

# Tai-e 太阿框架: An Introduction

Java 的静态分析框架



主页: <https://github.com/pascal-lab/Tai-e>

# Tai-e 太阿框架



## 主要内容

- 太阿中间表示
- 程序分析管理
- 程序分析实例



# Tai-e 太阿框架

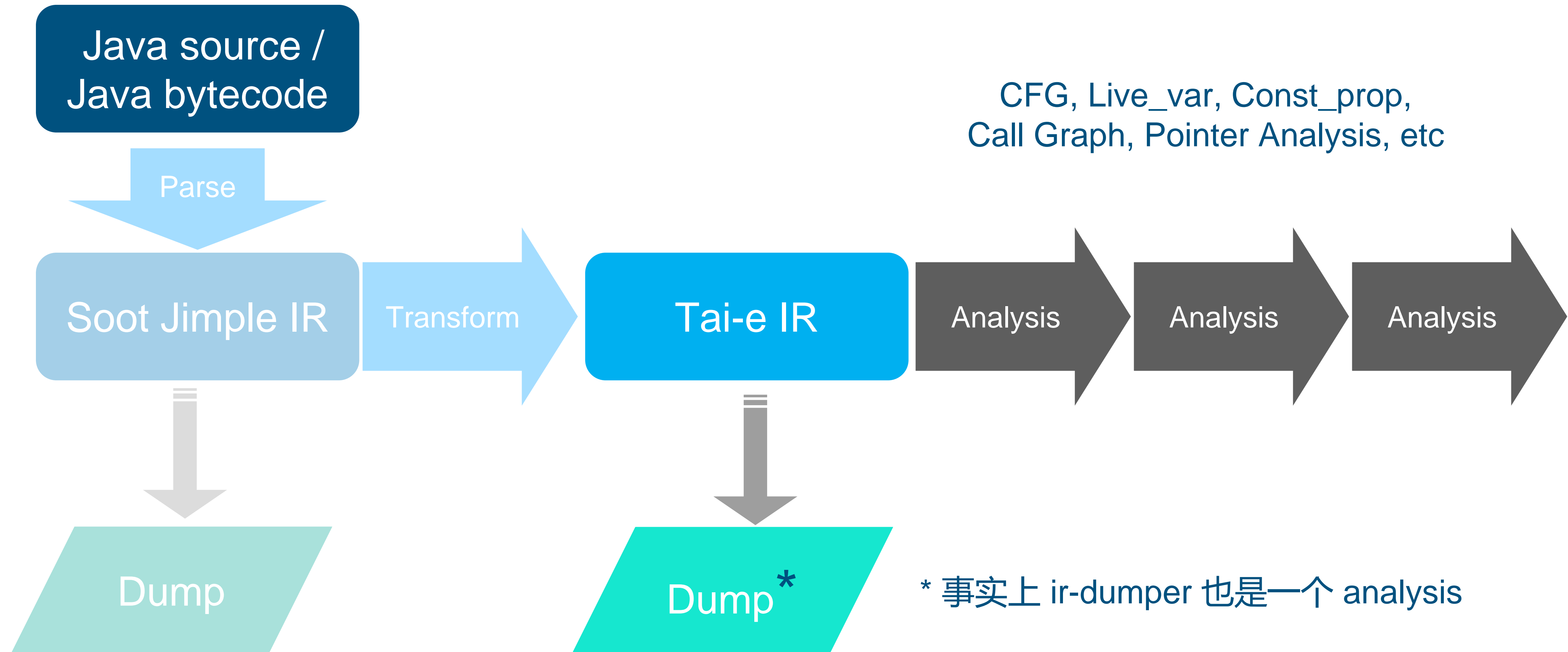


## 太阿中间表示 Tai-e IR

Tai-e IR is **typed**, **3-address**, **statement** and **expression** based representation of Java **method** body

- 1.How is Tai-e IR obtained?
- 2.How is Tai-e IR encoded?
- 3.How is Tai-e IR consumed?

