有赞Flutter热修落地实践

同城零售移动 朱守宇

YOUZAN power

业界的方案–官方

Code Push / Hot Update / out of band updates #14330

Open eseidelGoogle opened this issue on 30 Jan 2018 · 234 comments

eseidelGoogle commented on 30 Jan 2018 • edited by Hixie Contributor

This is currently not on Flutter's roadmap, for reasons discussed in this comment:
#14330 (comment)

This comment also gives a brief overview of the various kinds of "hot update" features that you might be thinking about, and gives terminology for referring to them, which can help if you wish to communicate unambiguously about this topic:
#14330 (comment)

Often people ask if Flutter supports "code push" or "hot update" or other similar names for pushing out-of-store updates to apps.

Currently we do not offer such a solution out of the box, but the primary blockers are not technological. Flutter supports just in time (JIT) or interpreter based execution on both Android and iOS devices. Currently we remove these libraries during --release builds, however we could easily include them.

The primary blockers to this feature resolve around current quirks of the iOS ecosystem which may require apps to use JavaScript for this kind of over-the-air-updates functionality. Thankfully Dart supports compiling to JavaScript and so one could imagine several ways in which one compile parts of ones application to JavaScript instead of Dart and thus allows replacement of or augmentation with those parts in deployed binaries.

This bug tracks adding some supported solution like this. I'll dupe all the other reports here.

业界的方案–开源案例

热修没有，动态化方案有很多

动态化方案	热修需求			业界案例(部分)
	不影响正常的业务开发	能覆盖所有的业务场景	双端一致性	
解释语言(JS)下发	使用JS开发或者将源码转为JS	P	Y	MXFlutter
动态组件(DSL)	P	取决于DSL模板的丰富度	Y	头条、闲鱼
二进制更新	Y	Y	仅适用于Android	Android插件化、DynamicPatching

红色:不支持　　P:部分支持　　Y:支持

如果想写个热修，该怎么做？如何入手？

JSPatch

```
UIView *view = [[UIView alllo] init];
[view setBackgroundColor: [UIColor grayColor]];
[view setAlpha:0.5];
```

```
{
    _clsName: “UIView”,
    alloc: function() {...},
    ...
}
```

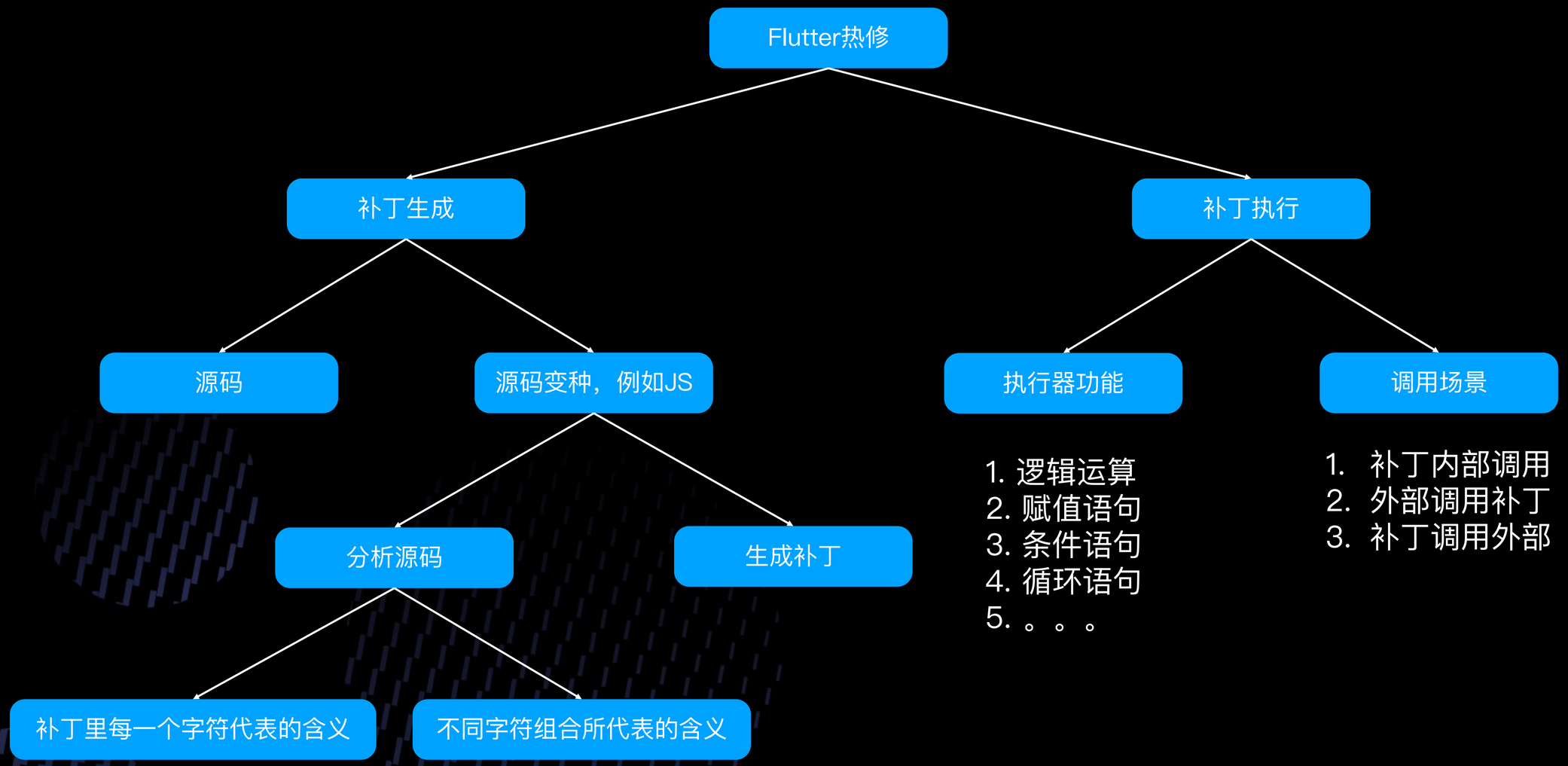
```
require(‘UIView’)
var view = UIView.alloc().init()
view.setBackgroundColor(require(‘UIColor’).grayColor())
view.setAlpha(0.5)
```

源码

热修补丁

执行补丁

热修的大致思路



Flutter热修–补丁–源码

直接将源码作为热修补丁

```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

```
/// b.dart
class SomeClass {}

/// c.dart
class SomeClass {}

/// d.dart
class SomeClass {}
```

- 手动指定是 a.dart 里的类

```
/// h.dart
class BugClass {}

/// i.dart
class BugClass {}

/// j.dart
class BugClass {}
```

- 手动补充，极为低效，业务方使用成本大
- 线上运行时候，无法确定究竟是哪个?

Flutter热修–补丁–源码

直接将源码作为热修补丁

