образование ПОЛИТЕХ одноклассники экосистема

Логирование. Неблокирующий IO

Галкин Александр Сергеевич

2022 г.

Оглавление

Logging

Логгирование

- Без логирования не обходится ни одна программа
- Что нам необходимо знать о работе программы?
  - Что пошло не так, в момент сбоя программы
  - Что привело к некорректному поведению
  - Какие запросы заставляют программу “тормозить”
  - Какие запросы происходят чаще всего (статистика)
  - Как вообще используется наша программа

Логирование. Антипаттерн

- Всю информацию можно записывать в стандартные потоки вывода
- Ошибки в System.err
- Отладочную информацию в System.out
- Минусы такого подхода
    - Нет гибкости настройки вывода информации
    - 90% ресурсов программы может быть занята на запись информации
    - Много условных операторов для разных режимов запуска

Логирование. Антипаттерн

java.util.logging.Logger

- ***void log(Level level, String msg)*** — основной метод логирования некоторого сообщения с заданным уровнем
- Уровни логирования **Level**:
- ***SEVERE*** — серьезные ошибки
- ***WARNING*** — предупреждения (что-то не совсем в порядке)
- ***INFO*** — основной уровень выполнения программы
- ***CONFIG*** — логирование конфигураци
- ***FINE***, ***FINER***, ***FINEST*** — отладочное логирование

java.util.logging.Logger

- На каждый уровень логирования, ***log*** имеет свои методы, например для уровня ***WARNING*** есть метод
- ***void warning(String msg)***
- На весь ***log*** можно задать уровень логирования или через метод ***void setLevel(Level newLevel)*** или с помощью конфигурации (сообщения уровня ниже заданного, логироваться не будут)

Logger. Шаблон использования

- Обычно создается один логгер на один класс:
  private static final Logger log = Logger.getLogger(LoggerExample.class.getName());
- Для кода выше создастся логгер с именем
  ru.mail.polis.course.classwork.iostreams.log.LoggerExample
- По сути, создастся 8 логгеров начиная от пустого и “***ru***” до итогового
- Все сообщения в ***log*** будут пытаться записаться и во все логгеры выше уровнем

Logger. Аргументы в сообщениях

- Конкатенация

  log.log(Level.INFO, "method arguments with arg1 = " + first + ", arg2 = " + second);

- Специальный метод

  log.log(Level.INFO, "method arguments with arg1 = {0}, arg2 = {1}", new Object[] {first, second});

- Для исключений спецсимволы подстановки не нужны:

  log.log(Level.SEVERE, "Exception", new NullPointerException());

java.util.logging.Handler

- Логгер не сам решает, как именно логировать сообщение
- Класс ***Handler*** это обработчик сообщения, который решает куда будет писаться сообщение
- ***java.util.logging.ConsoleHandler***
- ***java.util.logging.FileHandler***
- ***java.util.logging.SocketHandler***
- Обработчик задается или через конфигурацию или через метод ***void addHandler(Handler handler)***
- Можно добавить свой обработчик, если существующих не хватает

