Dart语法补充---external关键字

zyc_在路上

一、external关键字详解

1、概述

说道抽象类abstract，就不得不说一下external关键字，external关键字估计用到人很少，在看源码的时候经常可以看到，如下：

```
class Object {
  const Object();
  external bool operator ==(other);
  external int get hashCode;
  external String toString();
  @pragma("vm:entry-point")
  external dynamic noSuchMethod(Invocation invocation);
  external Type get runtimeType;
}
```

- 可以看到Object类里有很多方法都是用external声明，并且这些方法没有具体实现；
- 但我们看到class不是abstract class，为什么方法可以不用实现呢？这就是external的作用。

2、说明

external 外面的，外部的

external只声明方法，声明的方法需要由外部去实现，通常是由底层sdk

根据不同平台(vm、web等)实现；若外部没实现，则会返回null；

3、作用

- external修饰的方法具有一种实现方法声明和实现分离的特性。

  关键在于它能实现声明和实现分离，这样就能复用同一套对外API的声明，然后对应不同平台的多套实现；这样不管是dart for web 还是dart for vm，对于上层开发而言都是同一套API；
- external声明的方法由底层sdk根据不同平台实现，class不用声明为abstract class，所以class可直接实例化；

4、external声明方法实现

```
@patch
class 类名 {
  ...
  @patch
  external声明的方法名
  ...
}
```

*external*声明的方法，通过*@patch*注解实现，结构如上；

比如Object里各种external声明方法的实现如下：

patch 补丁，小块

```dart
@patch
class Object {
  ...
  @patch
  bool operator ==(Object other) native "Object_equals";

  static final _hashCodeRnd = new Random();

  static int _objectHashCode(obj) {
    var result = _getHash(obj);
    if (result == 0) {
      // We want the hash to be a Smi value greater than 0.
      result = _hashCodeRnd.nextInt(0x40000000);
      do {
        result = _hashCodeRnd.nextInt(0x40000000);
      } while (result == 0);
      _setHash(obj, result);
    }
    return result;
  }

  @patch
  int get hashCode => _objectHashCode(this);


  @patch
  String toString() native "Object_toString";

  @patch
  @pragma("vm:exact-result-type", "dart:core#_Type")
  Type get runtimeType native "Object_runtimeType";
  ...
}
```

二、如何找到flutter external声明方法的实现

1、external声明方法实现文件路径

移动端external声明方法实现在vm目录下：

```
flutter sdk目录/bin/cache/dart-sdk/lib/_internal/vm/lib
```

web端external声明方法实现在js_runtime目录下：

```
flutter sdk目录/bin/cache/dart-sdk/lib/_internal/js_runtime/lib
```

2、external声明方法实现文件命名

external方法的实现文件一般命名为xxx_patch.dart，如在vm/lib目录下，可以看到各种xxx_patch.dart文件：

3、external声明方法实现文件查找

可以在终端通过grep搜索命令找到对应类里external方法实现的xxx_patch.dart文件：

```
grep -n "class 类名" -r ./* --color=auto
```

以查找*Object*类里*external*方法的实现为例：

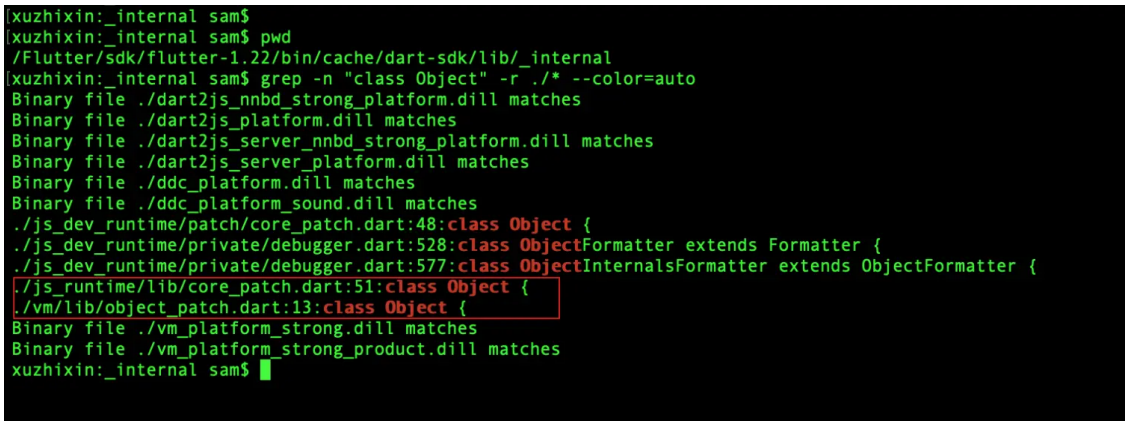
1、Object类定义如下：

```
class Object {
  const Object();
  external bool operator ==(other);
  external int get hashCode;
  external String toString();
  @pragma("vm:entry-point")
  external dynamic noSuchMethod(Invocation invocation);
  external Type get runtimeType;
}
```