```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.method();
 }
}
```

```
/// b.dart
class SomeClass {}

/// c.dart
class SomeClass {}

/// d.dart
class SomeClass {}
```

- 手动指定是 a.dart 里的类

- 需要明确指定是修复的哪个路径下的哪个类/方法
- 该类/方法依赖了哪些类/方法，他们又是来源于哪个路径下的
- 各个表达式的类型等关键信息

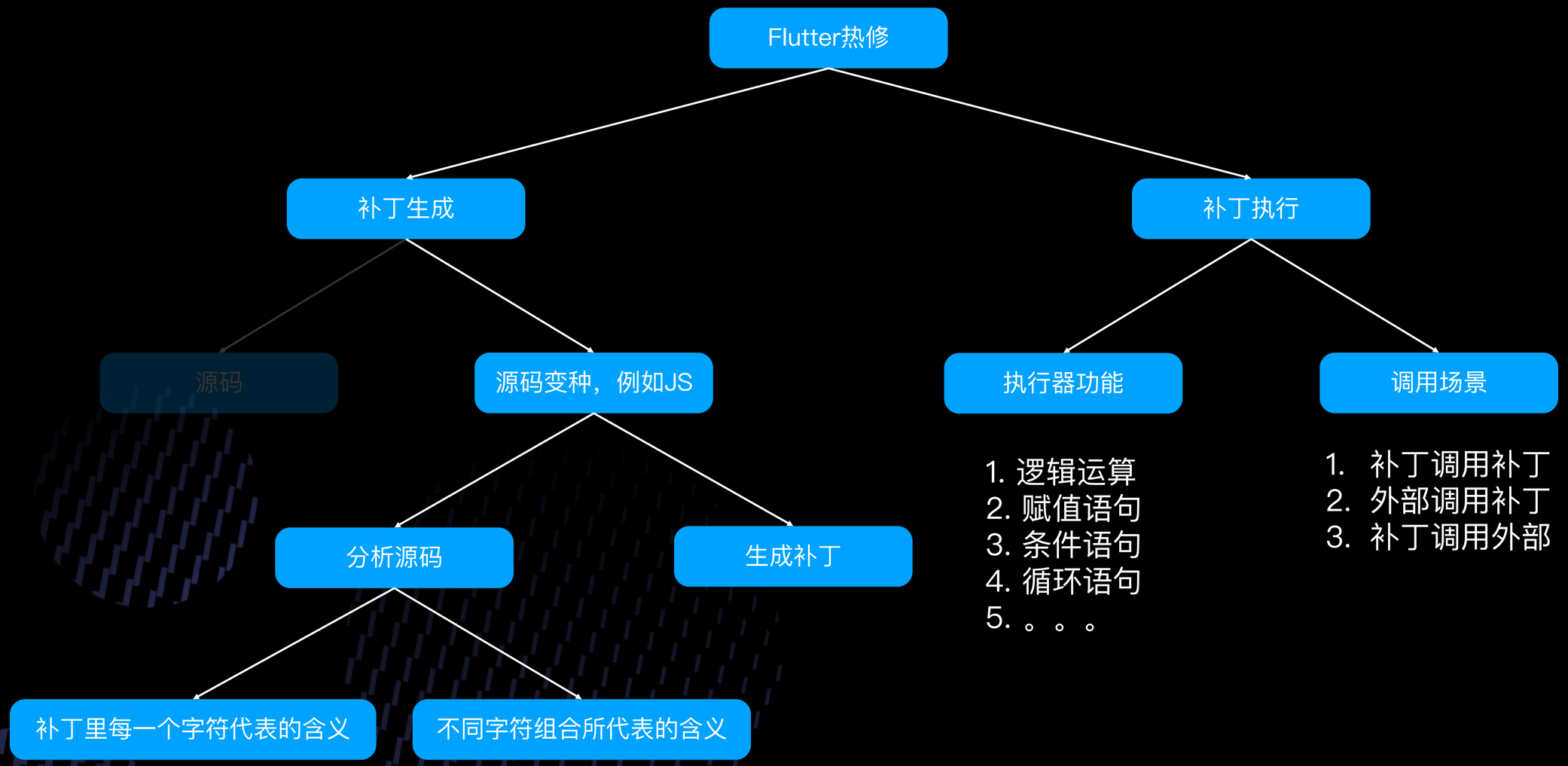
```
/// h.dart
class BugClass {}

/// i.dart
class BugClass {}

/// j.dart
class BugClass {}
```

- 手动补充，极为低效，业务方使用成本大
- 线上运行时候，无法确定究竟是哪个?

热修的大致思路



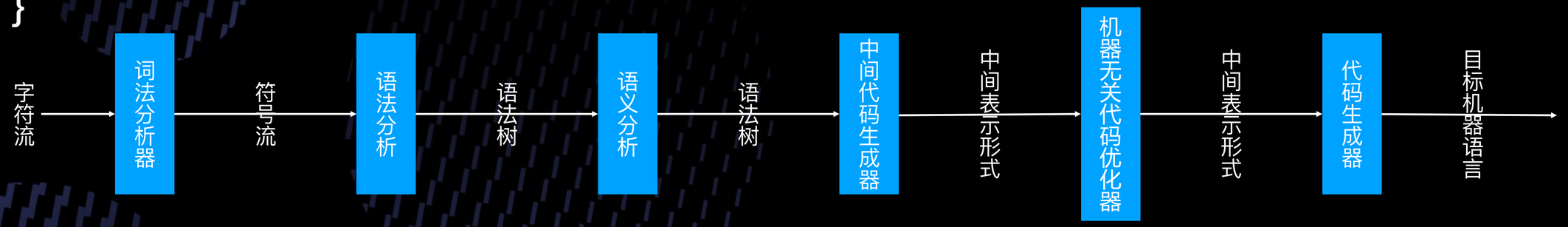
生成补丁–源码分析

