

Содержание

Предисловие	12
Благодарности	14
Об этой книге	16
Структура книги	16
Кому адресована книга	18
Как пользоваться этой книгой	18
Примеры исходного кода	18
Условные обозначения	19
Об авторе	20
От издательства	20
Часть I. Умение мыслить функционально	21
Глава 1. Основы функционального программирования	23
1.1. Чем может помочь функциональное программирование	25
1.2. Сущность функционального программирования	26
1.2.1. Декларативный характер функционального программирования	28
1.2.2. Чистые функции и трудности, связанные с побочными эффектами	30
1.2.3. Ссылочная прозрачность и подстановочность	35
1.2.4. Сохранение данных неизменяемыми	37
1.3. Преимущества функционального программирования	38
1.3.1. Побуждение к декомпозиции сложных задач	39
1.3.2. Обработка данных с помощью текучих цепочек	41
1.3.3. Реагирование на сложность асинхронных приложений	43
Резюме	45
Глава 2. Сценарий высшего порядка	47
2.1. Причины для выбора JavaScript	48
2.2. ФП в сравнении с ООП	49
2.2.1. Управление состоянием объектов в JavaScript	56
2.2.2. Обращение с объектами как со значениями	57
2.2.3. Глубокое замораживание подвижных частей	60
2.2.4. Перемещение по графам объектов и их модификация с помощью линз	62
2.3. Функции	64
2.3.1. Функции первого класса	65

2.3.2. Функции высшего порядка	66
2.3.3. Способы вызова функций	69
2.3.4. Методы функций	70
2.4. Замыкания и области видимости	71
2.4.1. Трудности, связанные с соблюдением глобальной области видимости	74
2.4.2. Область видимости функций JavaScript	75
2.4.3. Область видимости псевдоблока	76
2.4.4. Практические примеры применения замыканий	78
Резюме	82
Часть II. Погружаемся в функциональное программирование	83
Глава 3. Меньше структур данных и больше операций	85
3.1. Общее представление о потоке управления прикладной программой	86
3.2. Связывание методов с цепочку	87
3.3. Связывание функций в цепочку	88
3.3.1. Общее представление о лямбда-выражениях	89
3.3.2. Преобразование данных с помощью операции <code>_ .map</code>	91
3.3.3. Получение результатов с помощью операции <code>_ .reduce</code>	94
3.3.4. Исключение ненужных элементов с помощью операции <code>_ .filter</code>	98
3.4. Анализ прикладного кода	100
3.4.1. Декларативный характер цепочек функций с отложенными вычислениями	100
3.4.2. SQL-подобные данные: функции как данные	105
3.5. Умение мыслить рекурсивно	107
3.5.1. Что такое рекурсия	107
3.5.2. Как научиться мыслить рекурсивно	108
3.5.3. Рекурсивно определяемые структуры данных	111
Резюме	115
Глава 4. На пути к повторно используемому, модульному коду	117
4.1. Цепочки методов в сравнении с конвейерами функций	118
4.1.1. Связывание методов в цепочку	120
4.1.2. Организация функций в конвейеры	121
4.2. Требования к совместимости функций	122
4.2.1. Совместимые по типу функции	122
4.2.2. Функции и арность: вариант для кортежей	123
4.3. Вычисление каррированных функций	127
4.3.1. Эмуляция фабрик функций	130
4.3.2. Реализация повторно используемых шаблонов функций	132
4.4. Частичное применение и привязка параметров	133
4.4.1. Расширение базовых языковых средств	135

