Java Semi-Formal

Specification Language

(JSF)

目录

1 JSF基本语法 3

1.1 JSF语法形式 3

1.2 基本表达式 3

1.2.1 量化布尔表达式 3

1.2.2其他关键字 3

1.2.3 逻辑运算符 4

1.2.4描述形式 5

1.2.5变量约束 5

2 JSF方法规格 5

2.1 REQUIRES ：前置条件 6

2.2 MODIFIES：副作用 6

2.3 EFFECTS：后置条件 7

3 多线程场景下的方法规格 7

3.1多线程共享前置规格 8

3.2 多线程共享后置规格 8

# 1 JSF基本语法

## 1.1 JSF语法形式

JSF规格的形式有如下两种：

**//@ <JSF>**

**//@ <JSF>**

**……**

**/\*\*@ <JSF>**

**@ <JSF>**

**……**

**@ <JSF> \*/**

在/\*\*@……\*/的注释中，@符号在规格的每一行的开头之前使用。

## 1.2 基本表达式

### 1.2.1 量化布尔表达式

使用关键字\*all*替代，以及\*exist*替代。用法如下：

* （\*all* T x; b）表示所有的T类型的x使得b成立。
* （\*all* T x; r; b）等价于（\*all* T x; r==>b），表示所有的r范围里的T类型的x，使得b成立。
* （\*exist* T x; b）表示存在T类型的x满足b。
* （\*exist* T x; r; b）等价于（\*exist* T x; r==>b），表示存在一个的r范围里的T类型的x，使得b成立。

**例1：**

**//@ REQUIRES:** **(\all int i, j;**

**//@   0 <= i < j < a.length;**

**//@   a[i] <= a[j]);**

**public int lookup(int elem) {…**

**表示所有在“0 <= i < j < a.length”范围里的、使得“a[i] <= a[j]”成立的整数i和j。**

### 1.2.2其他关键字

* （\sum T x; range a to b; exp）等价于(x)。
* \max c，\min c表达了给定可比较大小的容器c中的最大值和最小值。

**例2：**

**//@ EFFECTS: \result == (\max arr);**

**表示后置条件中结果返回的是数组arr中的最大值。**

* \*result* 表示其所在方法的返回值。
* \*old*(e) 表示在当前方法被调用刚开始时表达式e的值，即未被当前方法修改之前的e。只能用于后置条件中。

**例3：**

**//@ EFFECTS: v.size == \old(v).size + 1;**

**表示当前方法执行后，容器v相对于之前的规模变大了一个。**

* 使用常规单词表达含义，而尽量减少缩写、和一些自定义的标识符。例如使用*length*表达数组长度，*size*表达数据结构的大小，v.*contains*(e)：表示容器v中包含元素e。

**例4：**

**public void add(Queue q, elem)**

**//@REQUIRES: q != NULL;**

**//@MODIFIES: q**

**//@ EFFECTS: q.contains(elem);**

**表示方法add执行后，q中包含元素elem。**

### 1.2.3 逻辑运算符

* A==> B ：表示了A蕴含B（如果A为真则一定有B为真，则称A蕴含了B）
* &&：逻辑与（与面向对象编程语言逻辑保持一致）
* ||：逻辑或（与面向对象编程语言逻辑保持一致）
* ！：逻辑非

### 1.2.4比较运算符

Java中常用的算数比较运算符都支持，如==，<, >, <=, >=, !=，等。为了灵活性，支持连写范围检查，如“**0<=i<v.length”**.

### 1.2.5描述形式

JSF中的规格描述可以使用Java程序中的常用方法来帮助表达，而不对语法进行限制。

**例6：**

**对于数组arr[]来说，描述其长度的语法是：arr.length；而对于字符串str来说，描述其长度的语法是：str.length()。当使用JSF描述时，只要识别到“length”，就表明是在描述长度，即允许规格中出现类似于“arr.length()”或者“arr.length”的写法。**

JSF的逻辑表达式有基础的类型概念，包括布尔、整数、对象和容器等。

### 1.2.5变量约束

JSF建议在规格中尽量不要使用中间变量。

# 2 JSF方法规格

方法规格置于方法名之后：

**<访问修饰符> <返回值类型> <方法名>（<参数列表>）{**

**/\*\*@ REQUIRES:**

**@ MODIFIES:**

**@ EFFECTS:**

**@ \*/**

**<方法体>**

**}**

**例6：**

**public boolean contains(int elem) {**

**/\*\* @ EFFECTS: \result == (\exists int i; 0 <= i < arr.size; arr[i] == elem);**

**@ \*/**

**……}**

**public void remove(int elem) {**

**/\*\* @ REQUIRES: arr != {}; //{}表示空集**

**@** **EFFECTS: !arr.contains(elem);**

**@ EFFECTS: (\forall int e; e != elem; arr.contains(e) == \old(arr).contains(e));**

**@ \*/**

**……}**

## 2.1 REQUIRES ：前置条件

前置条件，执行前过程对输入或系统状态的约束要求。方法规格的前置条件和后置条件实质上就是对输入进行划分。通过划分，简化了方法的计算逻辑——在前置条件范围内的，是方法必须要处理的输入，对应的处理结果必须满足相应的约束；而在这范围之外的可以不予考虑。