```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

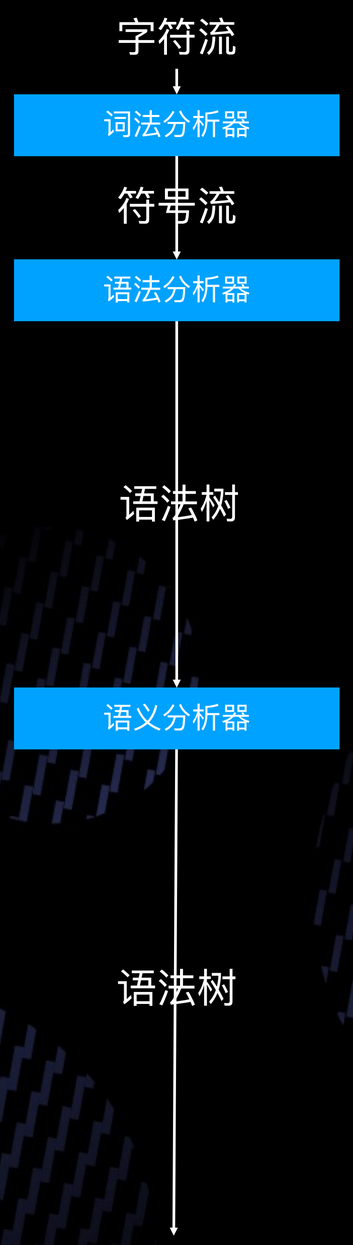
定义了一个叫 SomeClass 的class

该类有一个 someMethod 方法，该方法不需要任何参数

该方法有两个语句，一条变量声明语句、一条方法调用语句

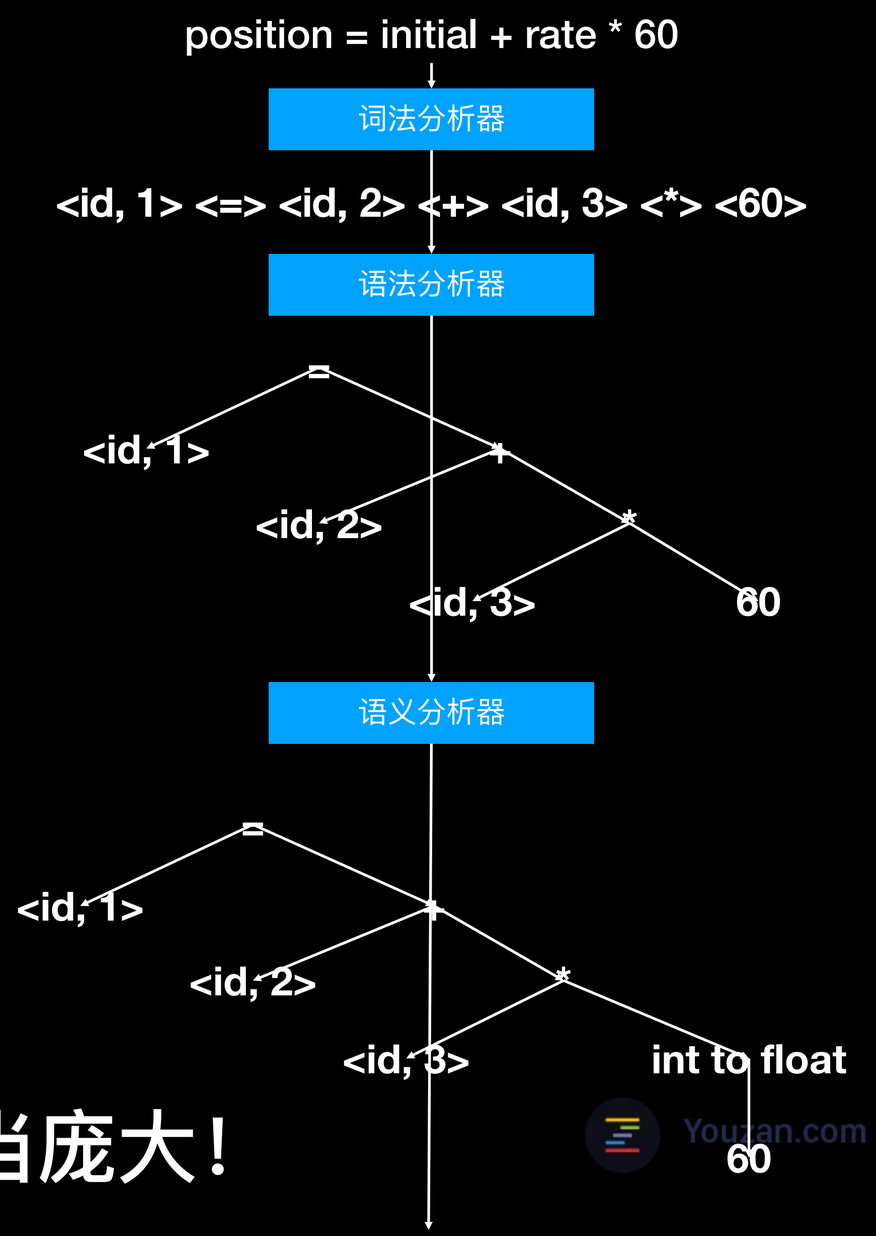


生成补丁-源码分析



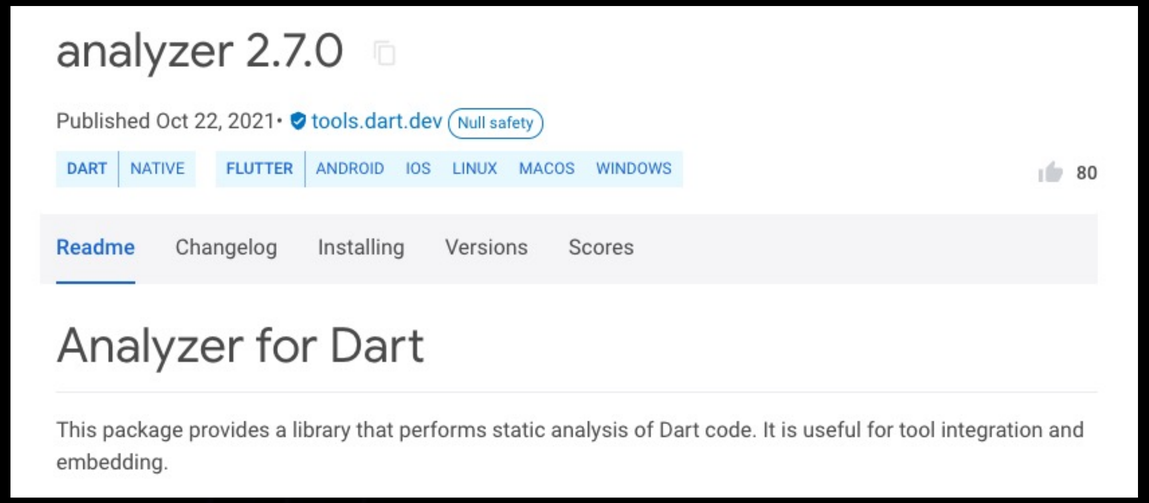
- 词法分析
  - 将字符流组织成有意义的词素序列
  - 对于每个词素，产生词法单元
    - <token–name, attribute–value>

