Bytecode	预计下半年	
其他		

第二步



代码数据化，标准化
代码分析DSL设计

Gödel 是一个声明式语言

- 相对于命令式编程，声明式编程更加着重与描述“要什么”而把如何实现交给计算引擎。

Gödel 执行性能优越

- Gödel 的计算执行依赖高度优化的 Datalog 引擎 Souffle，配合 Gödel 语言层面的编译优化，达到优于其它技术的性能。

Gödel 是一个逻辑推理语言

- Gödel 底层实现是基于逻辑推理语言 Datalog，通过描述“事实”和“规则”，程序可以不断地推导出新的事实。

Gödel 支持高级语言特性

- 包含类，类继承，函数，模块，用于编写和维护大型代码分析程序。
- 目前实现版本：0.3

技术方案分析 – Why Gödel ?



	Java/C++	SDK	Gödel	SQL
一句话特色	不是我做，就是你做	规则下的跳舞，部分定制	学一下，你随便写，剩下我来	几乎不用学，你随便写，剩下归我
描述能力	强 – 图灵等价	弱 - 流程预置	中 – Datalog + UDF	弱 – 计算模型限制
优化责任	用户	维护者+用户	计算引擎	计算引擎
学习成本	等于啥都没提供	高	中低	低
维护难度	随缘	只能维护SDK的部分	低	低

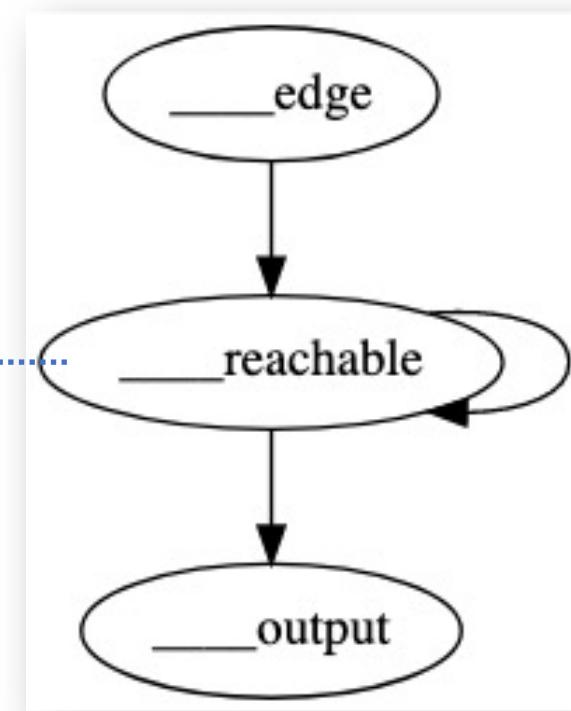
Gödel 语言样例介绍

Sample

```
predicate edge(int inVertex, int outVertex) {  
    (inVertex = 1 and outVertex = 2)  
    or  
    (inVertex = 2 and outVertex = 3)  
}  
  
predicate reachable(int a, int b) {  
    edge(a, b)  
    or  
    exists(int c) {  
        edge(a, c) and reachable(c, b)  
    }  
}  
  
predicate output(int a, int b) {  
    reachable(a, b)  
}
```

