1. Введение

1.1 Статический анализ кода

Определение 1.1.1. *Статический анализ кода - это анализ программного обеспечения, производимый без реального выполнения исследуемых программ.*

Статический анализ позволяет выявить многие виды ошибок еще до запуска программы, большинство из которых сложно искать и воспроизводить непосредственно во время работы приложения. В связи с этим активно развиваются различные инструменты, позволяющие статически доказывать отсутсвие в программах ошибок тех или иных видов.

привести примеры тулов в исторической последовательности

1.2 Неизменяемость в контексте объектно-ориентированного языка

В различных контекстах понятие неизменяемости может пониматься по-разному. В данной работе я рассматриваю несколько видов неизменяемости:

Определение 1.2.1. *Неизменяемый класс - класс, все представители которого являются неизменяемыми.*

Примером неизменяемого класса является, например, java.lang.String.

Определение 1.2.2. *Неизменяемый объект - объект, который не может быть изменен, при этом другие представители того же самого класса могут быть изменены.*

добавить картинку Если в какая-либо система позволяет выражать данное свойство объекта, будем говорить, что в данной системе есть поддержка *объектной неизменяемости.*

Определение 1.2.3. *реизменяемая ссылка - ссылка, которая не может быть использована для изменения объекта, на который она указывает (при этом объект может быть изменен через другую ссылку).*

добавить кратинку Если какая-либо система позволяет выражать данное свойство объекта, будем говорить, что в данной системе есть поддержка *ссылочной неизменяемости.*

Нужно заметить, что данные понятия не являются чем-то искусственным по отношению к языкам программирования. Приведем примеры использования данных понятий в языке программирования Java.

Например, в документации к классу org.joda.time.Period написано: "Неизменяемый временной период..."[1]. Таким образом, класс org.joda.time.Period является неизменяемым классом. *нужно ли приводить еще примеры*

[1] http://joda-time.sourceforge.net/apidocs/org/joda/time/Period.html

1.3 Обзор существующих решений

Рассмотрим, как пролема контроля изменяемости решается в различных объектно-ориентированных языках программирования.

1.3.1 C++

В языке C++ для выражения неизменяемости есть ключевое слово const.

В случае с нессылочными типами данных, если какая-либо переменная объявлена как const, то ее значение не может быть изменено после инициализации. Это означает, что в C++ есть объектная неизменяемость.

Листинг 1.1: Константая переменная

```
1  struct S
2  {
3      int val;
4  };
5
6
7          const S const_s;
8          const_s.val = 42;       // Error: const_s was declared as const
9      int i  = const_s.val;  // OK: field val is accessed for reading,
10                            // not for writing
11
12         const S non_const_s;
13         non_const_s.val = 42;  // OK: non_const_s was not declared as const
```

Для указателей и ссылкок значение модификатора const более сложное. Константным может быть сам указатель, значение, на которое он указывает или оба. Если какая-либо переменная объявлена как константый указатель, то ее значение не может быть изменено после инициализации. Если переменная объявлена как указатель на константный объект, то ее значение может быть изменено, но ее нельзя использовать для изменения объекта, на который она указывает. Таким образом, в C++ есть ссылочная неизменяемость. Нужно заметить, что не существует никакого способа сказать, что некий указатель указывает на неизменяемый объект. Все то же самое касается ссылок.

Листинг 1.2: Константный указатель

```
1  struct S
2  {
3      int val;
4  };
5
6  void Foo( S * ptr,
7            S const * ptrToConst,
```

```
8           S * const constPtr,
9           S const * const constPtrToConst )
10 {
11     ptr->val = 0;     // OK: modifies the "pointee" data
12     ptr  = NULL;      // OK: modifies the pointer
13
14     ptrToConst->val = 0; // Error: cannot modify the "pointee" data
15     ptrToConst  = NULL;  // OK: modifies the pointer
16
17     constPtr->val = 0; // OK: modifies the "pointee" data
18     constPtr  = NULL;  // Error: cannot modify the pointer
19
20     constPtrToConst->val = 0; // Error: cannot modify the "pointee" data
21     constPtrToConst  = NULL;  // Error: cannot modify the pointer
22 }
```

Теперь рассмотрим, что именно понимается под изменением какого-либо значения. В общем случае можно сказать, что если какое-то значение неизменяемо, то в ту часть памяти компьютера, где оно хранится, не может быть произведена запись. Методы, которые не изменяют значение объекта, на котором вызываются, могут быть помечены ключевым словом const. Тот факт, что метод действительно не изменяет объект, на котором вызывается, проверяется статически. Методы, помеченные как const могут быть вызваны как на константных, так и на неконстантных объектах. Методы, непомеченные как const, могут быть вызваны только на неконстантных объектах.