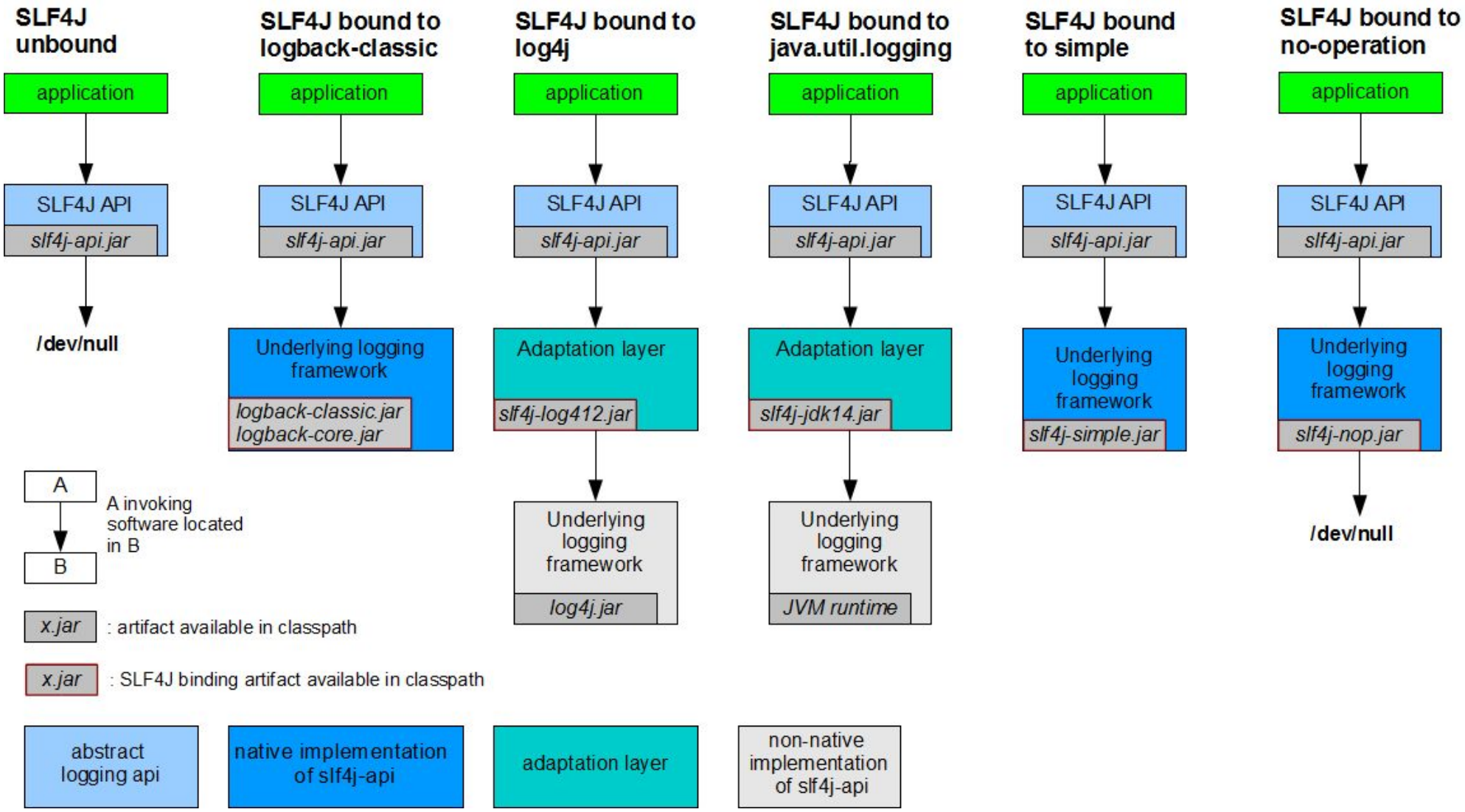
java.util.logging.Formatter

- Класс, который отвечает в каком формате сообщение записывается в лог
- Сначала сообщение преобразуется в нужный формат, а потом уже пишется в консоль, файл или передается по сети
- В Java два типа форматтеров
- ***java.util.logging.SimpleFormatter*** — человеко читаемый вид
- ***java.util.logging.XMLFormatter*** — машинно читаемый вид
- Можно добавить свой форматтер, если существующих не хватает

slf4j. Зависимости

- Simple Logging Facade for Java
- Основная библиотека: ***compile group: 'org.slf4j', name: 'slf4j-api', version: '1.7.29'***
- Простейшая реализация: ***compile group: 'org.slf4j', name: 'slf4j-simple', version: '1.7.29'***
- Классическая реализация: ***compile group: 'ch.qos.logback', name: 'logback-classic', version: '1.2.3'***
- Имеются реализации для всех основных библиотек, например: ***compile group: 'org.slf4j', name: 'slf4j-log4j12', version: '1.7.29'***