- 语义分析
  - 检查源程序是否和语言定义的语义一致
  - 类型检查等工作
  - 类型转换等工作



手写这一套？工作量相当庞大！

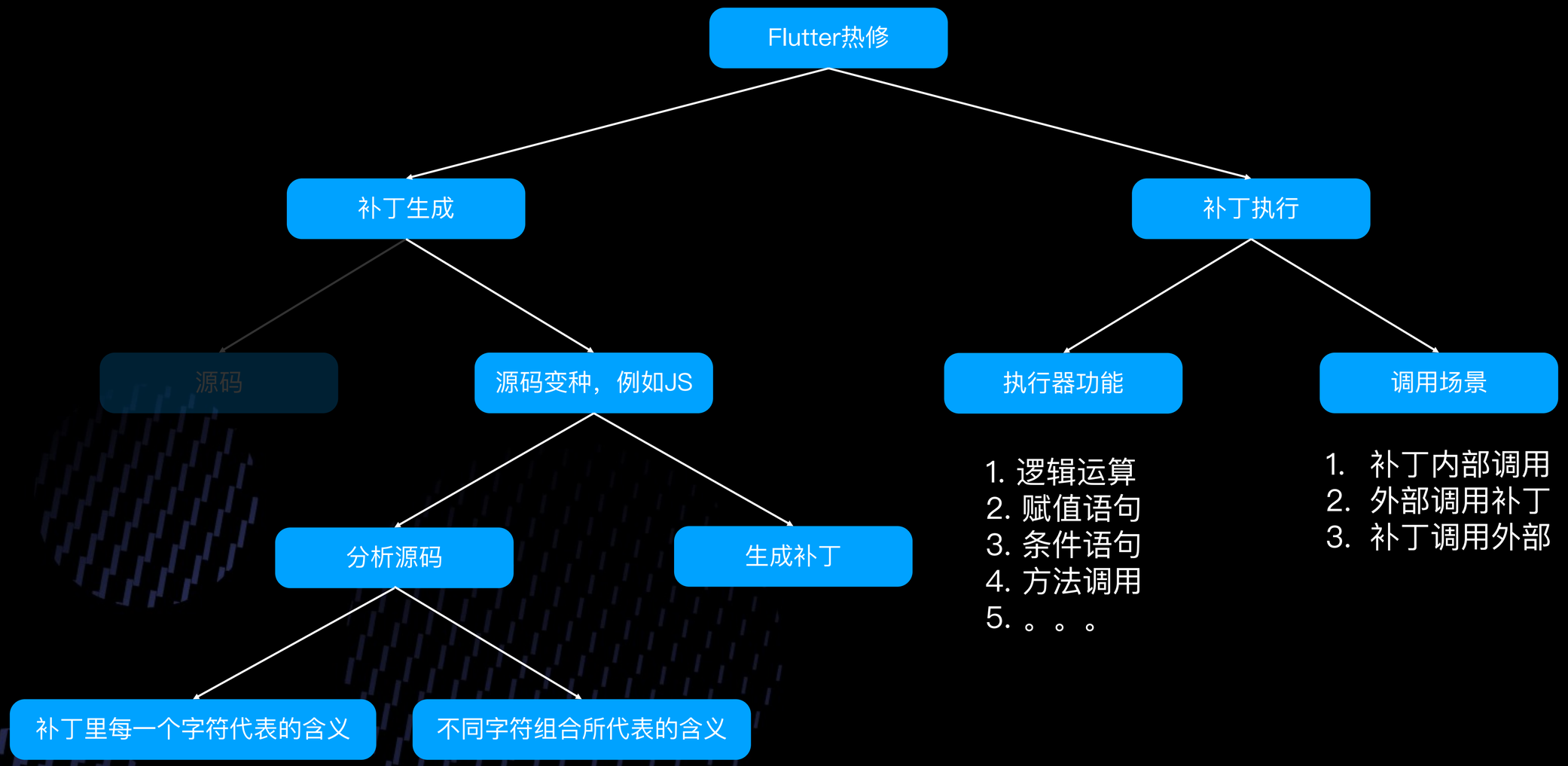
生成补丁–源码分析



json_serializable是该package的最佳实践，可以作为参考

- 同级目录下生成.g.dart文件
- 知道哪些类需要JSON序列化、包括继承/依赖关系的处理
- 所有的属性信息、如何调用构造函数

Flutter热修–执行补丁



执行器功能-四则运算

```c
# include <stdio.h>
int main() {
  float left, right, result;
  char operator;
  printf("Enter a number:");
  scanf("%f", &left);
  getchar();
  printf("Enter operator:[+, -, *, /]");
  scanf("%c", &operator);
  printf("Enter another number:");
  scanf("%f", &right);
  switch (operator) {
    case '+':
      result = left + right;
      break;
    case '-':
      result = left - right;
      break;
    case '*':
      result = left * right;
      break;
    case '/':
      result = left / right;
      break;
    default:
      break;
  }
  printf("%f %c %f = %f", left, operator, right, result);
}
```

- 每一种表达式都有其特定的规则
- 依据规则取出其各个环节的操作数
- 针对操作数完成指定的运算并返回

看一下Dart都有哪些语法，也这样支持一下。。。

执行补丁-执行器功能

130种表达式类型

binaryExpression	+、-、*、/、<、>、<=、>=、==、!=、&&、‖、<<、>>、|、&、~/、^、??...
Literal	BooleanLiteral、DoubleLiteral、IntegerLiteral、NullLiteratl、SimpleStringLiteral、SymbolLiteral、TypeLiteral

要支持这么多语法，工作量有点庞大

执行补丁–执行器功能

你会用到所有的语法吗?

实事求是，不抬杠

所有的语法都会在线上生效吗?

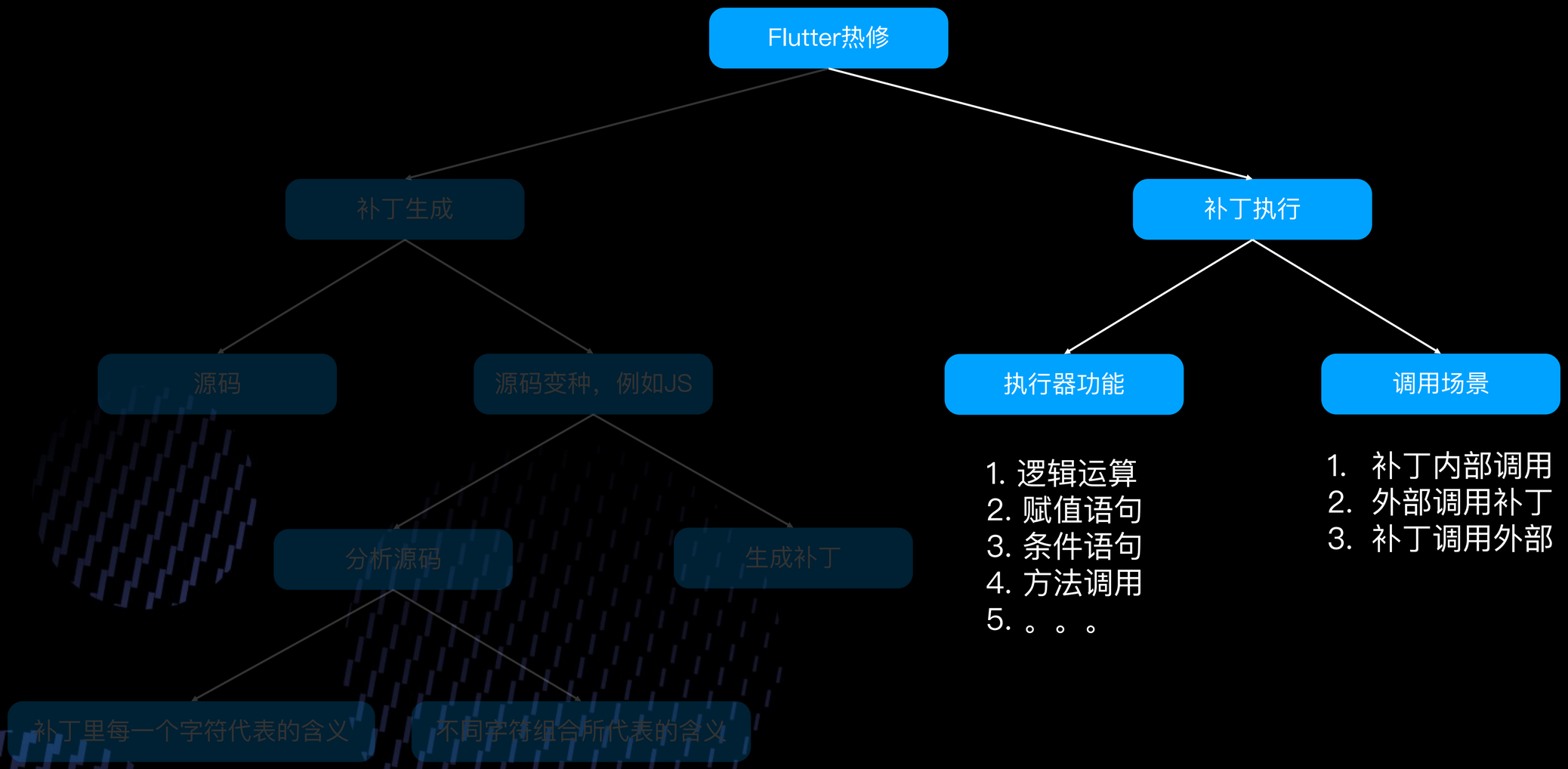
Assert相关、注释相关、声明定义类(mixin等)、import/part等包引入相关

每一种语法都是独一无二，不能用其他语法代替吗?

