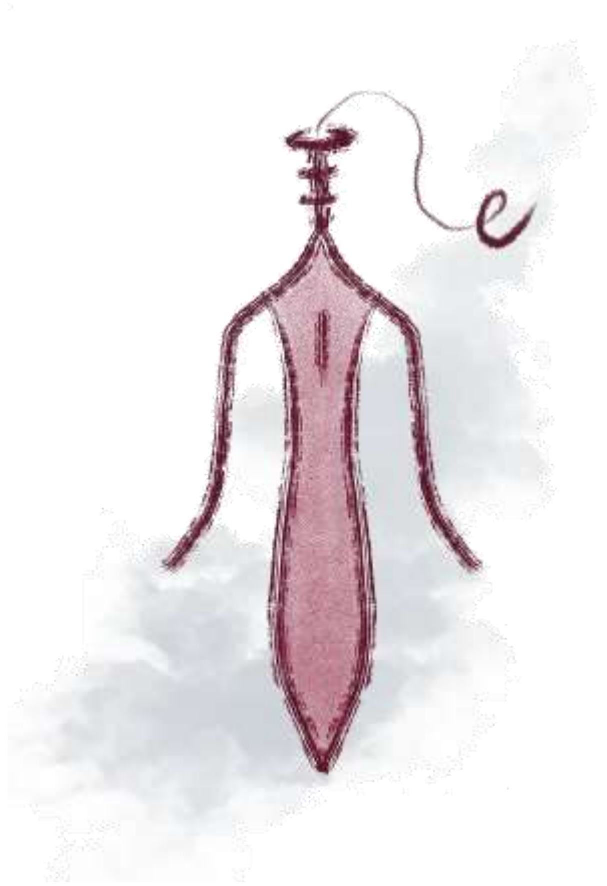
Tai-e Java程序分析框架

智旭生 2024.10.22

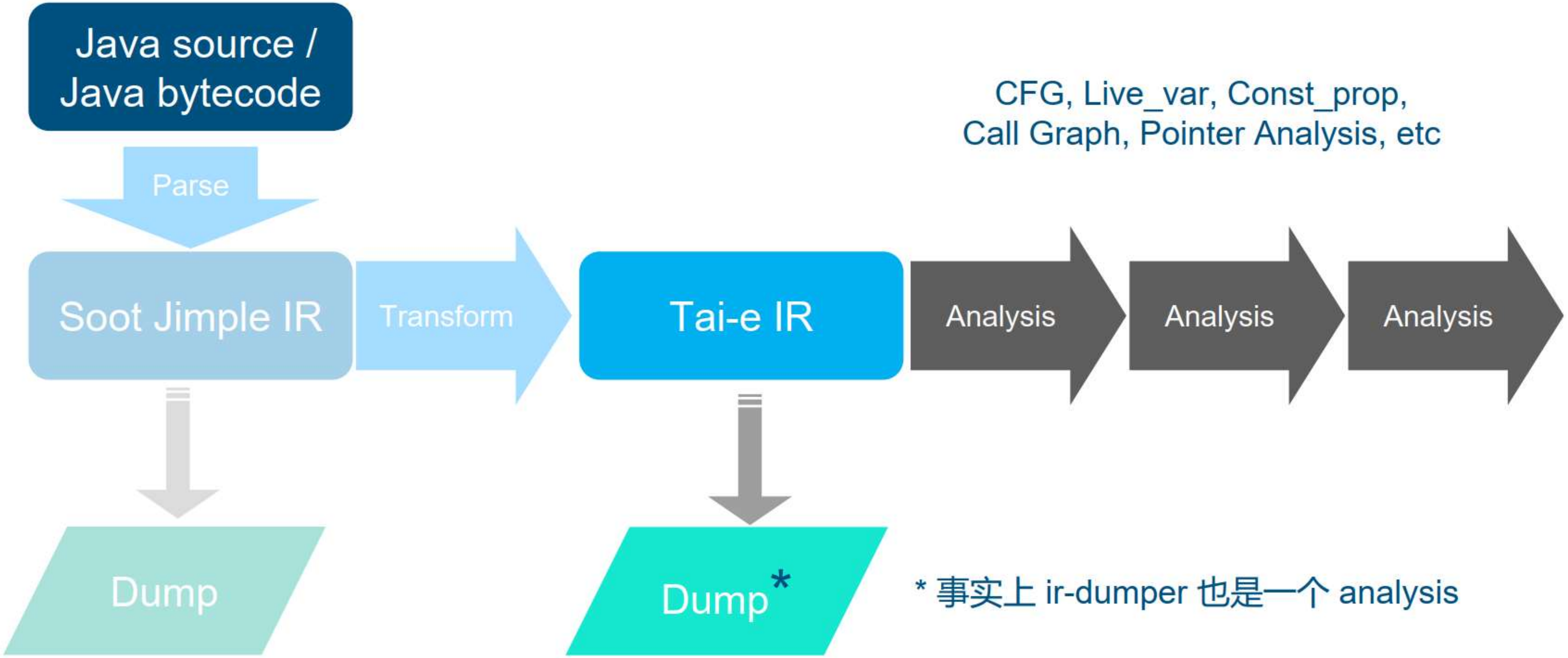
Intro: What is Tai-e?

