

3.2 Designing Specification 设计规约

一级标题, 二级标题, 三级标题, 注, 拓展

<http://web.mit.edu/6.031/www/sp20/classes/07-designing-specs/>

<http://web.mit.edu/6.031/www/sp20/classes/08-immutability/>

目录

Programming for communication (程序是为了交流)	1
3.1.1 Definition	1
拓展: Specifications of mutating methods	2
3.1.2 types (具体形式)	2
3.1.3 Reasons of use Designing Specification	3
3.1.4 Testing and verifying specifications	3
3.1.5 Classifying specifications (规约的分类)	4
3.1.6 Comparing specifications	4
拓展: Diagramming specifications	5
3.1.7 Quality of a specification	5
3.1.8 Behavioral equivalence (行为等价性)	7

Programming for communication (程序是为了交流)

1. Communicating with the computer.

First persuading the compiler that your program is sensible – syntactically correct and type-correct. Then getting the logic right so that it gives the right results at runtime. (代码中蕴含着设计决策, 代码编写规范是编程人员与计算机交流)

2. Communicating with other people.

Making the program easy to understand, so that when somebody has to fix it, improve it, or adapt it in the future, they can do so. (注释形式的设计决策: 是编程人员与其他人交流)

注: 程序规约属于第二种, Communicating with other people.

3.1.1 Definition

Abstractly speaking, a specification of a method has several parts:

1. a relationship of this method with requires and effects.
2. a requires clause (Precondition), describing restrictions on the parameters.
3. an effects clause (Postcondition), describing the return value, exceptions, and other

effects of the method.

Precondition (前置条件)

The precondition is an obligation on the client (i.e., the caller of the method). It's a condition over the state in which the method is invoked.

前置条件：对客户端的约束，在使用方法时必须满足的条件

Postcondition (后置条件)

The postcondition is an obligation on the implementer of the method.

后置条件：对开发者的约束，方法结束时必须满足的条件

Relationship between precondition and postcondition

If the precondition holds when the method is called, then the postcondition must hold when the method completes. (前置条件满足，则后置条件必须满足)。

If the precondition does not hold when the method is called, the implementation is not bound by the postcondition. (前置条件不满足，可以做任何事情)

拓展：Specifications of mutating methods

mutating methods:

某方法是 mutating methods，如果参数的值在该方法中被修改。

Specifications for mutating methods

If the effects do not explicitly say that an input can be mutated, then we assume mutation of the input is implicitly disallowed. (除非在后置条件里声明过，否则方法内部不应该改变输入参数) (意思大概就是：尽量避免产生该种类型的方法)

Reason of not use mutating methods

这种不确定性会在某个值的多个引用中产生糟糕的后果。详见 3.1 中的例子。

Solution

不可变的数据类型的值作为参数传递，例如基本数据类型。

3.1.2 types (具体形式)

一般形式:

```
static int find(int[] arr, int val)
  requires: val occurs exactly once in arr
  effects:  returns index i such that arr[i] = val
```

In java:

1. 必须在方法的开头使用/** */形式
2. 不需要注明函数类型，参数类型，返回值类型，下面已有，不需要重复。

```

/**
 * Find a value in an array.
 * @param arr array to search, requires that val occurs exactly once
 *         in arr
 * @param val value to search for
 * @return index i such that arr[i] = val
 */
static int find(int[] arr, int val)

```

3.1.3 Reasons of use Designing Specification

the linchpin of teamwork (是团队合作的基石)

The specification acts as a contract (契约): the implementer is responsible for meeting the contract, and a client that uses the method can rely on the contract.

Precise specifications in the code let you apportion blame to code fragments, and can spare you the agony of puzzling over where a fix should go. (精确的规约, 有助于区分责任)

Decouple implementor and clients (解耦编写者与调用者)

Designing Specification acts as a firewall (防火墙) between client and implementor, allowing the code of the unit and the code of a client to be changed independently, so long as the changes respect the specification. (使得二者可以在遵守规约的前提下, 独立的改变内部代码实现方式)

Designing Specification make code faster. (编写者不需要写代码确保输入的正确性, 调用者的责任)

3.1.4 Testing and verifying specifications

Black-box testing 黑盒测试

Black-box testing: to check if the tested program follow the specified specification in an implementation-independent way.

Your test cases should not count on any concrete implemented behavior.

- **An example specification:**

```

static int find(int[] arr, int val)
    requires: val occurs in arr
    effects: returns index i such that arr[i] = val

```

- **The test case:**

只需要保证的结果为7

```

int[] array = new int[] { 7, 7, 7 };
assertEquals(0, find(array, 7)); // bad test case: violates the spec
assertEquals(7, array[find(array, 7)]); // correct

```

Ex:

8 84

This test case has assumed a specific implementation that find always returns the lowest index.