Execution Plan

递归



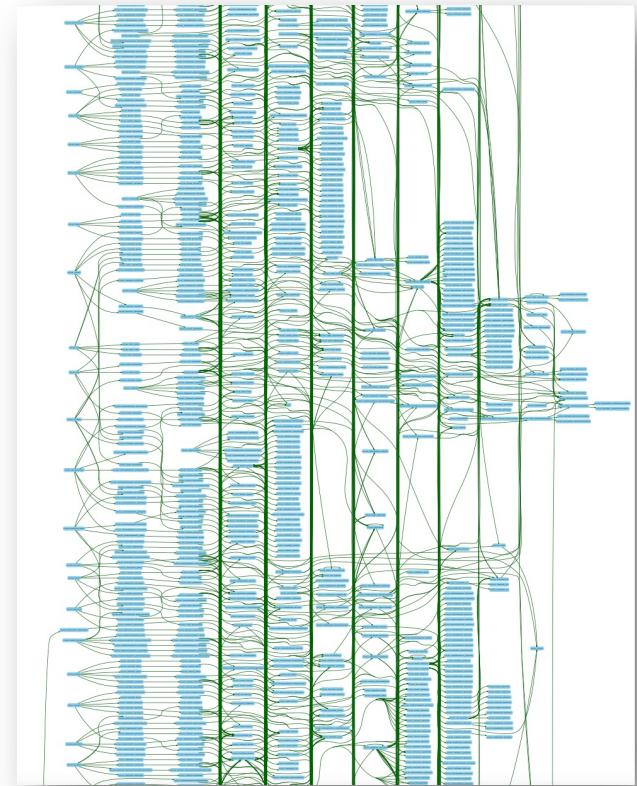
Gödel Query样例

增加了 Class, Module 等现代语言概念

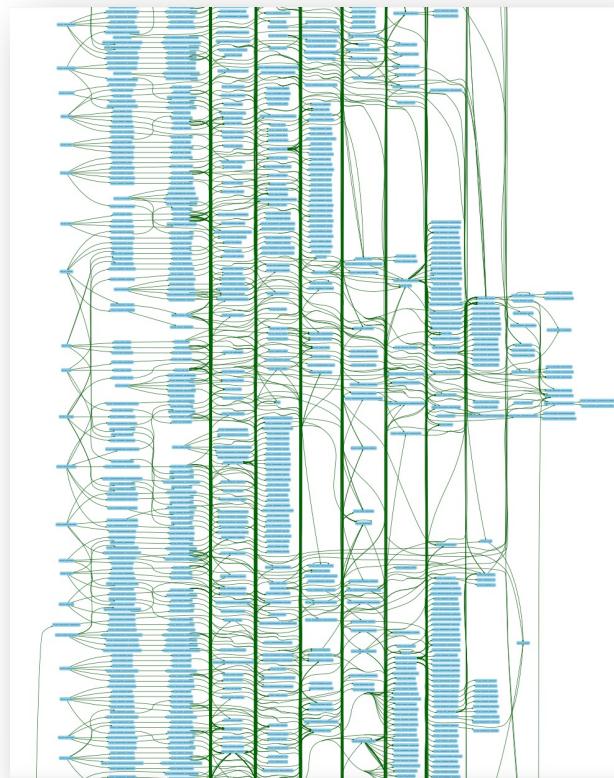
```
import coref.java.*  
  
//输出带有DAttribute注解的字段的名字  
predicate output(string name){  
    exists(Field f){  
  
        f.getAnnotation().getName()="DAttribute"  
        and  
        name = f.getName()  
    }  
}
```

计算节点数
2832

Evaluation Plan(局部)