- Tai-e is a new static analysis framework for Java, which features arguably the "best" designs from both the novel ones we proposed and those of classic frameworks such as Soot, WALA, Doop, and SpotBugs.



- 代码: https://github.com/pascal-lab/Tai-e
- 文档: https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/index.html
- API: https://tai-e.pascal-lab.net/docs/0.2.2/api/index.html

Tai-e 框架工作流程



* 事实上 ir-dumper 也是一个 analysis

上机实践1： 安装 && IR-Dumper

- 1. 下载代码，建议直接从项目包中解压
- 2. 在本地环境中安装OpenJDK17、Gradle
- 3. 在Tai-e目录下执行:
  - git clone https://github.com/pascal-lab/java-benchmarks
- 4. 在Tai-e目录下执行：
  - gradle run --args="-a ir-dumper -cp src/test/pku -m test.Hello"
  - 如果显示successful，说明运行成功，可在output/tir下找到输出

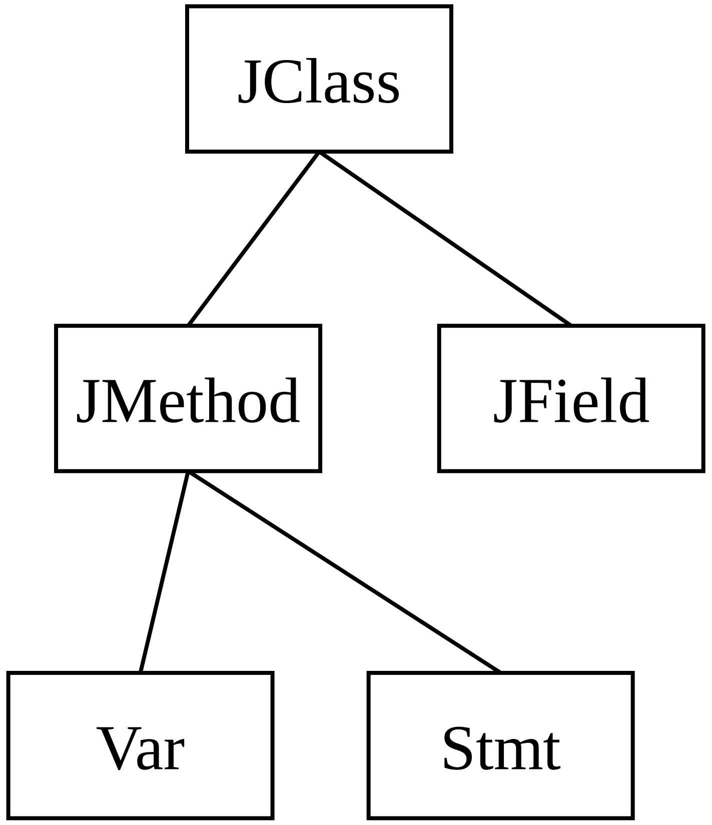
上机实践1：安装 && IR-Dumper

- 例如：

```
public class Example {
  static int f1;
  int f2;
  Example() { f1 = f1 + 1; }
  public static void main(String[] args){
    int x = 10;
    int y = x + f1;
    return;
  }
}
```

```
public class test.Example extends java.lang.Object {
  static int f1;
  int f2;
  void <init>() {
    int temp$0, %intconst0, temp$1;
    [0@L6] invokespecial %this.<java.lang.Object: void <init>()>();
    [1@L6] temp$0 = <test.Example: int f1>;
    [2@L6] %intconst0 = 1;
    [3@L6] temp$1 = temp$0 + %intconst0;
    [4@L6] <test.Example: int f1> = temp$1;
    [5@L6] return;
  }
  public static void main(java.lang.String[] args) {
    int x, temp$1, y;
    [0@L8] x = 10;
    [1@L9] temp$1 = <test.Example: int f1>;
    [2@L9] y = x + temp$1;
    [3@L10] return;
    [4@L10] return;
  }
}
```