- Switch–case vs if–else等条件语句
- while、do–while、for、for–in、for–each等
- async/await vs future

130 → 40

执行补丁–调用场景



调用场景–外部调用补丁

运行时进行方法交换(JSPatch)	Dart 没有运行时~~~
业务代码里手动添加拦截代码	工作量太大，业务侵入性太强！
编译阶段插桩(AOP)	业界有较成熟的方案：AspectD

外部调用补丁–AspectD方法拦截

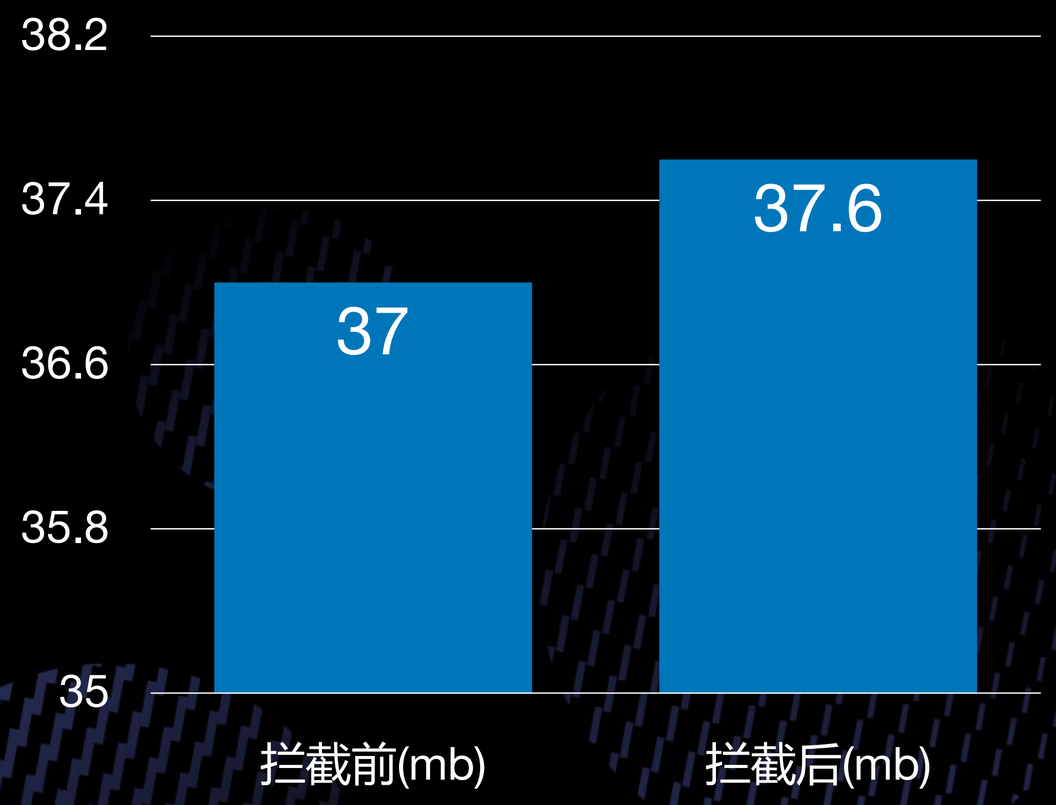
```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

```
/// aspectd_impl
@Aspect()
@pragma(“vm:entry-point”)
class FlutterHotFixAop {
 @pragma(“vm:entry-point”)
 FlutterHotFixAop();
 /// Hook example下的所有 .dart文件里的所有类、方法
 @Execute(“package:example/*.dart”, “.”, “-.+”,
          isRegex: true)
 dynamic hotFix(PointCut pointCut) {
  /*
   if (/*判断pointCut对应的方法需要热修*/) {
    return /*执行热修补丁*/;
   }
  */
   /// 调用原方法
   return pointCut.proceed();
 }
}
```

外部调用补丁–AspectD方法拦截

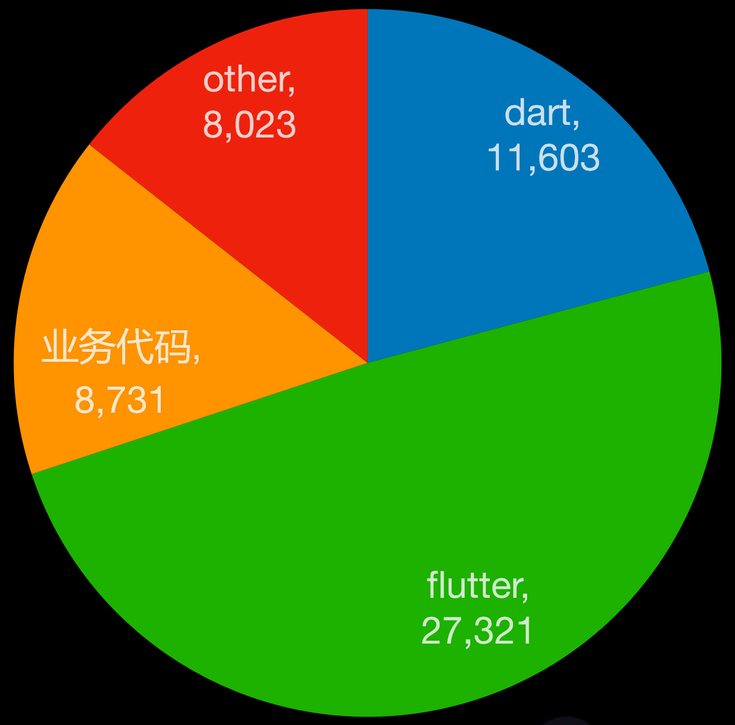
再加上APM、
无痕埋点....

拦截800个空方法，中间产物大小对比



6 * N = 6N (mb)

零售Flutter方法数量



Youzan.com

外部调用补丁–AspectD方法拦截

插桩之前

```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

插桩之后

```
/// example/a.dart
class SomeClass {
 dynamic someMethod() {
  /// 创建一个 PointCut实例，存储上下文信息
  final PointCut pointCut = PointCut(...params);
  return FlutterHotFixAop().hotFix(pointCut);
 }
 /// someMethod对应的副本
 dynamic someMethod_stub() {
  BugClass bugClass = BugClass();
  bugClass.methodHasBug(this);
 }
}
```

外部调用补丁–AspectD方法拦截

插桩之后

```
/// example/a.dart
class SomeClass {
  dynamic someMethod() {
    /// 创建一个 PointCut实例，存储上下文信息
    final PointCut pointCut = PointCut(...params);
    return FlutterHotFixAop().hotFix(pointCut);
  }
  /// someMethod对应的副本
  dynamic someMethod_stub() {
    BugClass bugClass = BugClass();
    bugClass.methodHasBug(this);
  }
}
```

存储原始实现的上下文信息

```
class  PointCut {
  dynamic proceed() {
    if (this.stub_key == “as_stub_0”) {
      return this.as_stub_0();
    }
    if (this.stub_key == “as_stub_1”) {
      return this.as_stub_1();
    }
    /// 穷举其它的插桩
  }
  dynamic as_stub_0() {
    return (this.target as SomeClass)
      .someMethod_stub();
  }
  dynamic as_stub_1() {
    /// 其它实现
  }
}
```