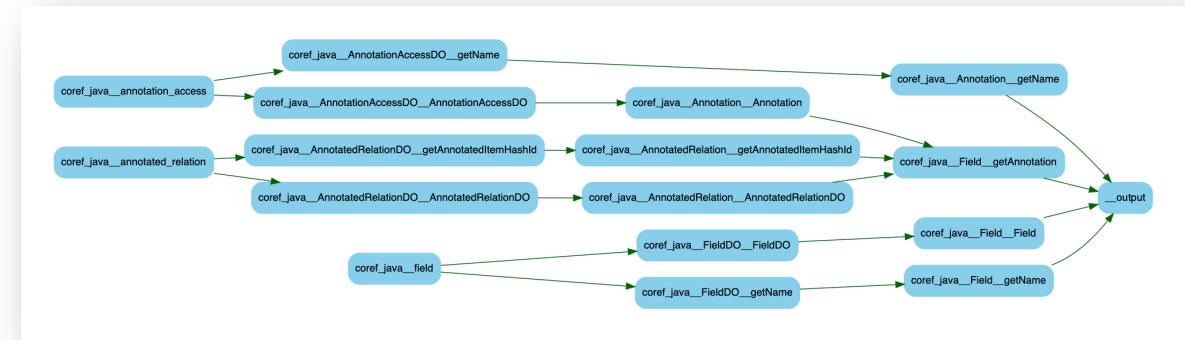
Gödel的计算模型暴露了大量优化机会



计算节点数 2832



计算节点数 17



优化前

Gödel Application



Gödel Query

Gödel Compiler

Datalog App

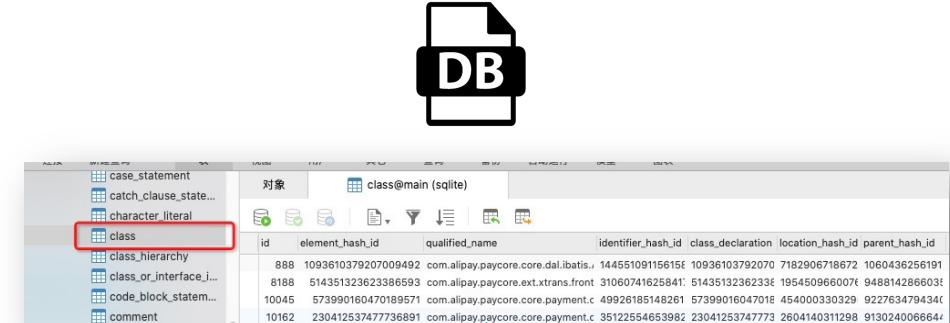
Datalog Engine

Result

```
2 import coref.java.Class
3
4 predicate output(int id, string className){
5     Class c |
6         className = c.getQualifiedName()
7         and
8         id = c
9 }
```

```
2
3 /**
4  * @filename: CLASS_MODEL
5  * @author: xiaoheng.xxh @ antgroup.
6  * @date: 2021/01/28
7  * @description: CLASS_MODEL provide
8  */
9
10 class Class ext ClassRaw{
```

```
class ClassRaw {
    ClassRaw() {
        class_raw(_,this,...,...,...)
    }
    /**
     * @description gets the qualified name of this element.
     * @return qualified name of this element, qualified:string.
     */
    string getQualifiedName() {
        return qualifiedName <- string qualifiedName |
            class_raw(_,qualifiedName,...,...,...)
    }
}
```

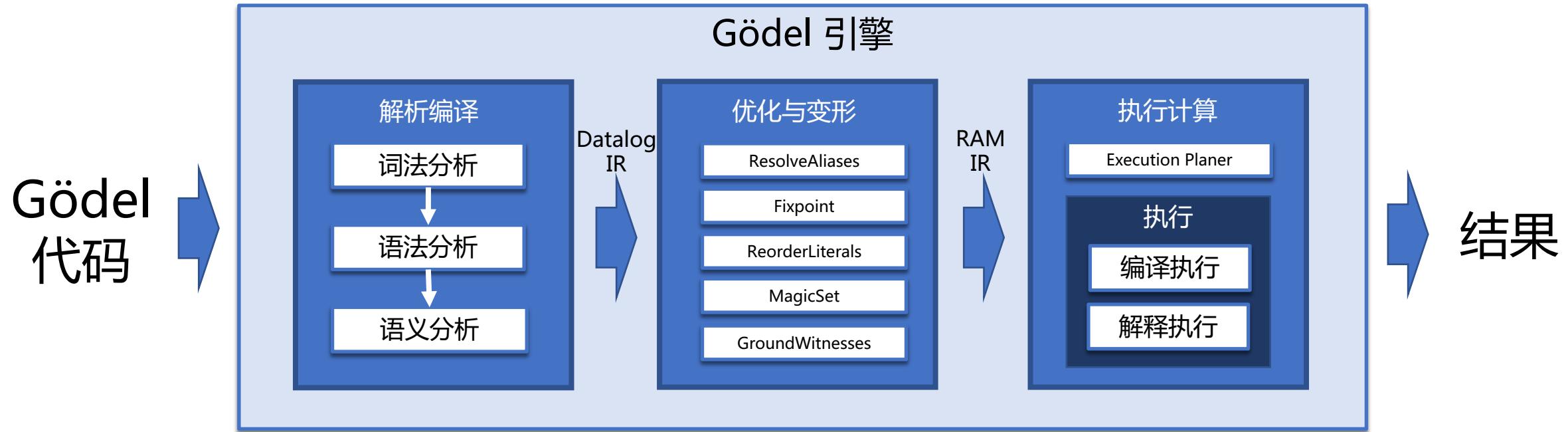


```
234 predicate class_raw(
235     int id,
236     int element_hash_id,
237     string qualified_name,
238     int identifier_hash_id,
239     int class_declaration_hash_id,
240     int location_hash_id,
241     int parent_hash_id)
242 { input("coref_java_src.db", "class")}
```