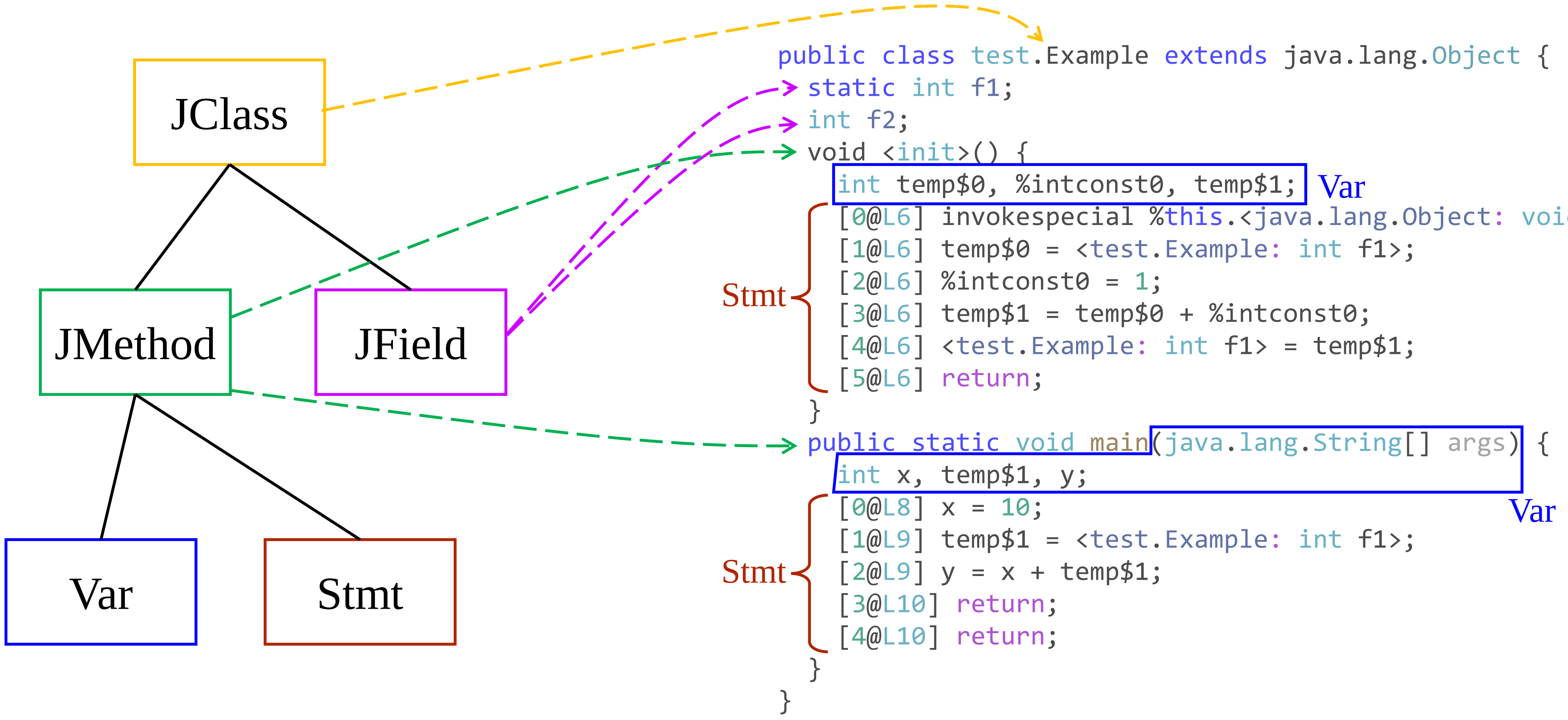
Program Abstraction and Tai-e IR



```
public class test.Example extends java.lang.Object {
  static int f1;
  int f2;
  void <init>() {
    int temp$0, %intconst0, temp$1;
    [0@L6] invokespecial %this.<java.lang.Object: void <init>()>();
    [1@L6] temp$0 = <test.Example: int f1>;
    [2@L6] %intconst0 = 1;
    [3@L6] temp$1 = temp$0 + %intconst0;
    [4@L6] <test.Example: int f1> = temp$1;
    [5@L6] return;
  }
  public static void main(java.lang.String[] args) {
    int x, temp$1, y;
    [0@L8] x = 10;
    [1@L9] temp$1 = <test.Example: int f1>;
    [2@L9] y = x + temp$1;
    [3@L10] return;
    [4@L10] return;
  }
}
```

