

Краткое содержание

Предисловие	18
Благодарности	20
О книге	22
Об авторах	26
Иллюстрация на обложке	27
От издательства	28
Глава 1. Введение	29
Глава 2. Дублирование кода не всегда плохо: дублирование кода и гибкость	46
Глава 3. Исключения и другие паттерны обработки ошибок в коде	75
Глава 4. Баланс между гибкостью и сложностью	112
Глава 5. Преждевременная оптимизация и оптимизация критического пути: решения, влияющие на производительность кода	137
Глава 6. Простота и затраты на обслуживание API	170
Глава 7. Эффективная работа с датой и временем	198

8 Краткое содержание

Глава 8. Локальность данных и использование памяти	258
Глава 9. Сторонние библиотеки: используемые библиотеки становятся кодом	290
Глава 10. Целостность и атомарность в распределенных системах	323
Глава 11. Семантика доставки в распределенных системах	347
Глава 12. Управление версиями и совместимостью.	374
Глава 13. Современные тенденции разработки и затраты на сопровождение кода.	433

Оглавление

Предисловие	18
Благодарности	20
О книге	22
Для кого эта книга.	22
Структура книги.	22
О коде	24
Форум liveBook	25
Об авторах	26
Иллюстрация на обложке	27
От издательства	28
Словарь паттернов проектирования	28
Глава 1. Введение	29
1.1. Последствия каждого решения и паттерна	30
1.1.1. Решения в модульном тестировании.	31
1.1.2. Соотношение модульных и интеграционных тестов	32
1.2. Программные паттерны проектирования и почему они работают не всегда	34
1.2.1. Измерение скорости выполнения.	39

10 Оглавление

1.3. Архитектурные паттерны проектирования и почему они работают не всегда	41
1.3.1. Масштабируемость и эластичность	41
1.3.2. Скорость разработки.	42
1.3.3. Сложность микросервисов	43
Итоги.	45
Глава 2. Дублирование кода не всегда плохо: дублирование кода и гибкость	46
2.1. Общий код в кодовых базах и дублирование	47
2.1.1. Добавление нового бизнес-требования, для которого дублирование кода необходимо	49
2.1.2. Реализация нового бизнес-требования	50
2.1.3. Оценка результата	51
2.2. Библиотеки и совместное использование кода в кодовых базах	52
2.2.1. Оценка компромиссов и недостатков совместно используемых библиотек	53
2.2.2. Создание совместно используемой библиотеки.	54
2.3. Выделение кода в отдельный микросервис.	55
2.3.1. Компромиссы и недостатки отдельного сервиса	58
2.3.2. Выводы о выделении отдельных сервисов	62
2.4. Улучшение слабой связанности за счет дублирования кода	63
2.5. Проектирование API с наследованием для сокращения дублирования.	67
2.5.1. Выделение базового обработчика запросов	68
2.5.2. Наследование и сильная связанность	70
2.5.3. Компромиссы между наследованием и композицией	72
2.5.4. Дублирование внутреннее и ситуативное.	73
Итоги.	74
Глава 3. Исключения и другие паттерны обработки ошибок в коде.	75
3.1. Иерархия исключений	77
3.1.1. Универсальный и детализированный подход к обработке ошибок	78

3.2. Лучшие паттерны для обработки исключений в собственном коде	81
3.2.1. Обработка проверяемых исключений в общедоступном API	82
3.2.2. Обработка непроверяемых исключений в общедоступном API	83
3.3. Антипаттерны в обработке исключений	85
3.3.1. Закрытие ресурсов при возникновении ошибки	87
3.3.2. Антипаттерн использования исключений для управления программной логикой	89
3.4. Исключения из сторонних библиотек	90
3.5. Исключения в многопоточных средах	93
3.5.1. Исключения в асинхронной программной логике с API обещаний	96
3.6. Функциональный подход к обработке ошибок с Try	99
3.6.1. Использование Try в рабочем коде	103
3.6.2. Объединение Try с кодом, выдающим исключение	105
3.7. Сравнение производительности кода обработки исключений	106
Итоги	110
Глава 4. Баланс между гибкостью и сложностью	112
4.1. Мощный, но не расширяемый API	113
4.1.1. Проектирование нового компонента	113
4.1.2. Начиная с простого	114
4.2. Возможность предоставления собственной библиотеки метрик	118
4.3. Обеспечение расширяемости API с использованием перехватчиков	121
4.3.1. Защита от непредвиденного использования API перехватчиков	123
4.3.2. Влияние API перехватчиков на производительность	125
4.4. Обеспечение расширяемости API за счет использования прослушивателей	129
4.4.1. Прослушиватели и перехватчики	130
4.4.2. Неизменяемость архитектуры	131