调用场景–补丁调用外部方法

```
/// example/bug.dart
class BugClass {
  dynamic methodHasBug(SomeClass someClassInstance) {
    someClassInstance.someMethod();
  }
}
```

从methodHasBug的调用里取到了该参数

该方法定义在SomeClass里，补丁里不包含SomeClass的任何信息

Allow using dart:mirrors #1150

Closed stevenroose opened this issue on 9 Jan 2016 · 61 comments

reflectable

补丁调用外部–反射

reflectable 3.0.4

Published 3 months ago Null safety

DART NATIVE JS FLUTTER ANDROID IOS LINUX MACOS WEB WINDOWS 79

Readme Changelog Example Installing Versions Scores

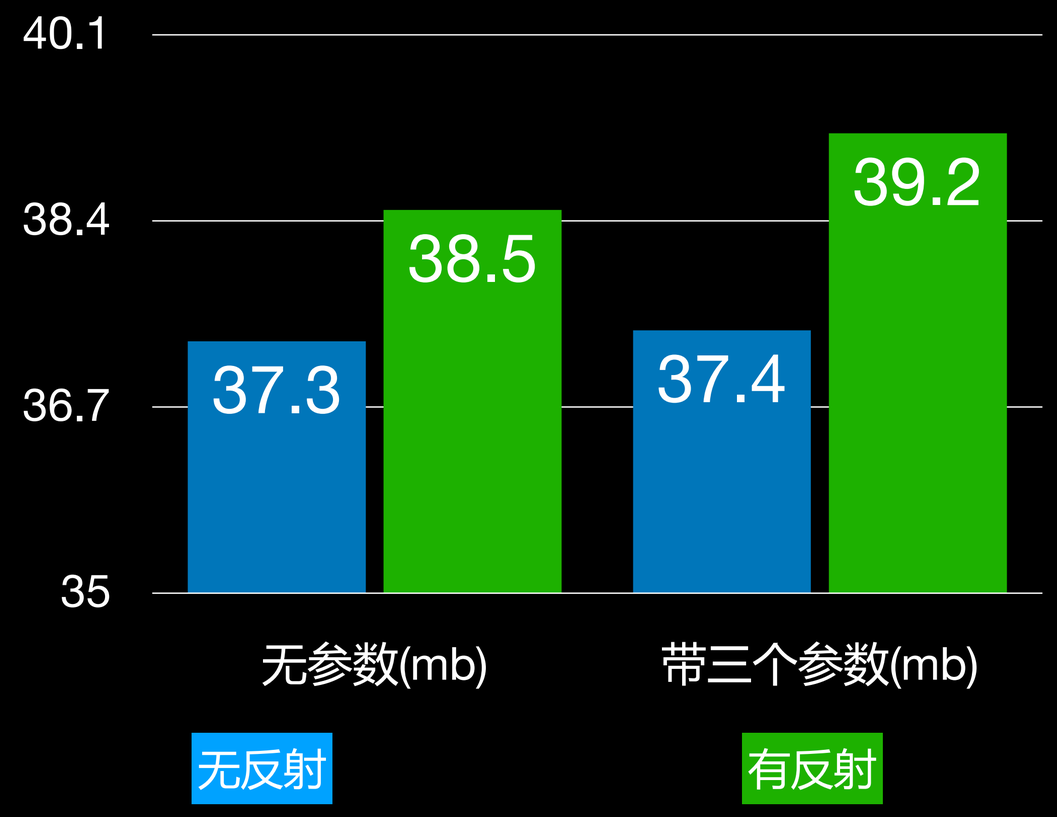
Reflectable

A listing of known limitations is given at the end of this page.

Introduction

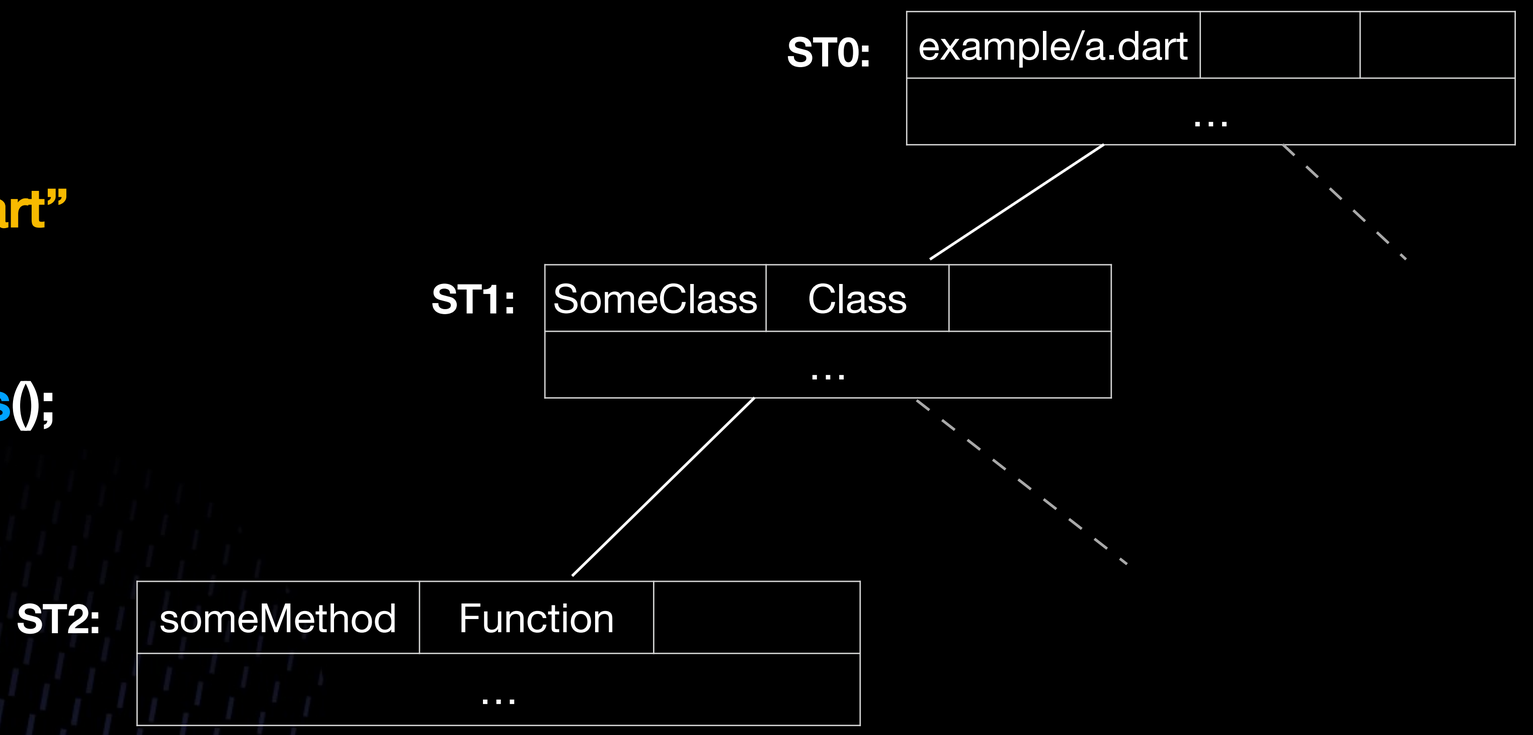
This package provides support for reflection which may be tailored to cover certain reflective features and omit others, thus reducing the resource requirements at run time.