# Tai-e 框架工作流程



# 上机实践1

## Tai-e Install & IR Dumper

**Prerequisite: Java 17 (OpenJDK17 Recommended)**

- Tai-e Jar Release
  - Download path/to/dir <https://github.com/pascal-lab/Tai-e/releases/download/v0.2.2/tai-e-all-0.2.2.jar>
  - User:path/to/dir\$ git clone <https://github.com/pascal-lab/java-benchmarks>
  - User:path/to/dir\$ java -jar tai-e-all-0.2.2.jar -a ir-dumper -cp src/test/pku -m test.Hello

推荐使用命令行，如果你使用 IDEA 构建运行，参考 [setup-in-intellij-idea](#)

# 上机实践1

## Tai-e Install & IR Dumper

Tai-e 参数解释

- -a analysis-id 分析的id
- -cp classpath 待分析的 java class/source 路径
- -m main 待分析的 java main class
- 更多参数，可以运行 `$ java -jar tai-e-all-0.2.2.jar` 查看文档

# 上机实践1

## Tai-e Install & IR Dumper

### Directory Tree

- path/to/dir
  - src/test/pku/test
    - Hello.java (or Hello.class)
- [tai-e-all-0.2.2.jar](#)
- [java-benchmarks](#)
- output/tir
  - test.Hello.tir

```
package test;
```

```
public class Hello{  
    public static void main(String[] args) {  
        int x = 2;  
        int y = x+x;  
    }  
}
```

```
$ java -jar tai-e-all-0.2.2.jar -a ir-dumper -cp src/test/pku -m test.Hello
```

```
public class test.Hello extends java.lang.Object {
```

```
    public static void main(java.lang.String[] args) {  
        int x, y;  
        [0@L5] x = 2;  
        [1@L6] y = x + x;  
        [2@L6] return;  
    }
```

```
    public void <init>() {  
        [0@L3] invokespecial %this.<java.lang.Object: void <init>()>();  
        [1@L3] return;  
    }
```

```
}
```

# Tai-e IR

```
public class test.Hello extends java.lang.Object {
```

```
    public static void main(java.lang.String[] args) {
```

```
        int x, y;
```

```
        [0@L5] x = 2;
```

```
        [1@L6] y = x + x;
```

```
        [2@L6] return;
```

```
    }
```

```
    public void <init>() {
```

```
        [0@L3] invokespecial %this.<java.lang.Object: void <init>()>();
```

```
        [1@L3] return;
```

```
    }
```

```
}
```

JClass

JMethod

超类

子类

Stmt

AssignStmt x = 2;

Invoke void<init>();

...more..., see docs

Exp

Var x

BinaryExp x+x

...more..., see docs

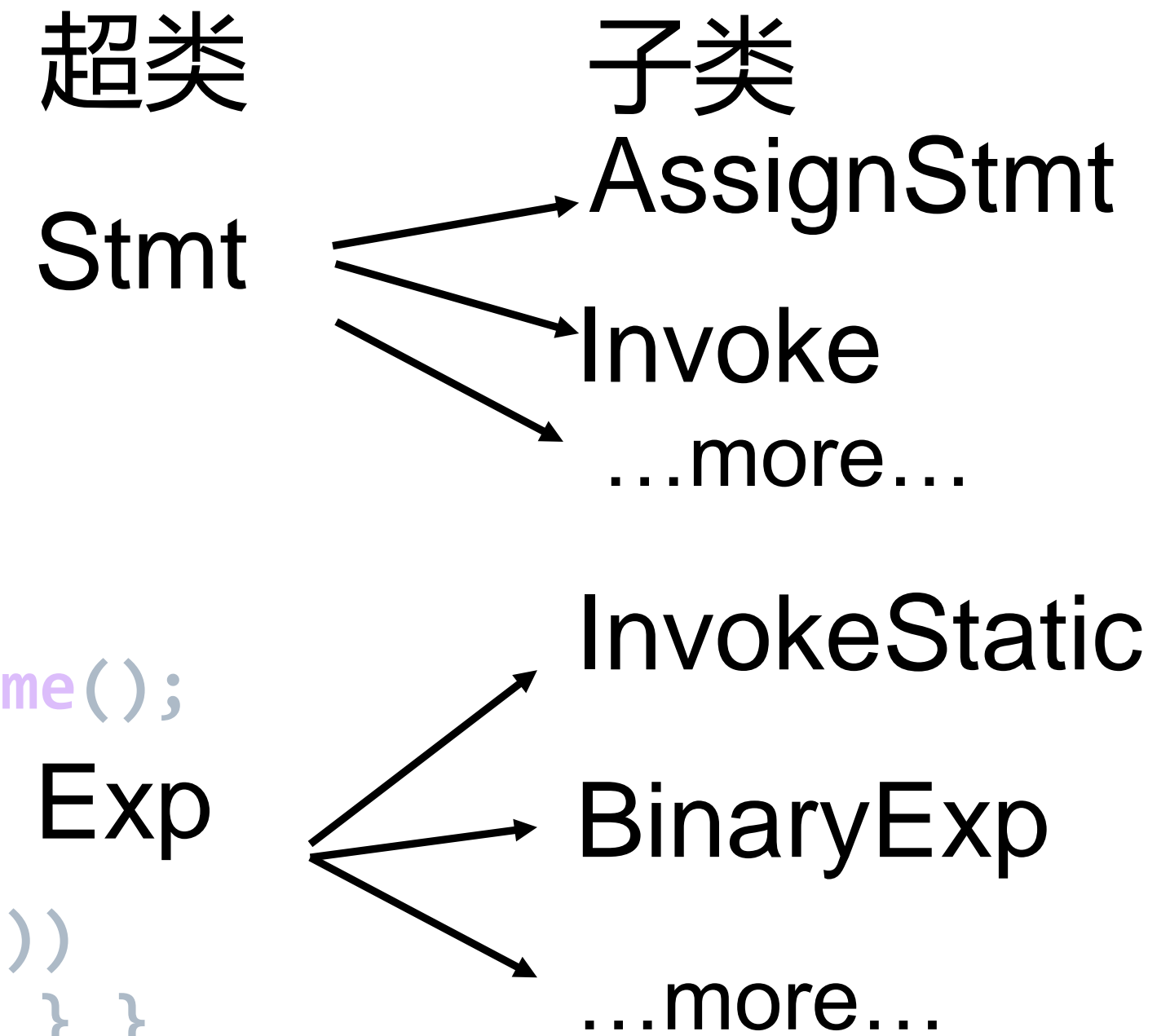
文档 <http://tai-e.pascal-lab.net/docs/current/reference/en/program-abstraction.html>