12 Оглавление

4.5. Анализ гибкости API и затраты на обслуживание	133
Итоги	136
Глава 5. Преждевременная оптимизация и оптимизация критического пути: решения, влияющие на производительность кода	137
5.1. Когда преждевременная оптимизация — зло.	138
5.1.1. Создание конвейера обработки учетных записей	139
5.1.2. Оптимизация обработки на основании ложных утверждений	140
5.1.3. Оценка оптимизации производительности	141
5.2. Критические пути в коде	144
5.2.1. Принцип Парето в контексте программных систем	146
5.2.2. Настройка количества параллельных пользователей (поток) для заданного уровня SLA	147
5.3. Словарный сервис с потенциальным критическим путем	148
5.3.1. Получение «слова дня»	149
5.3.2. Проверка существования слова	151
5.3.3. Предоставление доступа к WordsService с использованием сервиса HTTP	151
5.4. Обнаружение критического пути в коде	153
5.4.1. Применение Gatling для создания тестов производительности API.	153
5.4.2. Измерение хронометража кодовых путей с использованием MetricRegistry	157
5.5. Повышение производительности критического пути	159
5.5.1. Создание микротеста JMH для существующего решения	160
5.5.2. Оптимизация проверки с использованием кэширования. . . .	161
5.5.3. Увеличение количества входящих слов в тестах производительности	167
Итоги	169
Глава 6. Простота и затраты на обслуживание API	170
6.1. Базовая библиотека, используемая другими инструментами	171
6.1.1. Создание клиента облачного сервиса	172
6.1.2. Стратегии аутентификации	173

6.1.3. Понимание механизма конфигурации.	175
6.2. Прямое предоставление настроек зависимой библиотеки	179
6.2.1. Конфигурация пакетного инструмента	181
6.3. Абстрагирование настроек зависимой библиотеки.	183
6.3.1. Конфигурация стримингового сервиса	184
6.4. Добавление новой настройки для облачной клиентской библиотеки	186
6.4.1. Добавление новой настройки в пакетный инструмент.	187
6.4.2. Добавление новой настройки в стриминговый сервис	188
6.4.3. Сравнение UX-ориентированности и удобства обслуживания в двух решениях	189
6.5. Удаление настроек в облачной клиентской библиотеке.	190
6.5.1. Удаление настройки из пакетного инструмента.	192
6.5.2. Удаление настроек из стримингового сервиса.	194
6.5.3. Сравнение UX-ориентированности и затрат на обслуживание для двух решений.	196
Итоги	197
Глава 7. Эффективная работа с датой и временем.	198
7.1. Концепции представления даты и времени	200
7.1.1. Машинное время: моменты времени, эпохи и интервалы.	200
7.1.2. Календарные системы, даты, время и периоды	204
7.1.3. Часовые пояса, UTC и смещения от UTC	211
7.1.4. Концепции даты и времени, вызывающие приступ головной боли	216
7.2. Подготовка к работе с информацией о дате и времени	219
7.2.1. Ограничение объема работ	219
7.2.2. Уточнение требований к дате и времени	221
7.2.3. Использование подходящих библиотек или пакетов	227
7.3. Реализация кода даты и времени	229
7.3.1. Последовательное применение концепций.	229
7.3.2. Отказ от значений по умолчанию в целях улучшения тестируемости	232

14 Оглавление

7.3.3. Текстовое представление даты и времени	239
7.3.4. Объяснение кода в комментариях	246
7.4. Граничные случаи	249
7.4.1. Арифметические операции с календарями	249
7.4.2. Переходы часовых поясов в полночь.	250
7.4.3. Обработка неоднозначного или пропущенного времени	251
7.4.4. Изменения данных часовых поясов	251
Итоги	256
Глава 8. Локальность данных и использование памяти	258
8.1. Что такое локальность данных?.	259
8.1.1. Перемещение вычислений к данным	260
8.1.2. Масштабирование обработки с использованием локальности данных.	261
8.2. Секционирование и разбиение данных	263
8.2.1. Автономное секционирование больших данных	263
8.2.2. Секционирование и сегментирование	266
8.2.3. Алгоритмы секционирования	267
8.3. Соединение наборов больших данных из нескольких секций	270
8.3.1. Соединение данных на одной физической машине.	271
8.3.2. Соединение, требующее перемещения данных	273
8.3.3. Оптимизация соединения за счет широковещательной рассылки.	274
8.4. Обработка данных: память и диск	276
8.4.1. Обработка с хранением данных на диске	276
8.4.2. Для чего нужна парадигма MapReduce?	277
8.4.3. Вычисление времени обращения	280
8.4.4. Обработка данных в памяти	281
8.5. Реализация соединений с использованием Apache Spark.	283
8.5.1. Реализация соединения без рассылки	284
8.5.2. Реализация соединения с рассылкой	287
Итоги.	288