800个方法，中间产物大小对比

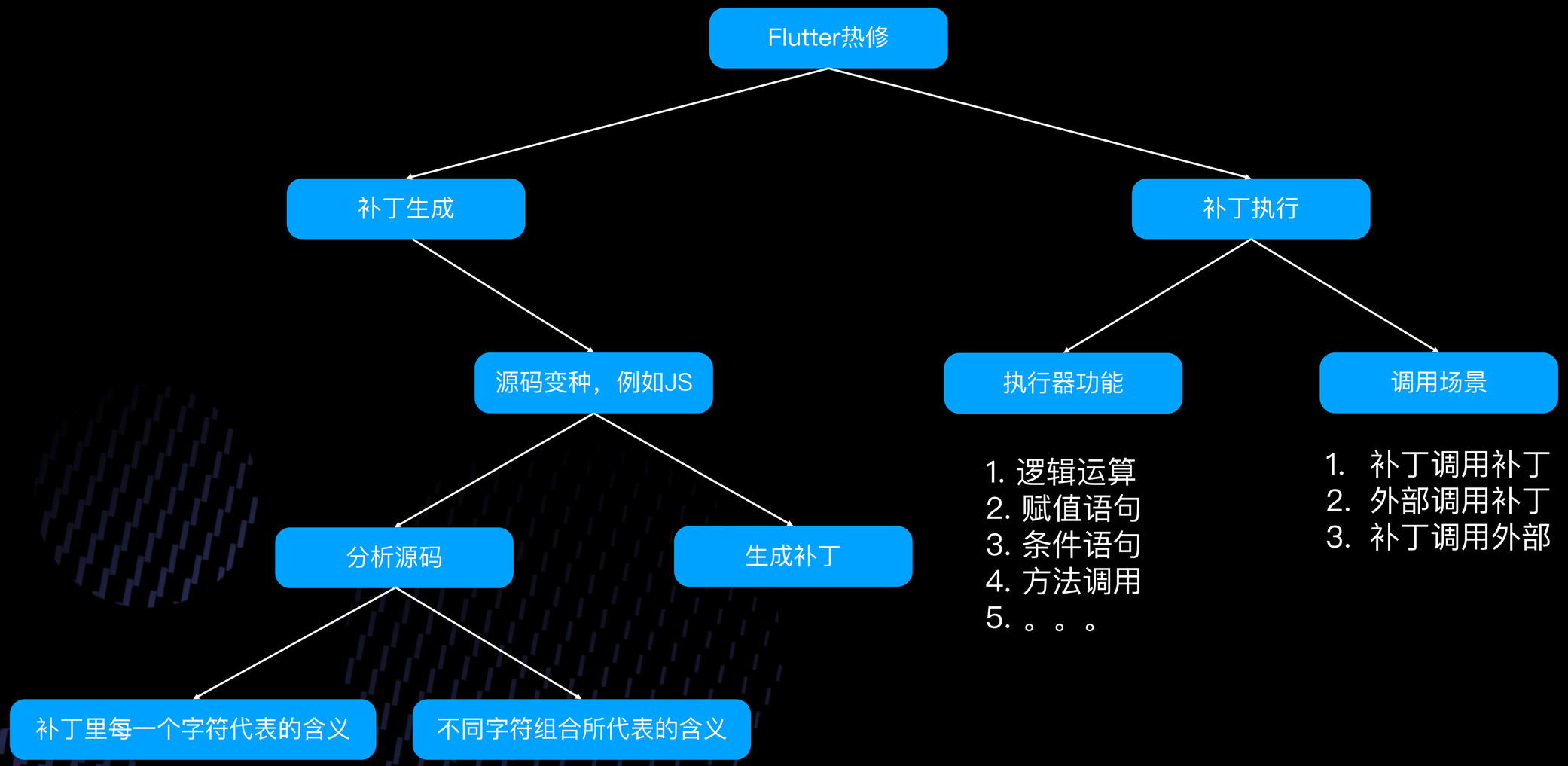


补丁调用外部方法–符号表

```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```



Flutter热修



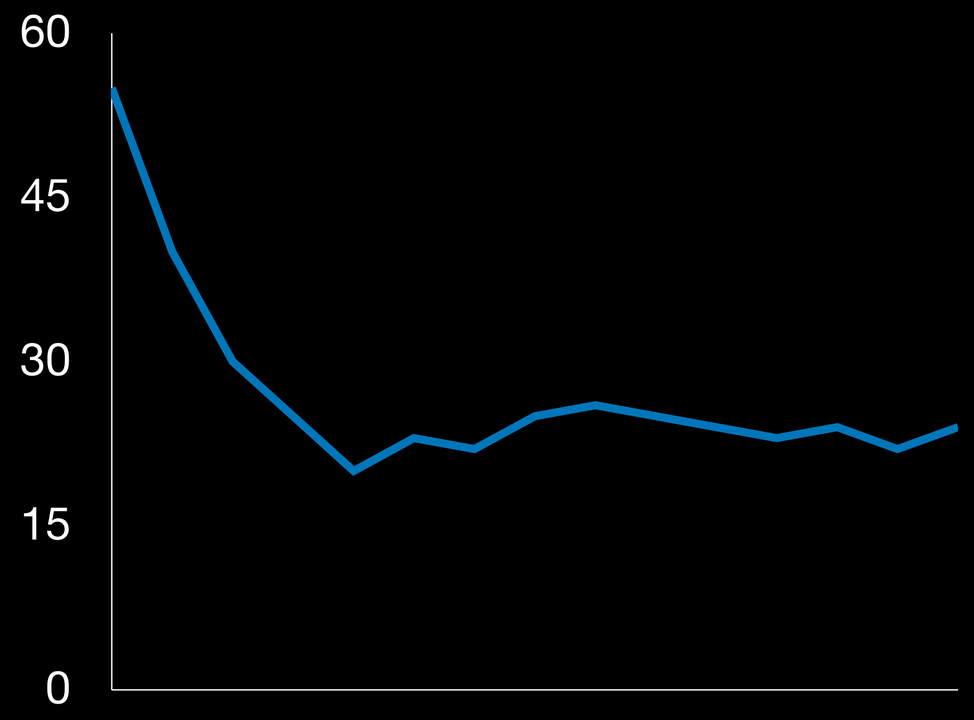
Flutter热修–踩坑

Crash

```
The following StackOverflowError was thrown while handling a g
Stack Overflow

When the exception was thrown, this was the stack
#0      new _HashVMBase (dart:collection-patch/comp
#1      new __InternalLinkedHashMap._Hash
#2      new __InternalLinkedHa
#3      new __InternalLinkedHashM
#4      new __InternalLinkedHash
#5      new _InternalLinkedHash
#6      new Map._ ._literal (dart:core-patch
```

FPS



Youzan.com

Flutter热修踩坑–Crash

```
/// example/a.dart
class SomeClass {
 dynamic someMethod() {
  /// 创建一个 PointCut实例，存储上下文信息
  final PointCut pointCut = PointCut(...params);
  return FlutterHotFixAop().hotFix(pointCut);
 }
 /// someMethod对应的副本
 dynamic someMethod_stub() {
  BugClass bugClass = BugClass();
  bugClass.methodHasBug(this);
 }
}
```

U Call, I Create.

Flutter热修踩坑–卡顿