假定输入到方法中的参数都是合法的，无需开发人员在前置条件中加以赘述。前置条件必须是布尔表达式。

## 2.2 MODIFIES：副作用

表示一个方法在执行过程会对哪些用户能够感知到的数据进行修改。所谓“能够感知到的修改” 指的是函数内部与外部互动，产生运算以外的其他结果，包括：对this的修改、对传入对象的修改、对环境对象的修改（如数据库、外接设备等）、对全局变量的修改。

如果没有修改的值，则此项可写为“None”，即此方法为纯方法。纯方法就是没有副作用的方法，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。

## 2.3 EFFECTS：后置条件

执行后返回结果或系统状态应该满足的要求，也就是过程在所有未被前置条件排出的输入下给出的结果。一种方法只有在遵守前提条件的情况下才能保留其后置条件。如果在不符合前提条件的情况下进行调用，则无法保证。

JSF对后置条件细化，针对不同的情况分为*normal\_behavior*和*exceptional\_behavior*(E)。*normal\_behavior*表示方法正常终止情况下的后置条件，可以省略不写。JSF允许在异常情况下使用*exceptional\_behavior*，来表示在某些情况下，方法执行效果为抛出类型为E的相应异常。E是Throwable的子类。

在后置条件描述中，每个@EFFECTS后跟着的都应该是一个可判定的布尔表达式，且应该只使用集合论和一阶逻辑谓词来表示的布尔表达式，从而有效描述方法的设计约束。虽然**也可以采用自然语言描述规格，但会不可避免的引入二义性**。规格不应该描述实现过程（即算法流程，通常表现为，IF condition THEN action等）。

# 3 多线程场景下的方法规格

多线程场景下，任何一个类都可能被多个线程共享。因此，除了功能视角的规格之外，还需要从线程共享控制视角来明确相应的规格。

共享对象方法的规格说明形式如下：

**<访问修饰符> <返回值类型> <方法名>（<参数列表>）{**

**/\*\* @ REQUIRES:**

**@ MODIFIES:**

**@ EFFECTS:**

**@ THREAD\_REQUIRES:**

**@ THREAD\_EFFECTS:**

**@ \*/**

**<方法体>**

**}**

## 3.1多线程共享前置规格

THREAD\_REQUIRES**描述是否要求线程主动对该方法的调用进行锁控制，使用**关键字*\locked(…)*表示。如果不写，则表明对使用者不做任何多线程同步控制要求。对于线程安全设计而言，不应对使用者提出此要求。*\locked(…)*中的参数可以是\this或者方法传入的对象变量，表示要求调用者对括号中的共享对象进行锁控制。

## 3.2 多线程共享后置规格

THREAD\_EFFECTS专门用来描述一个方法否对多线程并发执行进行了控制，使用关键词*\locked(x)*。如果不写，则表明该方法不对多线程并发执行进行任何控制。如果括号后面没有任何变量，则表明对整个方法进行同步控制；如果给定了变量，则表明该方法仅对相应变量的操作访问进行了同步控制。注意，\*locked(x)*中的x只能是方法输入参数中的对象，或者所处类的成员对象。对于线程安全设计而言，这部分一定要写。

# 4 类规格

类的规格说明应该尽可能继承已有类。类规格采用以下形式：

**<访问修饰符> class <类名>{**

**/\*\* @ OVERVIEW:**

**@ INHERIT: <父类名> | <方法列表>**

**@ <Local Definitions>**

**\*/**

**<构造方法>**

**<成员方法>**

**……**

**}**

## 4.1 OVERVIEW: 类的概述

阐明类的目标/整体轮廓。

## 4.2 INHERIT: 类间的继承

类间的继承是对已有类规格说明复用的一种灵活的手段，子类可以省略重复规格的书写，覆盖父类的方法或追加新的属性和方法。规格说明中的继承关系可以看作是宏定义语句，如果将类模式中继承语句全部扩展为实际语句, 便可以得到没有继承关系的规格说明。

类间继承列出的方法列表是在子类中未作改动的父类的方法，即可以直接复用的规格。可以不要求同学们列出此项。列出的好处是使JSF Tool将父类中可以直接复用的规格显示在子类规格的说明中，而无需查看此方法的父类规格。

在方法规格中不需要加关键字即可重写（又称覆盖）、追加新的属性和方法。JSF规格不限制也不检查规格是否符合Java语法。例如子类重写父类的方法，不检查子类的此方法是否与父类的具有相同的方法名、返回类型和参数表。需要注意的是，子类需要保持父类的方法规格仍然成立。

## 4.3 不变量

类不变量可以视作是限制实例中状态的条件。子类以及接口的实现都会继承不变量。如果不指定不变量，也可以明确地将这些规范在前后置条件中表明。但是，这意味着在类中添加方法时，要同时在前后置条件中添加这些规范，会比较麻烦。

**例8：**

**//@ INVARIANT: getScore() == A ==> Score > 85;**