Глава 9. Стронние библиотеки: используемые библиотеки становятся кодом290
9.1. Импортрование библиотеки и ответственность за ее настройки: берегитесь значений по умолчанию291
9.2. Модели параллельного выполнения и масштабируемость296
9.2.1. Использование асинхронных и синхронных API298
9.2.2. Распределенная масштабируемость301
9.3. Тестируемость.303
9.3.1. Тестовая библиотека.304
9.3.2. Тестирование с использованием объектов fake (тестовых двойников) и mock.306
9.3.3. Набор инструментов интеграционного тестирования311
9.4. Зависимости сторонних библиотек312
9.4.1. Предотвращение конфликтов версий313
9.4.2. Слишком много зависимостей.315
9.5. Выбор и обслуживание сторонних зависимостей.316
9.5.1. Первые впечатления.316
9.5.2. Разные подходы к повторному использованию кода.317
9.5.3. Привязка к производителю318
9.5.4. Лицензирование319
9.5.5. Библиотеки и фреймворки.319
9.5.6. Безопасность и обновления319
9.5.7. Список решений320
Итоги.321
Глава 10. Целостность и атомарность в распределенных системах323
10.1 Источники данных с доставкой «не менее одного раза»324
10.1.1. Трафик между сервисами с одним узлом324
10.1.2. Повтор попытки вызова326
10.1.3. Производство данных и идемпотентность327
10.1.4. Паттерн CQRS.330
10.2. Наивная реализация дедупликации332

10.3. Типичные ошибки при реализации дедупликации в распределенных системах	335
10.3.1. Одноузловый контекст	336
10.3.2. Многоузловый контекст.	338
10.4. Обеспечение атомарности логики для предотвращения ситуации гонки.	341
Итоги.	345
Глава 11. Семантика доставки в распределенных системах	347
11.1. Архитектура событийно-управляемых приложений	348
11.2. Производители и потребители на базе Apache Kafka.	352
11.2.1. Kafka на стороне потребителя	353
11.2.2. Конфигурация брокеров Kafka	355
11.3. Логика производителя.	356
11.3.1. Выбор между целостностью данных и доступностью для производителя	359
11.4. Код потребителя и разные семантики доставки	362
11.4.1. Ручная фиксация у потребителя	364
11.4.2. Перезапуск от самых ранних или поздних смещений	366
11.4.3. Семантика «фактически ровно один»	369
11.5. Использование гарантий доставки для обеспечения отказоустойчивости.	371
Итоги.	373
Глава 12. Управление версиями и совместимостью.	374
12.1. Версионирование на абстрактном уровне.	375
12.1.1. Свойства версий.	375
12.1.2. Обратная и прямая совместимость	377
12.1.3. Семантическое версионирование	378
12.1.4. Маркетинговые версии	381
12.2. Версионирование для библиотек	381
12.2.1. Совместимость уровня исходного кода, двоичная и семантическая	382
12.2.2. Графы зависимостей и ромбовидные зависимости	391

12.2.3. Снижение последствий от критических изменений	397
12.2.4. Управление библиотеками только для внутреннего пользования.	402
12.3. Версионирование для сетевых API.	403
12.3.1. Контекст вызовов сетевых API	404
12.3.2. Ясность для клиента	405
12.3.3. Популярные стратегии версионирования.	407
12.3.4. Дополнительные аспекты версионирования	413
12.4. Версионирование для хранилищ данных	417
12.4.1. Краткое введение в Protocol Buffers	418
12.4.2. Что является критическим изменением?	420
12.4.3. Миграция данных в системе хранения.	421
12.4.4. В ожидании неожиданного	425
12.4.5. Разделение API и представлений хранения данных	427
12.4.6. Оценка форматов хранения.	430
Итоги.	431

Глава 13. Современные тенденции разработки и затраты на сопровождение кода.	433
13.1. Когда использовать фреймворки внедрения зависимостей.	434
13.1.1. Самостоятельная реализация внедрения зависимостей	435
13.1.2. Использование фреймворка внедрения зависимостей.	438
13.2. Когда применяется реактивное программирование	441
13.2.1. Создание однопоточной обработки с блокированием	442
13.2.2. Использование CompletableFuture	444
13.2.3. Реализация реактивного решения.	447
13.3. Когда применяется функциональное программирование.	449
13.3.1. Создание функционального кода на нефункциональном языке	450
13.3.2. Хвостовая рекурсия	453
13.3.3. Использование неизменяемости	454
13.4. Отложенное и немедленное вычисление	456
Итоги.	459