```
class _PointCut {
  dynamic proceed() {
    if (this.stub_key == “as_stub_0”) {
      return this.as_stub_0();
    }
    if (this.stub_key == “as_stub_1”) {
      return this.as_stub_1();
    }
    /// 穷举其它的插桩
  }
  dynamic as_stub_0() {
    return (this.target as SomeClass)
      .someMethod_stub();
  }
  dynamic as_stub_1() {
    /// 其它实现
  }
}
```

时间复杂度:O(n)

有8700+个方法要拦截

Flutter热修踩坑–AOP优化

```
/// example/a.dart
class SomeClass {
 dynamic someMethod() {
  /// 创建一个 PointCut实例，存储上下文信息
  final PointCut pointCut = PointCut(...params);
  return FlutterHotFixAop().hotFix(pointCut);
 }
 /// someMethod对应的副本
 dynamic someMethod_stub() {
  BugClass bugClass = BugClass();
  bugClass.methodHasBug(this);
 }
}
```

```
class  PointCut {
 dynamic proceed() {
  if (this.stub_key == “as_stub_0”) {
   return this.as_stub_0();
  }
  if (this.stub_key == “as_stub_1”) {
   return this.as_stub_1();
  }
  /// 穷举其它的插桩
 }
 dynamic as_stub_0() {
  return (this.target as SomeClass)
   .someMethod_stub();
 }
 dynamic as_stub_1() {
  /// 其它实现
 }
}
```

Flutter热修踩坑–AOP优化

```
/// example/a.dart
class SomeClass {
  dynamic someMethod() {
    if (/*判断该方法需要热修*/) {
      return /*执行热修补丁*/;
    }
    BugClass bugClass = BugClass();
    bugClass.methodHasBug(this);
  }
}
```

如果想调用原方法怎么办?

Flutter热修踩坑–AOP优化

如果想调用原方法怎么办?

给方法添加一个「namedParameter」参数也不是啥难事~~~

```
void someFunction(String positionalParameter, {String? namedParameter}) {}
```

Flutter热修踩坑–AOP优化

```
/// example/a.dart
import “package:example/bug.dart”
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

```
library from “example/a.dart”
class SomeClass {
 dynamic someMethod({core::bool* callOriginImpl = true}) -> void {
    if (!callOriginImpl && /*判断该方法需要热修*/) {
      return /*执行热修补丁*/;
    }
    BugClass bugClass = BugClass();
    bugClass.methodHasBug(this);
 }
}
```

Flutter热修踩坑–AOP优化

```
/// example/a.dart
import “package:example/bug.dart
class SomeClass {
 dynamic someMethod() {
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

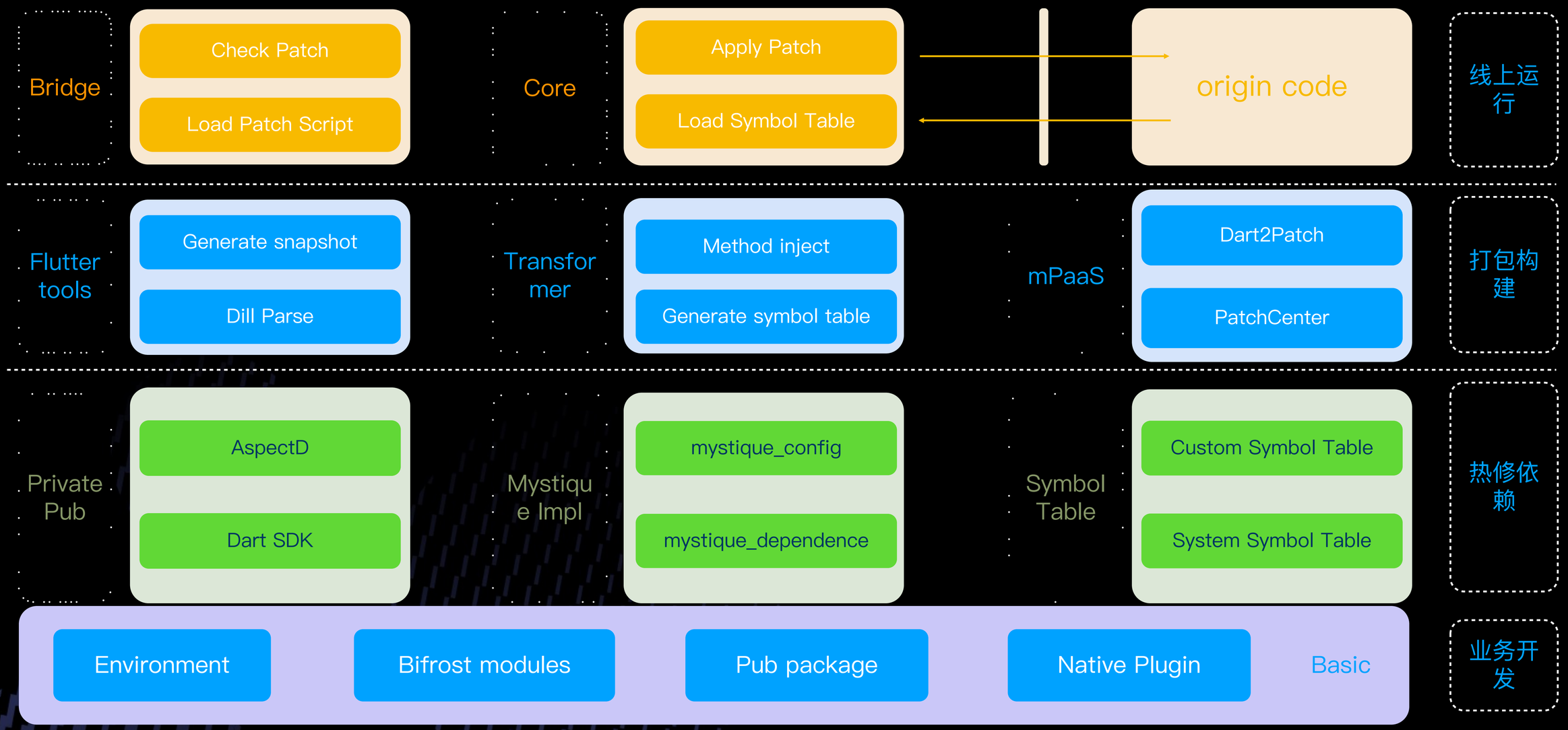
```
library from “example/a.dart”
class SomeClass {
 dynamic someMethod({core.bool* callOriginImpl = true}) -> void {
   if (callOriginImpl && /*判断该方法需要热修*/) {
     return /*执行热修补丁*/;
   }
   BugClass bugClass = BugClass();
   bugClass.methodHasBug(this);
 }
}
```

- 自定义Transformer，直接在业务方法体内增加插桩代码
- 优化包体积
- 提高执行效率，避免卡顿等性能影响
- 目标方法增加一个默认值为TRUE的namedParameter
- 支持原方法调用

Flutter热修踩坑–其他注意事项

- Flutter升级
  - 构建流程变了，之前的transformer可能都不再生效
  - kernel里的AST发生了变化（Flutter2.2.3/Dart2.13.4 vs Flutter2.5.0/Dart2.14.0）
  - 2.5之后自定义注解在AOT模式下失效，可替换成pragma
- 耐心与细心

热修架构



YOUZAN power