蚂蚁研发效能

Gödel 执行引擎



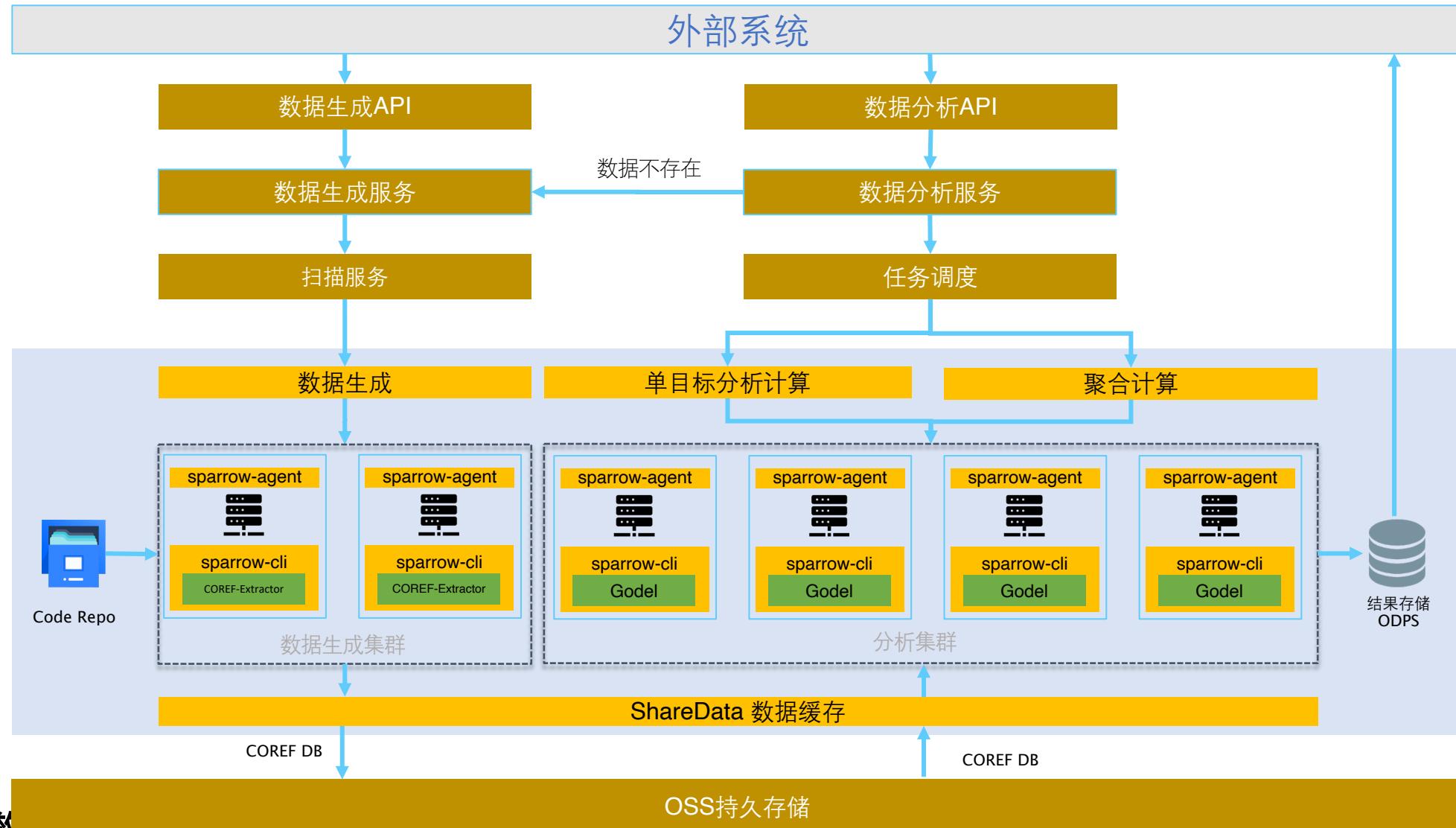
第三步



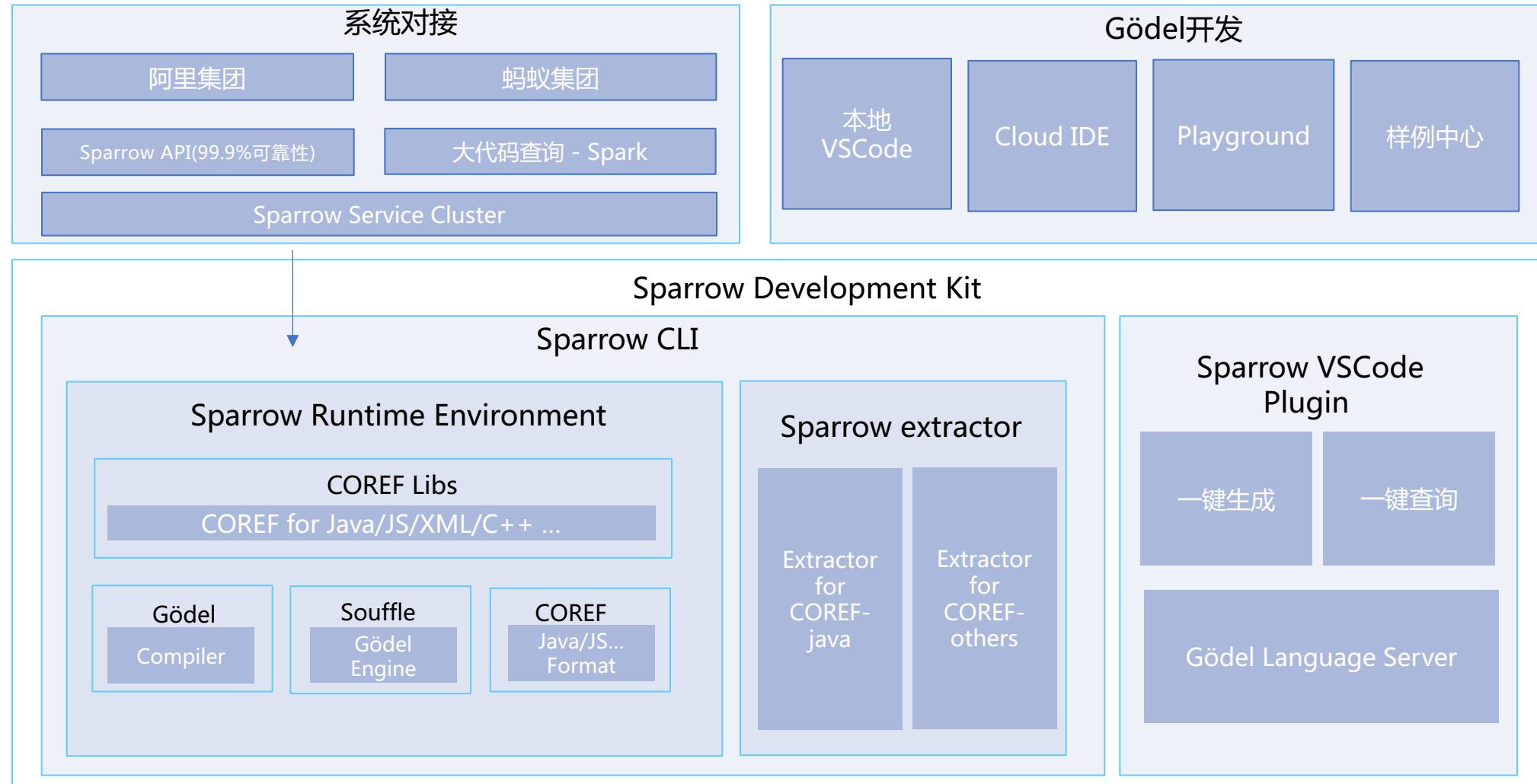
代码数据化，标准化
代码分析DSL设计
平台化，产品化



Sparrow 任务调度



Sparrow 系统组件