# Tai-e IR 模式匹配

```
// PreprocessResult.java
if(stmt instanceof Invoke)
{
    var exp = ((Invoke) stmt).getInvokeExp();
    if(exp instanceof InvokeStatic)
    {
        var methodRef = ((InvokeStatic)exp).getMethodRef();
        var className = methodRef.getDeclaringClass().getName();
        var methodName = methodRef.getName();
        if(className.equals("benchmark.internal.Benchmark")
            || className.equals("benchmark.internal.BenchmarkN"))
        {if(methodName.equals("alloc")) {/* do-something */ } }
    }
}
```

```
// Hello.java
BenchmarkN.alloc(1); // static method!
```



文档 <http://tai-e.pascal-lab.net/docs/current/reference/en/program-abstraction.html>

## 程序分析管理及其实现

1. Analysis Scope
2. Analysis Dependency
3. Analysis Level
4. Analysis Implementation



# 上机实践2

## Tai-e Source & Pointer Analysis Trivial

**指针分析(Pointer Analysis, PTA):** 分析指针/对象/成员指向的内存地址(may analysis)

**Prerequisite:** Java 17 (OpenJDK17 Recommended)、Gradle( <https://gradle.org/releases/> )

- Tai-e Source
  - User:path/to/dir\$ git clone This is a fork of Tai-e <https://github.com/Mepy/Tai-e/>
  - User:path/to/dir/**Tai-e**\$ git clone <https://github.com/pascal-lab/java-benchmarks>
  - User:path/to/dir/**Tai-e**\$ gradle # init
  - User:path/to/dir/**Tai-e**\$ gradle run --args="-a pku-pta-trivial -cp src/test/pku/ -m test.Hello"

推荐使用命令行，如果你使用 IDEA 构建运行，参考 [setup-in-intellij-idea](#)

# 上机实践2

## Tai-e Source & Pointer Analysis Trivial

### Directory Tree

- path/to/dir/**Tai-e**
  - src/test/pku/test
    - Hello.java
  - output/pta
    - test.Hello.pta

1 : 1, 2, 3  
2 : 1, 2, 3  
3 : 1, 2, 3

Sound but  
Nonsense 😞

```
package test;
public class Hello {

    private void main(String[] args) {
        BenchmarkN.alloc(1);
        A a = new A();
        BenchmarkN.alloc(2);
        A b = new A();
        BenchmarkN.alloc(3);
        A c = new A();
        if (args.length > 1) a = b;
        //if (args.length > 1) c = a;
        BenchmarkN.test(1, a);
        BenchmarkN.test(2, b);
        BenchmarkN.test(3, c);
    }
}
```

1 : 1, 2  
2 : 2  
3 : 3

Sound and  
Expected 😊

Your work!!