В даном подходе есть несколько недостатков. Первый из них связан с хранением в объекте указателей на другие объекты. Если некий объект является константным, то указатели, хранящиеся в нем в качетсве полей, будут константными, но при этом они могут быть использованы для изменения объекта, на который ссылаются. Рассмотрим пример:

Листинг 1.3: Пример изменения значения по указателю в константном методе

```
1  struct S
2  {
3      int val;
4      int *ptr;
5  };
6
7  void Foo(const S & s)
8  {
9      int i  = 42;
10     s.val  = i;  // Error: s is const, so val is a const int
11     s.ptr  = &i; // Error: s is const, so val is a const int
12     *s.ptr = i;  // OK: the data pointed to by ptr is always mutable
13 }
```

Несмотря на то, что s передается в метод Foo() как константый (что также делает константными всех его членов), объект, доступный через s.ptr можно изменять. Таким образом, в C++ нет поддержки глубокой неизменяемости.

добавить строгое определение глубокой неизменяемсти и картиночку

Также в C++ невозможно вернуть ссылку, чья изменяемость зависит от изменяемости this. Поэтому, например, во всех колелкциях STL содержатся по две перегруженные версии iterator и operator[], которые, фактически, делают одно и то же.

ссылка на источник

1.3.2 Java

В языке Java есть ключевое слово final, обозначающее, что значение соответствующего поля или переменной не может быть переприсвоено. Если все поля некоторого объекта объявлены как final, то можно говорить о том, что данный объект неизменяем. Действительно, после завершения конструктора в final поле всегда может находиться один и тот же объект, но сам этот объект может быть изменен. Таким образом, в Java нет поддержки глубокой неизменяемости.

Листинг 1.4: Ключевое слово final

```
1  public class MyClass {
2      public final int[] values;
3
4          public MyClass() {
5                  values = new int[10];
6          }
7
8  }
9
10     MyClass mc = new MyClass();
11     mc.values = new int[100]; // Error: field values was declared final
12     mc.values[2] = 4; // OK: values is declared final, but we can still
13                       // change object, referenced by this field.
```

Показательным является следующий пример: пусть есть некий класс, который содержит в себе ссылку на список объектов. Разработчик интерфейса этого класса хочет разрешить клиенту получать хранимый список, но не хочет, чтобы клиент мог модифицировать данный список. На Java код такого класса будет скорее всего выглядеть следующим образом:

Листинг 1.5: Неизменяемый список

```
1  public class ListContainer{
2      private final List<String> values = new ArrayList<String>();
3
4      public List<String> getValues() {
5          return Collections.unmodifiableList(values);
6      }
```

```
7
8 }
```

В данном случае, Collections.unmodifiableList(values) вернет обертку над исходным списком, у которой переопределены все изменяющие список методы так, что они бросают UnsupportedOperationException. Основным недостатком данного подхода является то, что ошибка будет обнаружена только во время выполнения программы. Ее локализация и исправление потребуют горадо больше усилий, чем если бы данная ошибка была выявлена на этапе компиляции.

Листинг 1.6: Использование неизменяемого списка

```
1         ListContainer container = new ListContainer();
2         List<String> containerValues = container.getValues();
3         int size = containerValues.size(); // OK: getting size is permitted
              for immutable list
4         containerValues.add("Hello!");      // Error: this code will be
              successfuly compiled, but
5                                             // will cause
                                                  UnsupportedOperationException on
                                                  runtime
```

Таким образом, в стандратной библиотеке Java неизменяемые коллеции реализованы просто как обертки над стандартыми интерфесами, у которых переипределены меняющие объект методы. Так как при вызове Collections.unmodifiableList() копирования элементов не происходит, то все изменения, сделанные в исходдной коллекции, будут "видны"в containerValues. Таким образом, результат работы метода ListContainer.getValues() является в некотором смысле неизменяемой ссылкой - через эту ссылку нельзя менять объект, но существуют другие ссылки на данный объект, через которые его можно менять.

Можно ли каким-либо образом избежать возможной ошибки времени исполнения, при этом не позволяя пользователю добавлять элементы в список values, хранящийся в ListContainer? Можно переписать класс ListContainer следующим образом:

Листинг 1.7: Неизменяемый список

```
1 public class ListContainer{
2     private final List<String> values = new ArrayList<String>();
3
4     public List<String> getValues() {
5         return new ArrayList<String>(values);
6     }
7
8 }
```

В этом случае, будет создана независимая копия списка values, которая и будет возвращена пользователю. Любые изменения, производимые с этой копией, не затронут исходный список.