4.4.2. Связывание отложенных функций	136
4.5. Составление конвейеров функций	137
4.5.1. Представление о композиции HTML-виджетов	138
4.5.2. Композиция функций: отделение описания от вычисления	139
4.5.3. Композиция с помощью функциональных библиотек	143
4.5.4. Меры борьбы с наличием нечистого и чистого кода	145
4.5.5. Введение в бесточечное программирование	147
4.6. Организация потока управления с помощью комбинаторов функций	149
4.6.1. Тождественность (I-комбинатор)	149
4.6.2. Ответвление (K-комбинатор)	150
4.6.3. Перемена (OR-комбинатор)	150
4.6.4. Последовательность (S-комбинатор)	151
4.6.5. Комбинатор разветвления	152
Резюме	154
Глава 5. Проектные шаблоны и сложность	155
5.1. Недостатки императивной обработки ошибок	156
5.1.1. Обработка ошибок в блоке операторов <code>try-catch</code>	156
5.1.2. Причины не генерировать исключения в функциональных программах	158
5.1.3. Затруднения, возникающие при проверке пустого значения	159
5.2. Выработка лучшего решения с помощью функторов	160
5.2.1. Заключение ненадежных значений в оболочку	161
5.2.2. Пояснение назначения функторов	163
5.3. Функциональный способ обработки ошибок с помощью монад	167
5.3.1. Монады: от потока управления до потока данных	168
5.3.2. Обработка ошибок с помощью монад <code>Maybe</code> и <code>Either</code>	172
5.3.3. Взаимодействие с внешними ресурсами с помощью монады типа <code>IO</code>	183
5.4. Монадические цепочки и композиции	186
Резюме	193
Часть III. Расширение функциональных навыков	195
Глава 6. Отказоустойчивость прикладного кода	197
6.1. Влияние функционального программирования на модульные тесты	198
6.2. Трудности тестирования императивных программ	199
6.2.1. Трудность выявления и разбиения на простые задачи	200
6.2.2. Зависимость от общих ресурсов, приводящая к непостоянным результатам	201
6.2.3. Предопределенный порядок вычисления	202
6.3. Тестирование функционального кода	204
6.3.1. Интерпретация функции как “черного ящика”	204

6.3.2. Концентрация основного внимания на бизнес-логике, а не на потоке управления программой	205
6.3.3. Отделение чистого кода от нечистого путем монадического изолирования	207
6.3.4. Имитация внешних зависимостей	210
6.4. Фиксация спецификаций при тестировании на основе свойств	212
6.5. Количественная оценка эффективности тестов через покрытие ими кода	220
6.5.1. Количественная оценка эффективности тестирования функционального кода	221
6.5.2. Количественная оценка сложности функционального кода	225
Резюме	228
Глава 7. Оптимизация функционального кода	229
7.1. Внутренний механизм выполнения функций	230
7.1.1. Карринг и стек контекста функций	232
7.1.2. Трудности, связанные с рекурсивным кодом	236
7.2. Отсрочка выполнения с помощью отложенного вычисления	238
7.2.1. Исключение вычислений с помощью комбинатора чередования функций	239
7.2.2. Использование преимуществ сокращенного слияния	240
7.3. Реализация стратегии вызовов по требованию	242
7.3.1. Общее представление о запоминании	243
7.3.2. Запоминание функций, требующих интенсивных вычислений	243
7.3.3. Выгодное применение карринга и запоминания	247
7.3.4. Декомпозиция для максимального запоминания	247
7.3.5. Применение запоминания к рекурсивным вызовам	248
7.4. Рекурсия и оптимизация хвостовых вызовов	250
7.4.1. Преобразование нехвостовых вызовов в хвостовые	252
Резюме	256
Глава 8. Обработка асинхронных событий и данных	257
8.1. Трудности, возникающие при написании асинхронного кода	259
8.1.1. Создание временных зависимостей между функциями	259
8.1.2. Неизбежное обращение к пирамиде обратных вызовов	260
8.1.3. Применение стиля передачи продолжений	263
8.2. Достижение асинхронного поведения объектов первого класса с помощью обязательств	267
8.2.1. Цепочки будущих методов	270
8.2.2. Композиция синхронного и асинхронного поведения	275
8.3. Генерирование данных по требованию	278
8.3.1. Генераторы и рекурсия	281
8.3.2. Протокол итератора	283

8.4. Функциональное и реактивное программирование средствами RxJS	284
8.4.1. Данные как наблюдаемые объекты	285
8.4.2. Функциональное и реактивное программирование	286
8.4.3. Библиотека RxJS и обязательства	288
Резюме	290
Приложение А. Библиотеки JavaScript, упоминаемые в книге	291
Функциональные библиотеки JavaScript	291
Библиотека Lodash	291
Библиотека Ramda	292
Библиотека RxJS	292
Другие применяемые библиотеки	293
Библиотека Log4js	293
Библиотека QUnit	293
Библиотека Sinon	293
Библиотека Blanket	294
Библиотека JSCheck	294
Предметный указатель	295