可以看到*Object*类里有很多方法都是用*external*声明

2、在flutter sdk目录/bin/cache/dart-sdk/lib/_internal目录下，执行查找class Object命令：

```
grep -n "class Object" -r ./* --color=auto
```



由此可知：*web端Object实现文件是./js_runtime/lib/core_patch.dart*

移动端*Object*实现文件是*./vm/lib/object_patch.dart*

打开web端Object实现文件./js_runtime/lib/core_patch.dart，如下：

```
// Patch for Object implementation.
@patch
class Object {
  @patch
  bool operator ==(Object other) => identical(this, other);

  @patch
  int get hashCode => Primitives.objectHashCode(this);

  @patch
  String toString() =>
Primitives.objectToHumanReadableString(this);

  @patch
  dynamic noSuchMethod(Invocation invocation) {
```

```
    throw new NoSuchMethodError(this,
invocation.memberName,
        invocation.positionalArguments,
invocation.namedArguments);
  }

  @patch
  Type get runtimeType => getRuntimeType(this);
```

打开移动端Object实现文件./vm/lib/object_patch.dart，如下：

```
// part of "core_patch.dart";

@pragma("vm:exact-result-type", "dart:core#_Smi")
int _getHash(obj) native "Object_getHash";
void _setHash(obj, hash) native "Object_setHash";

@patch
@pragma("vm:entry-point")
class Object {
  // The VM has its own implementation of equals.
  @patch
  @pragma("vm:exact-result-type", bool)
  @pragma("vm:prefer-inline")
  bool operator ==(Object other) native "Object_equals";

  // Helpers used to implement hashCode. If a hashCode is
used, we remember it
  // in a weak table in the VM (32 bit) or in the header of
the object (64
  // bit). A new hashCode value is calculated using a
random number generator.
  static final _hashCodeRnd = new Random();
```

```
  static int _objectHashCode(obj) {
    var result = _getHash(obj);
    if (result == 0) {
      // We want the hash to be a Smi value greater than 0.
      result = _hashCodeRnd.nextInt(0x40000000);
      do {
        result = _hashCodeRnd.nextInt(0x40000000);
      } while (result == 0);
      _setHash(obj, result);
    }
    return result;
  }

  @patch
  int get hashCode => _objectHashCode(this);
  int get _identityHashCode => _objectHashCode(this);

  @patch
  String toString() native "Object_toString";
  // A statically dispatched version of Object.toString.
  static String _toString(obj) native "Object_toString";

  @patch
  @pragma("vm:entry-point", "call")
  dynamic noSuchMethod(Invocation invocation) {
    // TODO(regis): Remove temp constructor identifier
'withInvocation'.
    throw new NoSuchMethodError.withInvocation(this,
invocation);
  }

  @patch
  @pragma("vm:exact-result-type", "dart:core#_Type")
```

```
  Type get runtimeType native "Object_runtimeType";

  @pragma("vm:entry-point", "call")
  @pragma("vm:exact-result-type", bool)
  static bool _haveSameRuntimeType(a, b) native
"Object_haveSameRuntimeType";

  // Call this function instead of inlining instanceof,
thus collecting
  // type feedback and reducing code size of unoptimized
code.
  @pragma("vm:entry-point", "call")
  bool _instanceOf(instantiatorTypeArguments,
functionTypeArguments, type)
      native "Object_instanceOf";

  // Group of functions for implementing fast simple
instance of.
  @pragma("vm:entry-point", "call")
  bool _simpleInstanceOf(type) native
"Object_simpleInstanceOf";
  @pragma("vm:entry-point", "call")
  bool _simpleInstanceOfTrue(type) => true;
  @pragma("vm:entry-point", "call")
  bool _simpleInstanceOfFalse(type) => false;
}
```

可以看到 *Object* 里各种 *external* 声明方法对应的 *@patch* 注解实现方法