Министерство образования и науки Российской Федерации
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Российский экономический университет им. Г.В. Плеханова»

Факультет математической экономики, статистики и информатики

Кафедра управления информационными системами
и программирования

Курсовая работа

по дисциплине «Языки и методы программирования»

на тему «Функциональное программирование на языке JavaScript»

Выполнил студент 2 курса
Группы № «424» / дневное отделение
Факультета математической экономики,
статистики и информатики
Эль-Айясс Дани Валид
Подпись _____

Научный руководитель:
старший преподаватель кафедры
управления информационными системами
и программирования
Лыкошин Александр Сергеевич
Подпись _____

Москва – 2017 г.

Оглавление

Введение

Функциональный подход к программированию основан на идее, что вся обработка информации и получение искомого результата могут быть представлены в виде вложенных или рекурсивных вызовов функций, выполняющих некоторые действия, так что значения одной функции используется как аргумент другой. Значение этой функции становится аргументом следующей и так далее, пока не будет получен результат - решение задачи. Программы строятся из логически расчлененных определений функций. Определения состоят из управляющих структур, организующих вычисления, и из вложенных вызовов функций. Функциональные программы не содержат операторов присваивания, а переменные, получив однажды значение, никогда не изменяются. Более того, функциональные программы вообще не имеют побочных эффектов. Обращение к функции не вызывает иного эффекта кроме вычисления результата. Это устраняет главный источник ошибок и делает порядок выполнения функций несущественным: так как побочные эффекты не могут изменять значение выражения, оно может быть вычислено в любое время. Поскольку выражения могут быть вычислены в любое время, можно свободно заменять переменные их значениями и наоборот. Функциональные программы более удобными для математической обработки, по сравнению с общепринятыми аналогами. Все эти характеристики приводят к более быстрой разработке короткого, простого и безошибочного кода.

Цель данной курсовой работы – ознакомиться с функциональным программированием на языке JavaScript.

Задачами курсовой работы является:

1. Рассмотреть общие сведения о парадигмах программирования в целом, и, в частности, о функциональном программировании;

2. Рассмотреть историю возникновения функционального программирования;
3. Рассмотреть особенности функционального программирования;
4. Проанализировать язык JavaScript с точки зрения функционального программирования;
5. Проанализировать основные концепции функционального программирования на языке JavaScript.

Предметом исследования данной работы является изучение функционального программирования на JS.

Объектом исследование курсовой работы является язык JS.

1. Теоретические часть

1.1. Три парадигмы программирования

Перед тем, как приступить к изучению функционального программирования, стоит понять, что же из себя представляет функциональное программирование. Функционально программирование – это одна из парадигм программирования.

Парадигма программирования – это совокупность идей и понятий, определяющих стиль написания компьютерных программ. Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

Давайте рассмотрим, какие вообще бывают парадигмы программирования.

Существуют три основных парадигмы: императивное программирование и декларативное программирование, которое в свою очередь подразделяется на логическое и функциональное.

В случае логического программирования основные принципы таковы: вы оперируете математической логикой для вывода новых фактов и состояний из уже известных.

Принцип работы с императивным программированием, наиболее распространённым, заключается в формировании инструкций, последовательных команд, которые должна выполнять машина.

Императивное программирование работает со строго определёнными состояниями и инструкциями. Функциональное же основывается на взаимодействии с функциями, то есть некими процессами, описывающими связь между входными и выходными параметрами. Таким образом, в то время,

как императивный язык описывает конкретное действие с известными входными параметрами, функциональный описывает некое тело взаимодействий, не опускаясь до конкретных случаев.

1.2.История функционального программирования

Теория, положенная в основу функционального подхода, родилась в 20-х – 30-х годах прошлого века. В числе разработчиков математических основ функционального программирования можно назвать Мозеса Шёнфинкеля (Германия и Россия) и Хаскелла Карри (Англия), разработавших комбинаторную логику, а также Алонзо Чёрча (США), создателя λ-исчисления.

Совместно с другими учёными Алонзо разработал формальную систему названную λ-исчислением. Система по сути была языком программирования для одной из воображаемых машин. Она была основана на функциях, которые принимают в качестве аргументов функции, и возвращают функцию. Такая функция была обозначена греческой буквой λ, что дало название всей системе.

Функциональное программирование – это практическая реализация идей Алонзо Чёрча.

1.3.Особенности функционального программирования

Каждая парадигма программирования отличается теми или иными свойствами, особо выделяемыми исследователями в качестве дифференцирующих свойств, отделяющих одну парадигму от другой. В качестве основных свойств функциональных языков традиционно рассматриваются следующие:

- краткость и простота;

- строгая типизация[1];
- модульность – это принцип программирования, состоящий в том, что большие и сложные программы разрабатываются и отлаживаются по частям, которые затем объединяются в единый комплекс;
- функции – это значения;
- чистота (отсутствие побочных эффектов).