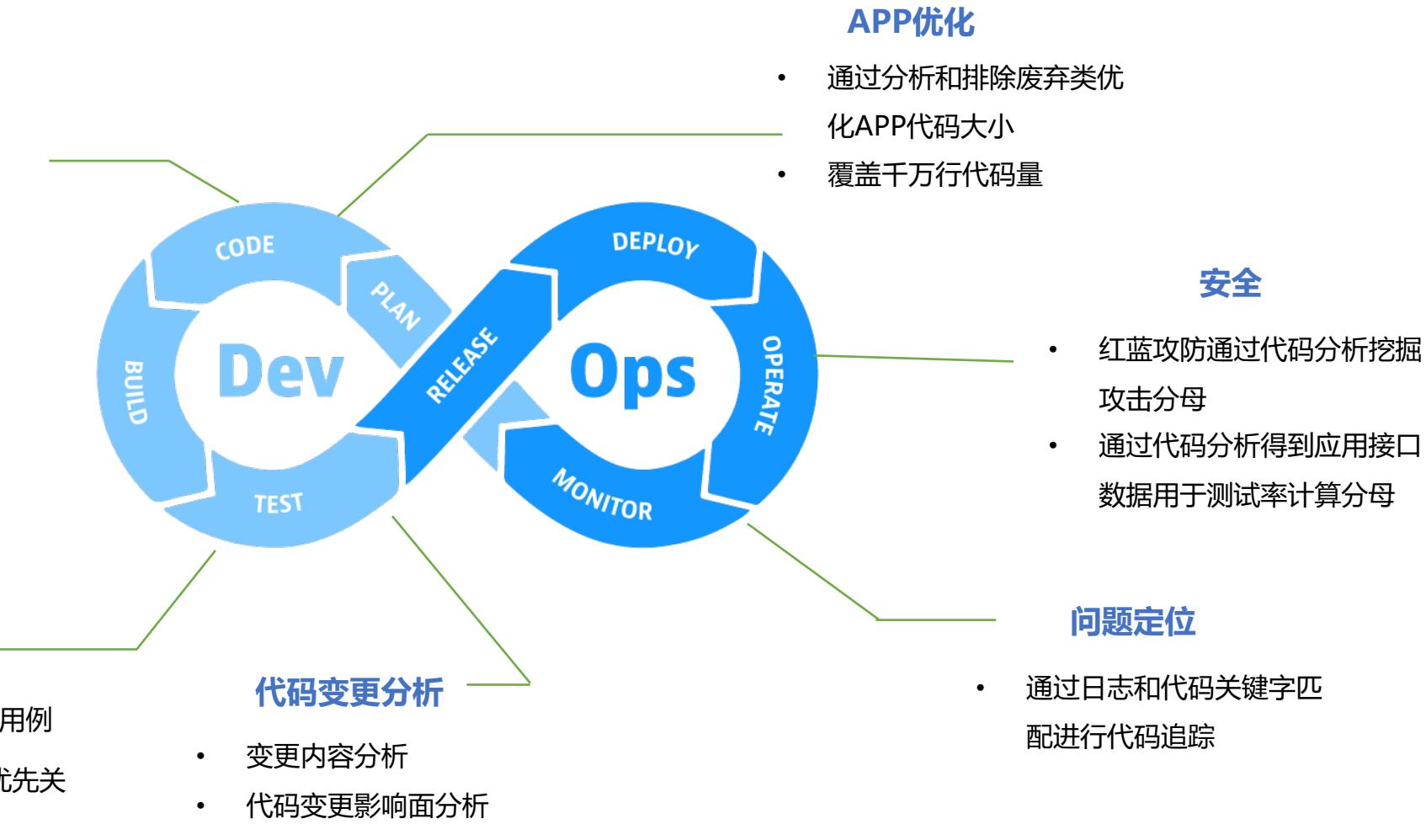


Sparrow - 样例学习中心



- Demo

已落地场景



总结一下



- 1 代码数据化，标准化
- 2 代码分析DSL设计
- 3 平台化，产品化

搞定

万级仓库，秒级响应

Demo



麦思博(msup)有限公司是一家面向技术型企业的培训咨询机构，携手2000余位中外客座导师，服务于技术团队的能力提升、软件工程效能和产品创新迭代，超过3000余家企业续约学习，是科技领域占有率第1的客座导师品牌，msup以整合全球领先经验实践为己任，为中国产业快速发展提供智库。

高可用架构主要关注互联网架构及高可用、可扩展及高性能领域的知识传播。订阅用户覆盖主流互联网及软件领域系统架构技术从业人员。高可用架构系列社群是一个社区组织，其精神是“分享+交流”，提倡社区的人人参与，同时从社区获得高质量的内容。