- 详见： https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/program-abstraction.html

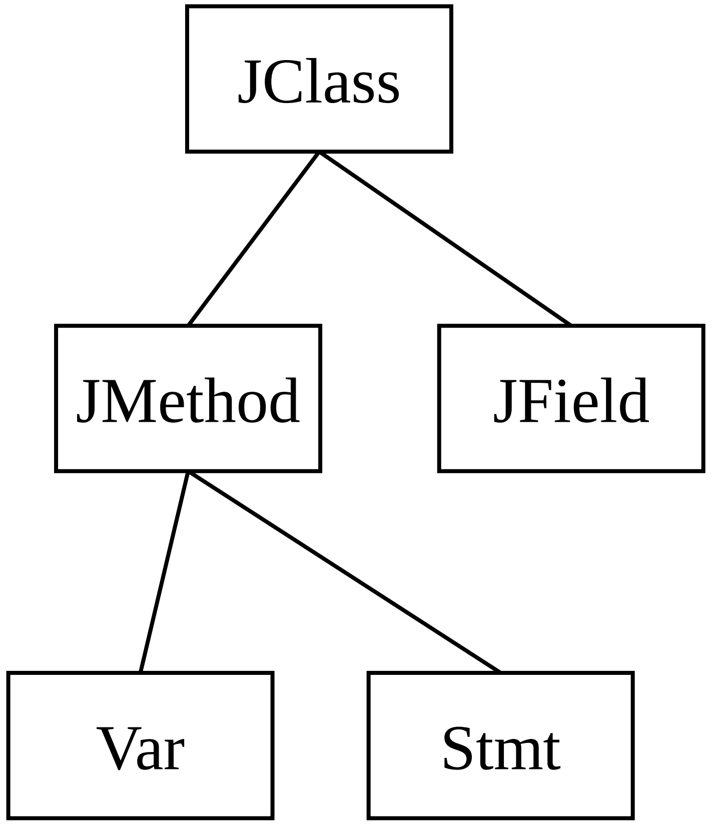
Program Abstraction and Tai-e IR



```
public class test.Example extends java.lang.Object {
  static int f1;
  int f2;
  void <init>() {
    int temp$0, %intconst0, temp$1;
    [0@L6] invokespecial %this.<java.lang.Object: void <init>()>();
    [1@L6] temp$0 = <test.Example: int f1>;
    [2@L6] %intconst0 = 1;
    [3@L6] temp$1 = temp$0 + %intconst0;
    [4@L6] <test.Example: int f1> = temp$1;
    [5@L6] return;
  }
  public static void main(java.lang.String[] args) {
    int x, temp$1, y;
    [0@L8] x = 10;
    [1@L9] temp$1 = <test.Example: int f1>;
    [2@L9] y = x + temp$1;
    [3@L10] return;
    [4@L10] return;
  }
}
```

• 详见： https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/program-abstraction.html

Program Abstraction and Tai-e IR



- Stmt可进一步分为AssignStmt， JumpStmt， Invoke等
    - 在框架中由不同的类继承interface Stmt来表示
    - 表达式Exp的表示方式与Stmt类似

- PointerAnalysisTrivial中实现了简单的程序结构遍历
    - 见src/main/java/pku/目录下的PreprocessResult.java和PointerAnalysisTrivial.java

- 详见：https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/program-abstraction.html

上机实践2：pku-pta-trivial/遍历程序结构

- 1. 在Tai-e目录下执行：
  - `gradle run --args="-a pku-pta-trivial -cp src/test/pku -m test.Hello"`
  - 如果显示successful，说明运行成功，可在Tai-e目录下找到result.txt，即为输出