1.4.О JavaScript

Важно отметить, что парадигма программирования не определяется однозначно языком программирования; практически все современные языки программирования в той или иной мере допускают использование различных парадигм. JavaScript не является исключением.

JavaScript – это мультипарадигменный язык программирования. Он поддерживает как императивный, так и функциональный стили.

JavaScript не придирчив, когда дело доходит до парадигм. На нем можно смешивать смешать императивный и функциональный код, как сочтете нужным, и все равно получите нужный результат.

JavaScript. одновременно поддерживает широкий спектр программных стилей в пределах одной и той же кодовой базы, так что правильные выборы, касающиеся читаемости и производительности зависят от вас.

Функциональный JavaScript не обязательно применять во всем коде. Совсем немного знаний о функциональном подходе, поможет сделать код на

[1] Строгая типизация выделяется тем, что язык не позволяет смешивать в выражениях различные типы и не выполняет автоматические неявные преобразования, например, нельзя вычесть из строки множество.

JavaScript чище и элегантнее, независимо от того, какой подход вы предпочитаете.

2. Функциональное программирование на JS

В данной главе будут рассмотрены концепции, имеющие решающее значение для понимания функционального программирования:

- Чистота: чистые функции, нечистые функции, побочные эффекты.
- Состояние: с ним и без него.
- Неизменяемость и изменяемость.
- Императивное и декларативное программирование.
- Функции высшего порядка.
- Функциональное программирование.

2.1. Чистые и нечистые функции

2.1.1. Чистые функции

Возвращаемое значение чистой функции зависит только от ее входных данных (аргументов) и не влечет никаких побочных эффектов. С одним и тем же входящим аргументом результат всегда будет одинаковый. Пример:

```
function half(x) {
    return x / 2;
}
```

Функция `half(x)` принимает число `x` и возвращает значение половины `x`. Если мы передадим этой функции аргумент `8`, она вернет `4`. После вызова чистая функция всегда может быть заменена результатом своей работы. Например, мы могли бы заменить `half(8)` на `4`: где бы эта функция не использовалась в нашем коде, подмена никак не повлияла бы на конечный результат. Это называется ссылочной прозрачностью.

Чистые функции зависят только от того, что им передано. Например, чистая функция не может ссылаться на переменные из области видимости родителя, если они явно не передаются в нее в качестве аргументов. Но и даже

тогда функция не может изменять что-либо в родительской области видимости.

Итого:

- Чистые функции должны принимать аргументы.
- Одни и те же входные данные (аргументы) всегда произведут одинаковые выходные данные (вернут одинаковый результат).
- Чистые функции основываются только на внутреннем состоянии и не изменяют внешнее (console.log изменяет глобальное состояние).
- Чистые функции не производят побочных эффектов.
- Чистые функции не могут вызывать нечистые функции.

2.1.2. Нечистые функции

Нечистая функция изменяет состояние вне своей области видимости. Любые функции с побочными эффектами (см. далее)—нечистые. Пример нечистой функции:

```
var num = 8;
function half() {
    return num / 2;
}
```

2.1.3. Побочные эффекты в JavaScript

Когда функция или выражение изменяет состояние вне своего контекста, результат является побочным эффектом. Если функция производит побочные эффекты, она считается нечистой. Функции, вызывающие побочные эффекты, менее предсказуемы и их труднее тестировать, поскольку они приводят к изменениям вне их локальной области видимости.

2.1.4. Вывод

Много качественного кода состоит из нечистых функций, процедурно вызывающихся чистыми. Это все равно несет массу преимуществ для тестирования и неизменяемости.

2.2.Функции с состоянием и без

2.2.1. С состоянием

Программы, приложения или компоненты с состоянием хранят в памяти данные о текущем состоянии. Они могут изменять состояние, а также имеют доступ к его истории. Следующий пример демонстрирует это:

```
var number = 1;
function increment() {
    return number++;
}
// глобальная переменная изменяется: number = 2
increment();
```

2.2.2. Без состояния

Функции или компоненты без состояния выполняют задачи, словно каждый раз их запускают впервые. Они не ссылаются или не используют в своем исполнении раннее созданные данные. Отсутствие состояния обеспечивает ссылочную прозрачность. Функции зависят только от их аргументов и не имеют доступа, не нуждаются в знании чего-либо вне их области видимости. Чистые функции не имеют состояния. Пример:

```
var number = 1;
function increment(n) {
    return n + 1;
}
// глобальная переменная НЕ изменяется: возвращает 2
increment(number);
```

Приложения без состояния все еще управляют состоянием. Однако они возвращают свое текущее состояние без изменения предыдущего состояния. Это принцип функционального программирования.

