**1.1. List和Map的泛型**

List使用时的泛型写法:

// 创建List的方式

var names1 = ['why', 'kobe', 'james', 111];

print(names1.runtimeType); // List<Object>

// 限制类型

var names2 = <String>['why', 'kobe', 'james', 111]; // 最后一个报错

List<String> names3 = ['why', 'kobe', 'james', 111]; // 最后一个报错

Map使用时的泛型写法:

// 创建Map的方式

var infos1 = {1: 'one', 'name': 'why', 'age': 18};

print(infos1.runtimeType); // \_InternalLinkedHashMap<Object, Object>

// 对类型进行显示

Map<String, String> infos2 = {'name': 'why', 'age': 18}; // 18不能放在value中

var infos3 = <String, String>{'name': 'why', 'age': 18}; // 18不能放在value中

**1.2. 类定义的泛型**

如果我们需要定义一个类, 用于存储位置信息Location, 但是并不确定使用者希望使用的是int类型,还是double类型, 甚至是一个字符串, 这个时候如何定义呢?

* 一种方案是使用Object类型, 但是在之后使用时, 非常不方便
* 另一种方案就是使用泛型.

Location类的定义: Object方式

main(List<String> args) {

Location l1 = Location(10, 20);

print(l1.x.runtimeType); // Object

}

class Location {

Object x;

Object y;

Location(this.x, this.y);

}

Location类的定义: 泛型方式

main(List<String> args) {

Location l2 = Location<int>(10, 20);

print(l2.x.runtimeType); // int

Location l3 = Location<String>('aaa', 'bbb');

print(l3.x.runtimeType); // String

}

}

class Location<T> {

T x;

T y;

Location(this.x, this.y);

}

如果我们希望类型只能是num类型, 怎么做呢?

main(List<String> args) {

Location l2 = Location<int>(10, 20);

print(l2.x.runtimeType);

// 错误的写法, 类型必须继承自num

Location l3 = Location<String>('aaa', 'bbb');

print(l3.x.runtimeType);

}

class Location<T extends num> {

T x;

T y;

Location(this.x, this.y);

}

**1.3. 泛型方法的定义**

最初，Dart仅仅在类中支持泛型。后来一种称为泛型方法的新语法允许在方法和函数中使用类型参数。

main(List<String> args) {

var names = ['why', 'kobe'];

var first = getFirst(names);

print('$first ${first.runtimeType}'); // why String

}

T getFirst<T>(List<T> ts) {

return ts[0];

}