Dart语言异常和函数式编程

上期遗留

Exception(异常)

Dart中提供了Exception和Error两种子类型，Dart可以抛出任何非null对象为异常，不仅仅实现了Exception 或者Error

1. throw一个Exception,Error,或者任意对象

```
throw new FormatException('Expected at least 1section');
```

```
throw new UnimplementedError();
```

```
throw 'Out of llamas!'
```

2. Catch

```
try {
breedMoreLlamas();
} on OutOfLlamasException {
buyMoreLlamas();
}
```

3. on 捕获具体的Exception类型

```
try {
breedMoreLlamas();
} on OutOfLlamasException {
// A specific exception
buyMoreLlamas();
} on Exception catch (e) {
// Anything else that is an exception
```

```
print('Unknown exception: $e');
} catch (e) {
// No specified type, handles all
print('Something really unknown: $e');
}
```

4. Catch和On可以配合使用，Catch中可以加入第二个参数获取调用栈信息

```
...
} on Exception catch (e) {
print('Exception details:\n $e');
} catch (e, s) {
print('Exception details:\n $e');
print('Stack trace:\n $s');
}
```

5. rethrow关键字可以把捕获的异常给重新抛出来

```
final foo = '';
void misbehave() {
  try {
    foo = "You can't change a final variable's value.";
  } catch (e) {
    print('misbehave() partially handled ${e.runtimeType}.');
    rethrow; // Allow callers to see the exception.
  }
}

void main() {
  try {
    misbehave();
  } catch (e) {
    print('main() finished handling ${e.runtimeType}.');
  }
}
```

6. Finally 确保最后语句执行

```
try {
breedMoreLlamas();
```

```
} finally {
// Always clean up, even if an exception is thrown.
cleanLlamaStalls();
}
```

浅谈函数式编程

函数式编程形式

1. 函数式编程是一种“编程范式”，也就是如何编写程序的一种方法论。它属于“结构化编程”的一种，他的主要思想就是把运算过程尽量写成一系列嵌套的函数调用。

```
(1 + 2) * 3 - 4
```

常规写法：

```
var a = 1 + 2
var b = a * 3
var c = b - 4
```

函数式编程要求使用函数，把运算过程定义成不同函数

```
var result = subtract(multiply(add(1,2), 3), 4)
```

这就是函数式编程

函数式编程特点

i. 函数是第一等公民，函数和其它数据类型一样，处于平等地位，可以赋值给其它变量，也可以作为参数传入，或者作为别的函数返回值

```
var print = function(i) {console.log(i)} [1,2,3].forEach(print)
```

ii. 只用”表达式“，不用”语句"

"表达式"（expression）是一个单纯的运算过程，总是有返回值；"语句"（statement）

是执行某种操作，没有返回值。函数式编程要求，只使用表达式，不使用语句。也就是说，每一步都是单纯的运算，而且都有返回值。

iii. 没有“副作用”
所谓"副作用"（side effect），指的是函数内部与外部互动（最典型的情况，就是修改全局变量的值），产生运算以外的其他结果。函数式编程强调没有"副作用"，意味着函数要保持独立，所有功能就是返回一个新的值，没有其他行为，尤其是不得修改外部变量的值。

iv. 引用透明

引用透明（Referential transparency），指的是函数的运行不依赖于外部变量或"状态"，只依赖于输入的参数，任何时候只要参数相同，引用函数所得到的返回值总是相同的。

函数式编程意义

1. 代码简洁，开发快速
2. 接近于自然语言，易于理解
```
subtract(multiply(add(1,2), 3), 4) // 换一种写法 add(1,2).multiply(3).subtract(4)
```
3. 更方便的代码管理 函数式编程不依赖、也不会改变外界的状态，只要给定输入参数，返回的结果必定相同。因此，每一个函数都可以被看做独立单元，很有利于进行单元测试（unit testing）和除错（debugging），以及模块化组合

4. 易于“并发编程”

函数式编程不需要考虑"死锁"（deadlock），因为它不修改变量，所以根本不存在"锁"线程的问题。不必担心一个线程的数据，被另一个线程修改，所以可以很放心地把工作分摊到多个线程，部署"并发编程"（concurrency）。

```
var s1 = Op1();
var s2 = Op2();
var s3 = concat(s1, s2);
```

5. 代码热升级

函数式编程没有副作用，只要保证接口不变，内部实现是与外部无关的。所以，可以在运行状态下直接升级代码，不需要重启，也不需要停机。

Dart中函数式编程的例子