2.2.3. Вывод

Управление состоянием важно для любого сложного приложения. Функции или компоненты с состоянием изменяют состояние и его историю, их труднее тестировать и отлаживать. Функции без состояния полагаются только на свои входные данные для создания данных выходных. Программа без состояния возвращает новое состояние, а не модифицирует существующее состояние.

2.3.Концепция изменяемости и изменяемости

2.3.1. Описание

Концепции неизменяемости и изменяемости имеет большое значение в функциональном программировании. Важно знать, что эти термины означают в классическом понимании, и как они реализуются в JavaScript. Определения достаточно просты:

Если объект является неизменяемым, его значение не может быть изменено после создания.

Если объект изменяем, его значение может быть изменено после создания.

В JavaScript строки и числовые литералы[2] реализованы неизменяемыми. Это легко понять, если рассмотреть, как мы работаем с ними:

```
var str = 'Hello!';
var anotherStr = str.substring(2);
// результат: str = 'Hello!' (не изменена)
// результат: anotherStr = 'llo!' (новая строка)
```

Используя метод `.substring()` на нашем `'Hello!'`, строка не изменяет исходную строку. Вместо этого она создает новую строку. Мы могли

[2] Литерал – это любое значение, указанное явным образом в коде. В качестве литералов в JavaScript могут выступать числа, строки (текстовые значения), логические значения и т.д.

бы переопределить значение переменной `str` на что-то другое, но, как только мы создали нашу строку `'Hello!'`, она навсегда останется `'Hello!'`.

Числовые литералы также неизменяемы. Следующий пример всегда будет иметь одинаковый результат:

```
var three = 1 + 2;
// результат: three = 3
```

Ни при каких обстоятельствах `1 + 2` не может стать чем-либо, кроме `3`.

Это демонстрирует, что в JavaScript присутствует реализация неизменяемости. Однако язык JavaScript позволяет изменить многое. Например, объекты и массивы. Рассмотрим следующий пример:

```
var arr = [1, 2, 3];
arr.push(4);
// результат: arr = [1, 2, 3, 4]

var obj = { greeting: 'Hello' };
obj.name = 'Jon';
// результат: obj = { greeting: 'Hello', name: 'Jon' }
```

В этих примерах исходные объекты изменены.

2.3.2. Вывод

В целом, JavaScript – язык с сильной изменяемостью. Некоторые стили JavaScript-кодирования опираются на эту врожденную изменяемость. Ключевая особенность функционального программирования — неизменяемость. Поэтому при написании функционального JavaScript, реализация неизменяемости требует внимательности. Если вы что-то нечаянно модифицируете, JavaScript не будет выбрасывать ошибки.

Неизменяемость имеет свои преимущества. В результате получается код, который проще понимать.

Недостатком неизменяемости является то, что многие алгоритмы и операции не могут быть эффективно реализованы.

2.4. Функции высшего порядка

2.4.1. Описание

Функция высшего порядка — это функция, которая принимает другую функцию в качестве аргумента или возвращает функцию в результате.

В JavaScript функции являются объектами первого класса[3]. Они могут храниться и передаваться как значения: мы можем присвоить функцию переменной или передать функцию другой функции.

```
const double = function(x) {
    return x * 2;
}
const timesTwo = double;

timesTwo(4);
// результат: возвращает 8
```

Мы также можем передать функцию в качестве аргумента любой другой созданной нами функции, а затем выполнить этот аргумент:

```
function sayHi() {
    alert('Hi!');
}
function greet(greeting) {
    greeting();
}
greet(sayHi); // "Hi!"
```

Примечание: при передаче именованной функции в качестве аргумента, как и в двух приведенных выше примерах, мы не используем круглые скобки `()`. Таким образом мы передаем функцию как объект. Круглые скобки выполняют функцию и передают результат вместо самой функции.

[3] Объектами первого класса называются элементы, которые могут быть переданы как параметр, возвращены из функции, присвоены переменной.

Функции высшего порядка также могут возвращать другую функцию:

```
function whenMeetingJohn() {
    return function() {
        alert('Hi!');
    }
}
var atLunchToday = whenMeetingJohn();

atLunchToday(); // "Hi!"
```

2.4.2. Вывод

Природа JavaScript функций как объектов первого класса делает их основой функционального программирования в JavaScript.

2.5.Функциональное программирование

Были рассмотрены такие концепции, как чистота функции, отсутствие состояния функции, неизменяемость, и функции высшего порядка. Все они важны для понимания парадигмы функционального программирования.