Альтернативный подход можно наблюдать на примере библиотеки gs-collections [2]. В ней есть две отдельные иерархии - для изменяемых коллекций и для неизменяемых. Таким образом, попытка выхвать изменяющий коллекцию метод на неизменяемой коллекции приведет к ошибке компиляции.

Листинг 1.8: Неизменяемый список

```
1 public class ListContainer{
2     private final MutableList<String> values = new FastList<String>();
3
4     public ImmutableList<String> getValues() {
5         return values.toImmutable();
6     }
7
8 }
```

С одной стороны, этот подход позволяет избежать ошибок, связанных с неправомерным изменением объектов во время выполнения программы, но с дургой он требует гораздо более тщательного продумывания интерфейсов, а также более трудоемок в реализации.

добавить про документацию

1.3.3 C#

В C# ключевое слово readonly, примененное к полям имеет тот же смысл, что слово final в Java: присвоение значения полю, которое было объявлено с модификатором readonly может произойти либо по месту его объявления, либо в конструкторе, если это нестатическое поле (для статического поля - в статическом конструкторе).

Листинг 1.9: Ключевое слово readonly

```
1  using System;
2  public class ReadOnlyTest
3  {
4     class MyClass
5     {
6        public int x;
7        public readonly int y = 25; // Initialize a readonly field
8        public readonly int z;
9
10       public MyClass()
11       {
12          z = 24;   // Initialize a readonly instance field
13       }
14
15       public MyClass(int p1, int p2, int p3)
```

[2] https://github.com/goldmansachs/gs-collections

```
16          {
17              x = p1;
18              y = p2;    // OK: readonly field can be reassigned in constructor
19              z = p3;
20          }
21      }
22
23      public static void Main()
24      {
25          MyClass p2 = new MyClass();
26          p2.x = 55;    // OK: field x is not readonly
27          p2.y = 33;    // Error: field y can't be reassigned, as it is readonly
28
29      }
30  }
```

В C# также есть ключевое слово const, которое обозначает, что значение перменной может быть присвоено только в момент ее объявления. То есть, поля объекта, объявленные как readonly, могут иметь различные значения в зависимости от того, какой конструктор и с какими параметрами был вызван. Поле, объявленные как const всегда будет иметь одно и то же значение, так как являются константой времени компиляции.

Листинг 1.10: Ключевое слово const

```
1   using System;
2   public class ReadOnlyTest
3   {
4       class MyClass
5       {
6           public int x;
7           public const int y = 25; // Initialize a const field
8
9           public MyClass()
10          {
11              z = 24;    // Initialize a readonly instance field
12          }
13
14          public MyClass(int p1, int p2)
15          {
16              x = p1;
17              y = p2;    // Error: const field can not be reassigned in constructor
18          }
19      }
```

```
20
21      public static void Main()
22      {
23          MyClass p2 = new MyClass();
24          p2.x = 55;    // OK: field x is not readonly
25          p2.y = 33;    // Error: field y can't be reassigned, as it is const
26
27      }
28  }
```

1.3.4 D

нужно ли тут объяснить что вообще за язык такой - D? И если нужно, то не нужно ли то же самое делать про остальные яызки?

Во второй версии языка прогаммирования D существует два ключевых слова для выражения неизменяемости: const и immutable. Ключевое слово immutable означает, что не существует ссылки, через которую данные могут быть изменены. const обозначает, что по данной ссылке данные менять нельзя, но может существовать ссылка, через которую данные могут быть изменены.

Листинг 1.11: const vs immutable

```
1       int[] foo = new int[5];       // foo is mutable.
2       const int[] bar = foo;        // bar is a const view of mutable data.
3       immutable int[] baz = foo;    // Error:  all views of immutable data must be
            immutable.
4
5       immutable int[] nums = new immutable(int)[5];   // No mutable reference to
            nums may be created.
6       const int[] constNums = nums;                   // Immutable is implicitly
            convertible to const.
7       int[] mutableNums = nums;                       // Error:  Cannot create a
            mutable view of immutable data.
```

В отличае от const в C++, const и immutable в D обеспечивают полноценную глубокую неизменяемость, то есть, любые данные, доступные через const или immutbale объект, также константны или неизменяемы, соответсвенно.

