УДК 004.432.2

ПРОБЛЕМА ИСПОЛЬЗОВАНИЯ ЧИСТЫХ ФУНКЦИЙ В ЯЗЫКЕ ПРОГРАММИРОВАНИЯ JAVASCRIPT

К.И. ЛАПКОВСКИЙ
(Представлено: ст. преп. Д.В. ПЯТКИН)

Статья посвящена использованию чистых функций в языке программирования, который не разрабатывался для написания программ в функциональном стиле. Рассмотрим применимость языка JavaScript в функциональном программировании на примере главной концепции функционального программирования – использовании чистых функций. Проанализировав примеры различных функций в языке JavaScript, сформулируем правила, которым необходимо следовать при написании программы в функциональном стиле.

Введение. Функциональное программирование с каждым годом приобретает все большую популярность. Одна из концепций функционального программирования – использование чистых функций. Написание программы, опираясь эту на парадигму и используя специальные языки программирования (Scala, Haskel), позволяет создать приложение, которое одинаково легко справляется с любым объемом данных, работая при этом очень быстро. Однако насколько возможно использование не специализированных языков программирования для такой цели? Рассмотрим применение чистых функций в популярном языке программирования JavaScript.

Основной раздел

JavaScript – это язык программирования с слабой динамической типизация, реализующий прототипную модель наследования [1].

Прототипное наследование

Прототип – хранилище методов [2]. Если в объекте вызывается метод, то этот метод сначала ищется в самом объекте, потом в прототипе этого объекта. Прототипы могут формировать цепочки, создавая при этом сколь угодно сложную иерархию. Технически в прототипе можно хранить не только методы, но и свойства, но этой возможностью пользуются редко. Использование прототипов помогает в переиспользовании программный код [3]. Классическое наследование решает ту же задачу.

Чистые функции

Чистыми называют функции, которые не имеют побочных эффектов ввода-вывода и памяти (они зависят только от своих параметров и возвращают только свой результат). Чистые функции обладают несколькими полезными свойствами, многие из которых можно использовать для оптимизации кода:

– если результат чистой функции не используется, её вызов может быть удален без вреда для других выражений;

– результат вызова чистой функции может быть мемоизирован, то есть сохранен в таблице значений вместе с аргументами вызова. Если в дальнейшем функция вызывается с этими же аргументами, её результат может быть взят прямо из таблицы, не вычисляясь (иногда это называется принципом прозрачности ссылок). Мемоизация, ценой небольшого расхода памяти, позволяет существенно увеличить производительность и уменьшить порядок роста некоторых рекурсивных алгоритмов;

– если нет никакой зависимости по данным между двумя чистыми функциями, то порядок их вычисления можно поменять или распараллелить (говоря иначе вычисление чистых функций удовлетворяет принципам thread-safe);

– если весь язык не допускает побочных эффектов, то можно использовать любую политику вычисления. Это предоставляет свободу компилятору комбинировать и реорганизовывать вычисление выражений в программе (например, исключить древовидные структуры) [1].

Примеры функций в JavaScript

В листинге 1 представлены четыре функции. Проанализируем являются ли они чистыми.

Листинг 1 – функции JavaScript

```
// A: Простое умножение
function doubleA(n) {
  return n * 2
}
// B: С переменной
var two = 2
```

```
function doubleB(n) {
  return n * two
}
// C: С вспомогающей функцией
function getTwo() {
  return 2
}
function doubleC(n) {
  return n * getTwo()
}
// D: Преобразование массива
function doubleD(arr) {
  return arr.map(n => n * 2)
}
```

Функция B не является чистой, так как читает переменную из внешней области видимости, тем самым становится зависимой от этой переменной. Если в функцию B два раза передать одинаковые параметры, но перед вторым вызовом функции изменить переменную two, возвращаемые значения после первого и второго вызова функции doubleB будут отличаться.

Функция doubleC так же нельзя назвать чистой. Функция doubleC зависит от функции getTwo, которая может быть переопределена. Если getTwo присвоить значение отличное от функции, то doubleC выбросит исключение, так как нельзя вызвать не функцию.

Рассмотрим функцию doubleD. Внутри функции вызывается метод map, который хранится в прототипе. Так как все экземпляры классов ссылаются на один и тот же прототип, то при изменении прототипа поведение всех экземпляров класса тоже изменится. Помимо этого, функцию можно “перекрыть”, то есть определить функцию с таким же названием, как и в прототипе. Функция doubleD не чистая.

Чтобы понять является ли doubleA чистой функцией необходимо знать механизм, который использует JavaScript для преобразования объектов в примитивы. Когда интрепритатору необходимо преобразовать объект в примитив, для объекта вызывается скрытый метод toPrimitive [4]. ToPrimitive вызывает метод valueOf. Если valueOf возвращает примитив, то этот примитив и используется как результат преобразования. Если valueOf вернул объект, то вызывается метод toString, возвращаемое значение метода toString и будет результатом преобразования. Методы toString, toPrimitive и valueOf можно переопределить или перекрыть, как и обычные методы прототипа.

Заключение

Таким образом возможность изменять прототип и перекрывать функции мешает использовать чистые функции в JavaScript.

Однако можно обезопасить свою программу от перекрытия функций, если явно вызывать методы прототипа, и не доверять поиск метода интерпритатору JavaScript. То есть для вызова метода map рекомендуется использовать синтаксис, представленный в листинге 2.

Листинг 2 – вызов метода прототипа

```
Array.prototype.map.call(array, ()=>{})
```

ЛИТЕРАТУРА

1. Wikipedia [Электронный ресурс] // Функциональное программирование. – Режим доступа: https://ru.wikipedia.org/wiki/Функциональное_программирование#Чистые_функции. – Дата доступа: 24.09.2018.
2. Современный учебник JavaScript [Электронный ресурс] // Прототип объекта. – Режим доступа: https://learn.javascript.ru/prototype. Дата доступа: 24.09.2018.
3. MDN web docs [Электронный ресурс] // Object.prototype. – Режим доступа: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/prototype. – Дата доступа: 24.09.2018.
4. You don’t know JS [Электронный ресурс] // Coercion. – Режим доступа: https://github.com/getify/You-Dont-Know-JS/blob/master/types%20%26%20grammar/ch4.md. – Дата доступа: 25.09.2018.