Функциональное программирование охватывает приведенные выше концепции следующими способами:

- Основные функции реализованы с использованием чистых функций без побочных эффектов.
- Данные неизменяемы.
- Функциональные программы не имеют состояния.

Примечание: если бы мы попытались написать код на JavaScript, состоящий только из чистых функций без побочных эффектов, он не смог бы взаимодействовать с окружением и поэтому не был бы особенно полезным.

Неизменяемость данных и отсутствие состояния гарантируют, что состояние программы не изменяется. Вместо этого возвращаются новые значения. Чистые функции используются для функциональности ядра. Чтобы

запустить программу и обработать необходимые побочные эффекты, нечистые функции могут вызывать чистые.

Заключение

В качестве заключение хотелось бы отметить, что из-за того, что JavaScript является мультипарадигменным языком, поддерживающим как императивное программирование, так и функциональное, он предлагает очень большой спектр возможностей для разработчиков, работающий на этом языке.

В данной курсовой работе были решены следующие задачи:

1. Рассмотрены общие сведения о парадигмах программирования в целом, и, в частности, о функциональном программировании;
2. Рассмотрена история возникновения функционального программирования;
3. Рассмотрены особенности функционального программирования;
4. Проанализирован язык JavaScript с точки зрения функционального программирования;
5. Проанализированы основные концепции функционального программирования на языке JavaScript.

Список литературы

1. Конституция Российской Федерации. Принята всенародным голосованием 12.12.1993г (с учетом поправок, внесенных Законами Российской Федерации о поправках к Конституции Российской Федерации от 30.12.2008г. № 6-ФКЗ и от 30.12.2008г. № 7-ФКЗ) // Российская газета, 2009г. № 7 - от 21января.

2. Федеральный закон от 04.05.2011г. № 99-ФЗ «О лицензировании отдельных видов деятельности» (ред. с изм. от 21.11.2011г. № 327-ФЗ) // СЗ РФ. 2011г. № 19. Ст. 2716; 2011г. № 48. Ст. 6728.

3. Федеральный закон от 06.04.2011 N 63-ФЗ «Об электронной подписи» (ред. от 01.07.2011 N 169-ФЗ) // СЗ РФ. 2011г. № 15. Ст. 2036; 2011г. № 27. Ст. 3880.

4. Постановление Правительства РФ от 25.12.2007г. № 931 «О некоторых мерах по обеспечению информационного взаимодействия государственных органов и органов местного самоуправления при оказании государственных услуг гражданам и организациям» (ред. от 10.03.2009г.) // ФЗ РФ. 2007г. № 53. Ст. 6627; 2009г. № 12. Ст. 1429.

5. Харрисон Дж. Введение в функциональное программирование: научно-практическое пособие / Дж. Харрисон. – М.: Мир, 1993 – 170 с.

6. Харрисон П., Филд А. Функциональное программирование: научно-практическое пособие / П. Харрисон, А. Филд. – М.: Мир, 1997 – 312 с.

7. Ликбез по типизации в языках программирования. [Электронный ресурс]. URL: https://habrahabr.ru/post/161205/ (дата обращения: 21.06.2017).

8. Основные принципы программирования: функциональное программирование. [Электронный ресурс]. URL: https://tproger.ru/translations/functional-programming-concepts/ (дата обращения: 21.06.2017).

9. Функциональное программирование. [Электронный ресурс]. URL: https://geekbrains.ru/posts/functional_programming (дата обращения: 21.06.2017).

10. Функциональное программирование для всех. [Электронный ресурс]. URL: https://habrahabr.ru/post/142351/ (дата обращения: 21.06.2017).

11. Функциональное программирование на Javascript. [Электронный ресурс]. URL: https://habrahabr.ru/post/154105/ (дата обращения: 21.06.2017).

12. Функциональное программирование с примерами на JavaScript. Часть первая. Основные техники функционального программирования. [Электронный ресурс]. URL: https://tproger.ru/translations/functional-js-1/ (дата обращения: 21.06.2017).

13. JS для начинающих. Урок 1.1: Литералы. [Электронный ресурс]. URL: https://true-coder.ru/js-dlya-nachinayushhix/js-dlya-nachinayushhix-urok-1-literaly.html (дата обращения: 21.06.2017).

14. λ-исчисление. Часть первая: история и теория. [Электронный ресурс]. URL: https://habrahabr.ru/post/215807/ (дата обращения: 21.06.2017).

15. λ-исчисление. Часть вторая: практика. [Электронный ресурс]. URL: https://habrahabr.ru/post/215991/ (дата обращения: 21.06.2017).