Листинг 1.12: const vs immutable

```
1   class Foo {
2       Foo next;
3       int num;
4   }
5
6       immutable Foo foo = new immutable(Foo);
```

```
7     foo.next.num = 5;  // Error:  foo.next is of type immutable(Foo).
8                        // foo.next.num is of type immutable(int).
```

1.3.5 Javari

Javari - это расширение языка Java, которое добавляет в Java ссылочную неизменяемость, комбинируя статические и динамические проверки неизменяемости. Авторы вводят следующее определение:

Определение 1.3.1. *Абстрактное состояние объекта – это состояние самого объекта и все достижимые из него по ссылкам состояния.*

Javari предоставляет гарантии относительно всего транзитивно достижимого состояния объекта - то есть, состояния самого объекта и состояний всех объектов, доступных из него по нестатическим ссылкам. При этом некоторые части класса могут быть исключны из его абстракного состояния.

Javari добавляет к Java пять дополнительных ключевых слов assignable, readonly, mutable, ?readonly и romaby. Рассмотрим их использование на примерах.

Если какая-либо ссылка объявлена как readonly, то она не может быть использована для изменения объекта, на который она указывает. Пусть, например, переменная rodate имеет тип readonly Date. Тогда rodate не может быть быть использована только для тех операций, которые не меняют объект, на который ссылкается rodate:

Листинг 1.13: Неизменяемая ссылка

```
1 readonly Date rodate = ...; // readonly reference to a Date object
2 rodate.getMonth();     // OK
3 rodate.setYear(2005); // Error
4
5 /*mutable*/ Date date = new Date(); // mutable Date
6 rodate.getMonth();     // OK
7 rodate.setYear(2005); // Error
```

Пусть в Java существует некий ссылочный типа T. Тогда readonly T в Javari является супертипом T. Изменяемая ссылка может быть использована везде, где ожидатся неизменяемая ссылка. Это связано с тем, что неизменяемая ссылка только лишь запрещает менять объект, на который она ссылается, при этом ничего относительно этого объекта не гарантируя.

рисуночек!

На данном рисунке представлена иерархия классов в Javari. Система типов гарантирует, что изменяющие объект методы не могут быть вызваны на неизменяемых ссылках, и что объект, на который ссылается readonly переменная не может быть скопирован в не-readonly переменную.

Ключевое слово readonly может быть спользовано при декларации любой переменной, поля, параметра или возвращаемого значения метода. Его также можно применять к неявному параметру this:

Листинг 1.14: readonly метод

```
1 public char charAt(int index) readonly { ... }
```

В контексте этого метода this будет неизменяемым.

про два метода отличающихся неизменяемостью

Javari также позволяет исключать некоторые поля из абстрактного состояния объекта. По умолчанию все поля являются частью абстрактного состояния объекта и, соответсвенно, не могу быть изменены через неизменяемую ссылку. Если поле объявлено как assignable, то его значение всегда может быть переприсвоено (даже через read-only ссылку). Ключевое слово mutable означает, что поле может быть изменено даже через неизменяемую ссылку. Это может быть полезно для кэширования данных или, например, для реализации логирования, как в следующем примере:

Листинг 1.15: assignable и mutable поля

```
1  class Foo {
2      assignable int hc;
3      final mutable List<String> log = new ArrayList<String>;
4
5      int hashCode() readonly {
6          log.add("hashCode invoked");
7          if (hc == 0) {
8              hc = ... ;
9          }
10     return hc; }
11 }
```

Javari также позволяет добавлять модификаторы мутабельности к типовым параметрам:

Листинг 1.16: Модификаторы мутабельности в типовых параметрах

```
1 /*mutable*/ List</*mutable*/ Date> ld1; // add/remove and mutate elements
2 /*mutable*/ List<readonly Date> ld2; // add/remove
3 readonly List</*mutable*/ Date> ld3; // mutate elements
4 readonly List<readonly Date> ld4; // (neither)
```

добавить про дженерики

Выбор реализации именно ссылочной неизменяемости авторы объясняют несколькими причинами:

- Многие объекты изменяются во время их конструирования и не изменяются после. *explain*
- Ссылочная неизменяемость дает возможность интрфейсу указать, что метод не изменяет объект, ссылку на который принимает в качестве параметра, или что через ссылку, которую мметод возвращает, нельзя менять объект.
- Дополнитлеьный анализ может усилить анализ ссылочной неизменяемости до объектной неизменяемости там, где это необходимо.

Оглавление