# Tai-e 分析管理

分析管理(AnalysisManager.java)

Tai-e 根据命令行参数加载全程序(-cp,-m以及**库程序**, 构成 **World**)

根据分析计划(-a, -p) 进行分析(分析之间存在**依赖**)

分析范围(scope)默认是 APP classes & methods

分析层级

- Method 方法层级的分析 : for method in scope, analysis(method)
- Class 类层级的分析 : for class in scope, analysis(class)
- World 全程序的分析 : 魔改 scope、自由分析全程序(e.g. 过程间分析)

WorldBuilder starts ...

**5496 classes** with **52319 methods**  
in the world

WorldBuilder finishes, elapsed  
time: 1.91s

pku-pta-trivial starts ...

**4 classes** in scope (APP) of class  
analyses

文档 <http://tai-e.pascal-lab.net/docs/current/reference/en/develop-new-analysis.html>



# Tai-e 分析

## Method 方法层级分析

### 实现

```
// Preprocess.java
package pku;
public class Preprocess extends MethodAnalysis<PreprocessResult>
{
    public static final String ID = "pku-pta-preprocess";
    public Preprocess(AnalysisConfig config) {
        super(config);
    }
    @Override
    public PreprocessResult analyze(IR ir) {
        var result = new PreprocessResult();
        // ir.getResult(Dep.ID); // 获取依赖分析的结果, Tai-e 会先执行 Dep 再执行 Preprocess
        result.analysis(ir);
        return result;
    }
}
```

### 注册

```
// resources/tai-e-analysis.yml
- description: describe it!
  analysisClass: pku.Preprocess
  id: pku-pta-preprocess
  requires: [ dep ] // dependencies

- description: describe it!
  analysisClass: pku.Dep
  id: dep
```

文档 <http://tai-e.pascal-lab.net/docs/current/reference/en/develop-new-analysis.html>

# Tai-e 分析

方法层级分析的例子：过程内常量传播(ConstantPropagation.java)

注册

```
// resources/tai-e-analysis.yml
- description: constant propagation
  analysisClass: pascal.taie.analysis.dataflow.analysis.constprop.ConstantPropagation
  id: const-prop
  requires: [ cfg ]
  options:
    edge-refine: true # refine lattice value via edge transfer

public class CFGBuilder extends MethodAnalysis<CFG<Stmt>> {
  public static final String ID = "cfg";
  @Override
  public CFG<Stmt> analyze(IR ir) { /* do-something */ }
}
```

分析结果

为了熟悉 Tai-e 框架(以及数据流分析)，你可以仿照常量传播，实现符号分析。  
分析结果的类型应为 `MapFact<Var, Symbol>`，其中 `[[ Symbol ]]` = { 正, 负, 未, 零 }