3.1.5 Classifying specifications (规约的分类)

Deterministic vs. not determined specs (确定的规约与不确定的规约)

Deterministic:

when presented with a state satisfying the precondition, the outcome is completely determined. (当给定一个满足 precondition 的输入, 其输出是唯一的, 明确的)

Under-deterministic (欠定的规约)

specification allows multiple valid outputs for the same input. (给定一个输入, 不同的实现方式给出的结果是不同的, 但对一种特定的实现方式, 结果是相同的)

欠定: 一旦确定, 输出不变

Nondeterministic:

sometimes behaves one way and sometimes another, even if called in the same program with the same inputs (e.g., depending on random or timing) (在该种实现方式下, 同一个函数输入同一个参数执行多次结果也可能不同, 在确定的实现方式下, 结果可能是不同的)

注: Under-deterministic (欠定的规约) and Nondeterministic belong to not determined

注: 关于这一部分, 无法区分 Under-deterministic 意为与 determined 进行对立的那一部分, 亦或者是欠定的那一部分。详见: <http://web.mit.edu/6.031/www/sp20/classes/07-designing-specs/> 中关于这一部分的描述。

Declarative vs. operational specs

Operational specifications (操作时规约)

Operational specifications give a series of steps that the method performs; pseudocode descriptions are operational. (操作式规约, 在规约中展示某些具体实现细节)

Declarative specs (声明式规约)

Declarative specifications don't give details of intermediate steps. Instead, they just give properties of the final outcome, and how it's related to the initial state. 声明式规约: 没有内部实现的描述, 只有“初-终”状态)

注: Declarative specs (声明式规约) 更好, 因为对于操作式规约, 调用者会根据规约的内部实现内容进行调用, 在某种程度上会不利于编程人员进行编写。

3.1.6 Comparing specifications

How to compare:

specification S2 is stronger than or equal to a specification S1 (规约的强度 $S2 \geq S1$)

IF:

S2's precondition is weaker than or equal to S1's 前置条件更弱

S2's postcondition is stronger than or equal to S1's, for the states that satisfy S1's precondition. 后置条件更强

注: When a specification is strengthened,

Fewer implementations satisfy it **and** More clients can use it.

拓展: Diagramming specifications

Each point in this space represents a method implementation.

Each circle in this space represents spec.

When S2 is stronger than S1, it defines a smaller region in this diagram. 更强的规约, 表达为更小的区域

3.1.7 Quality of a specification

coherent (内聚的):

内聚度标志一个模块内部各成分彼此结合的紧密程度。


对于一个方法中, 必须使得在其内完成的任务具有强大的相关性, 提高其内聚度。可以将之拆分为多个方法, 增强其内聚度。

尽量使得一个方法完成一个任务。

Informative (信息丰富的)

不可以让调用者产生理解的歧义。

Ex:



信息丰富的

```
static V put (Map<K,V> map, K key, V val)
  requires: val may be null, and map may contain null values
  effects:  inserts (key, val) into the mapping,
            overriding any existing mapping for key, and
            returns old value for key, unless none,
            in which case it returns null
```

- **If null is returned, you can't tell whether the key was not bound previously, or whether it was in fact bound to null.**
- **This is not a very good design, because the return value is useless unless you know for sure that you didn't insert null.**

Strong enough (足够强) 输出

对于需要处理的情况, 必须完全涵盖需要处理的内容, 包括返回值, 包括对于参数的赋值问题等等。

```
static File open(String filename)
    effects: opens a file named filename
```

■ **This is a bad specification.**

- It lacks important details: is the file opened for reading or writing? Does it already exist or is it created?
- **It's too strong, since there's no way it can guarantee to open a file.**
The process in which it runs may lack permission to open a file, or there might be some problem with the file system beyond the control of the program.

■ **Instead, the specification should say something much weaker: it attempts to open a file, and if it succeeds, the file has certain properties.**

- **太强的spec，在很多特殊情况下难以达到，给开发者增加了实现的难度（client当然非常高兴）。**

Weak enough（足够弱）输入

- We must use extra care when specifying the **special cases**, to make sure they don't undermine what would otherwise be a useful method.
- For example, there's no point throwing an exception for a bad argument but allowing arbitrary mutations, because a client won't be able to determine what mutations have actually been made.

```
static void addAll(List<T> list1, List<T> list2)
    effects: adds the elements of list2 to list1,
            unless it encounters a null element,
            at which point it throws a NullPointerException
```

没有充分阐明遇到null之后参数是否变化

Use abstract types（使用抽象类型）

Writing our specification with abstract types gives more freedom to both the client and the implementor. 在规约里使用抽象类型，可以给方法的实现体与客户端更大的自由度

Precondition or postcondition

不写 Precondition，就要在代码内部 check；若代价太大，在规约里加入 precondition，把责任交给 client

是否使用前置条件取决于：(1) check 的代价；(2) 方法的使用范围

– 如果只在类的内部使用该方法(private)，那么可以不使用前置条件，在使用 该方法的各个位置进行 check——责任交给内部 client；

– 如果在其他地方使用该方法(public)，那么必须要使用前置条件，若 client 端 不满足则方法抛出异常。

注：strong enough 说明的是输出的完备化，weak enough 说明的是输入的细致化

3.1.8 Behavioral equivalence (行：为等价性)

Definition of behavioral equivalence (行为等价性)

To determine behavioral equivalence, the question is whether we could substitute one implementation for the other. (两个函数是否具有行为等价性，取决于二者能否替换)

Determination in the eye of the client

判断两个方法是否就有相同的行为等价性，如果是，就具有行为等价性。