- 2. 理解pku-pta-trivial如何实现程序结构的遍历

Tai-e 分析的实现与管理

- Tai-e is highly extensible. You can develop a new analysis and make it available in Tai-e.
- 分析分为三种层级：Method, Class, Program
  - 分别通过继承`MethodAnslysis`, `ClassAnalysis`, `ProgramAnalysis`来实现
- 完成一项分析的实现后，需要填写配置文件
  - 配置文件：`src/main/resources/tai-e-analyses.yml`

- 详见：https://tai-e.pascal-lab.net/docs/current/reference/en/develop-new-analysis.html

Tai-e 分析的实现与管理

- 以pku-pta-trivial为例：

```java
// PointerAnalysisTrivial.java
public class PointerAnalysisTrivial extends
ProgramAnalysis<PointerAnalysisResult> {
    public static final String ID = "pku-pta-trivial";
    public PointerAnalysisTrivial(AnalysisConfig config){
        super(config);
        ...
    }
    @Override
    public PointerAnalysisResult analyze() {
        ...
    }
}
```

```yaml
# tai-e-analyses.yml
- description: pku software analysis courses project
pointer analysis, trivial cases
  analysisClass: pku.PointerAnalysisTrivial
  id: pku-pta-trivial
  requires: [ ]
```

- 详见：https://tai-e.pascal-lab.net/docs/current/reference/en/develop-new-analysis.html

Tai-e 分析的实现与管理

- 以pku-pta-trivial为例：

```java
// PointerAnalysisTrivial.java
public class PointerAnalysisTrivial extends
ProgramAnalysis<PointerAnalysisResult> {
    public static final String ID = "pku-pta-trivial";
    public PointerAnalysisTrivial(AnalysisConfig config){
        super(config);
        ...
    }
    @Override
    public PointerAnalysisResult analyze() {
        ...
    }
}
```

```yaml
# tai-e-analyses.yml
- description: pku software analysis courses project
pointer analysis, trivial cases
  analysisClass: pku.PointerAnalysisTrivial
  id: pku-pta-trivial
  requires: [ ]
```

analyze方法的返回值类型

说明该分析是Program层级的分析

分析过程的实现

- 详见：https://tai-e.pascal-lab.net/docs/current/reference/en/develop-new-analysis.html

Tai-e 分析的实现与管理

- 以pku-pta-trivial为例：

```java
// PointerAnalysisTrivial.java
public class PointerAnalysisTrivial extends
ProgramAnalysis<PointerAnalysisResult> {
    public static final String ID = "pku-pta-trivial";
    public PointerAnalysisTrivial(AnalysisConfig config){
        super(config);
        ...
    }
    @Override
    public PointerAnalysisResult analyze() {
        ...
    }
}
```

```yaml
# tai-e-analyses.yml
- description: pku software analysis courses project
pointer analysis, trivial cases
  analysisClass: pku.PointerAnalysisTrivial
  id: pku-pta-trivial
  requires: [ ]
```

指定分析器的类

用于识别这项分析

标注出该分析的dependency

- 详见：https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/analysis-management.html

Tai-e 运行过程

- 在运行时，Tai-e会根据配置文件生成一个分析计划（即要执行的分析列表），然后按计划依次运行分析。
- 每完成一个分析后，Tai-e 会自动将结果存储在内存中。

- 课程实践要求：不能使用任何(直接或间接)依赖pta的算法

- 详见：https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/analysis-management.html

上机实践3：获取const-prop的分析结果

- 编辑pku-pta-trivial的代码和配置文件，在pku-pta-trivial分析过程中获取const-prop的分析结果，参考：
  - API文档中的`interface ResultHolder`：https://tai-e.pascal-lab.net/docs/0.2.2/api/pascal/taie/util/ResultHolder.html
  - 分析结果的管理：https://tai-e.pascal-lab.net/docs/0.2.2/reference/en/analysis-management.html

More Reference

- 南京大学《软件分析》Lab文档：https://tai-e.pascal-lab.net/intro/overview.html
- T. Tan and Y. Li, "Tai-e: A Static Analysis Framework for Java by Harnessing the Best Designs of Classics," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA 2023)*, 2023. Available: https://dl.acm.org/doi/abs/10.1145/3597926.3598120
- SA22: soot.pptx
- SA23: slides_taie.pdf

Q & A