# Tai-e 全程序分析

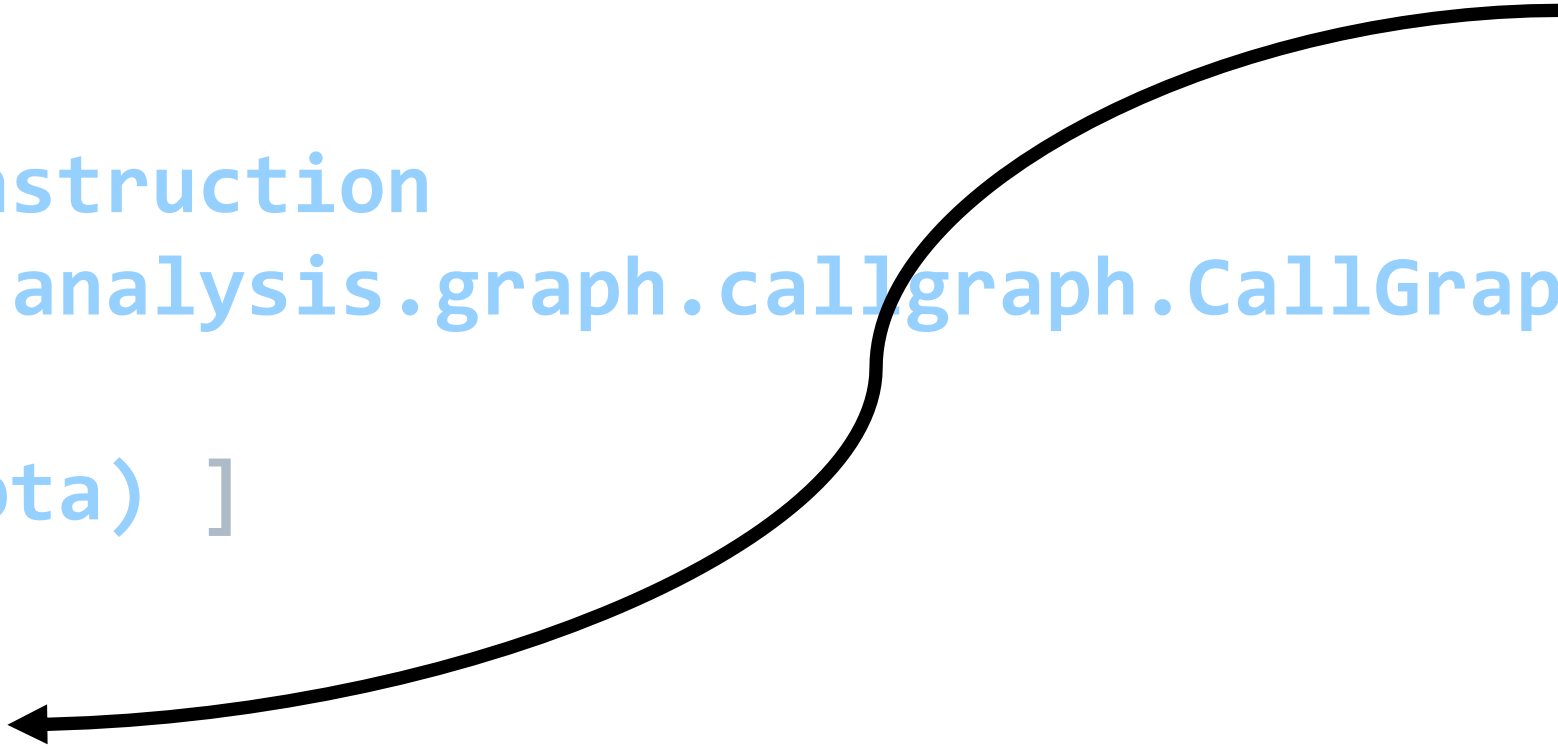
e.g.过程间常量传播

过程间分析需要调用图(Call Graph, CG), 这是全程序层级的分析

Tai-e 框架实现了 CG , 但其默认设置使用指针分析(PTA)来实现

如果你的指针分析需要使用 Tai-e 的 CG, 你必须将算法改为 **CHA** (类层次结构分析)

```
- description: call graph construction
  analysisClass: pascal.taie.analysis.graph.callgraph.CallGraphBuilder
  id: cg
  requires: [ pta(algorithm=pta) ]
  options:
    algorithm: pta # | cha
    dump: false # whether dump call graph in dot file
    dump-methods: false # whether dump reachable methods
    dump-call-edges: false # whether dump call edges
```



# Tai-e 全程序分析

User:path/to/dir/**Tai-e**\$ gradle run --args="-a **pku-pta** -cp src/test/pku/ -m **test.Hello**"

```
public class PointerAnalysis extends ProgramAnalysis<PointerAnalysisResult>
{
    public static final String ID = "pku-pta";
    @Override
    public PointerAnalysisResult analyze() {
        var result = new PointerAnalysisResult();
        var world = World.get();
        var main = world.getMainMethod();
        var jclass ← main.getDeclaringClass();
        /*
            • 从main开始遍历
            • 构建调用图
            • 进行指针分析
        */
        return result;
    }
}
```



# 参考资料

- 部分页面修改自 SA 2022 [Soot.pptx](#)
- Tai-e 实验 [手册](#)
- Tai-e: A Static Analysis Framework for Java by Harnessing the Best Designs of Classics
  - 代码 [repo](#)
  - 文档 [docs](#)

← READ THIS



**Q & A**