slf4j. Введение



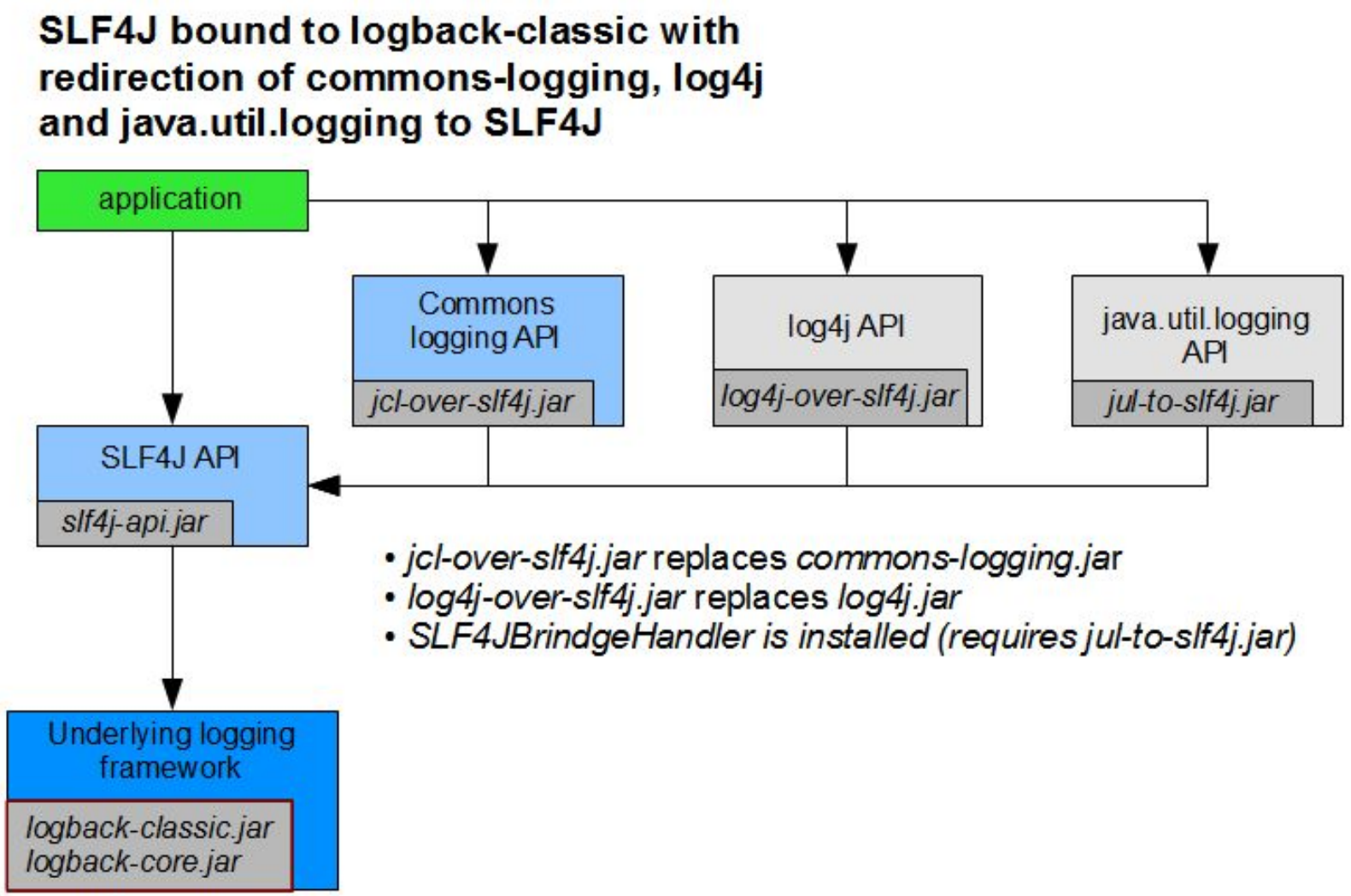
slf4j. Уровни логирования

- ***ERROR*** — серьезные ошибки
- ***WARNING*** — предупреждения (что-то не совсем в порядке)
- ***INFO*** — основной уровень выполнения программы
- ***DEBUG, TRACE*** — отладочное логирование

slf4j. Отличия

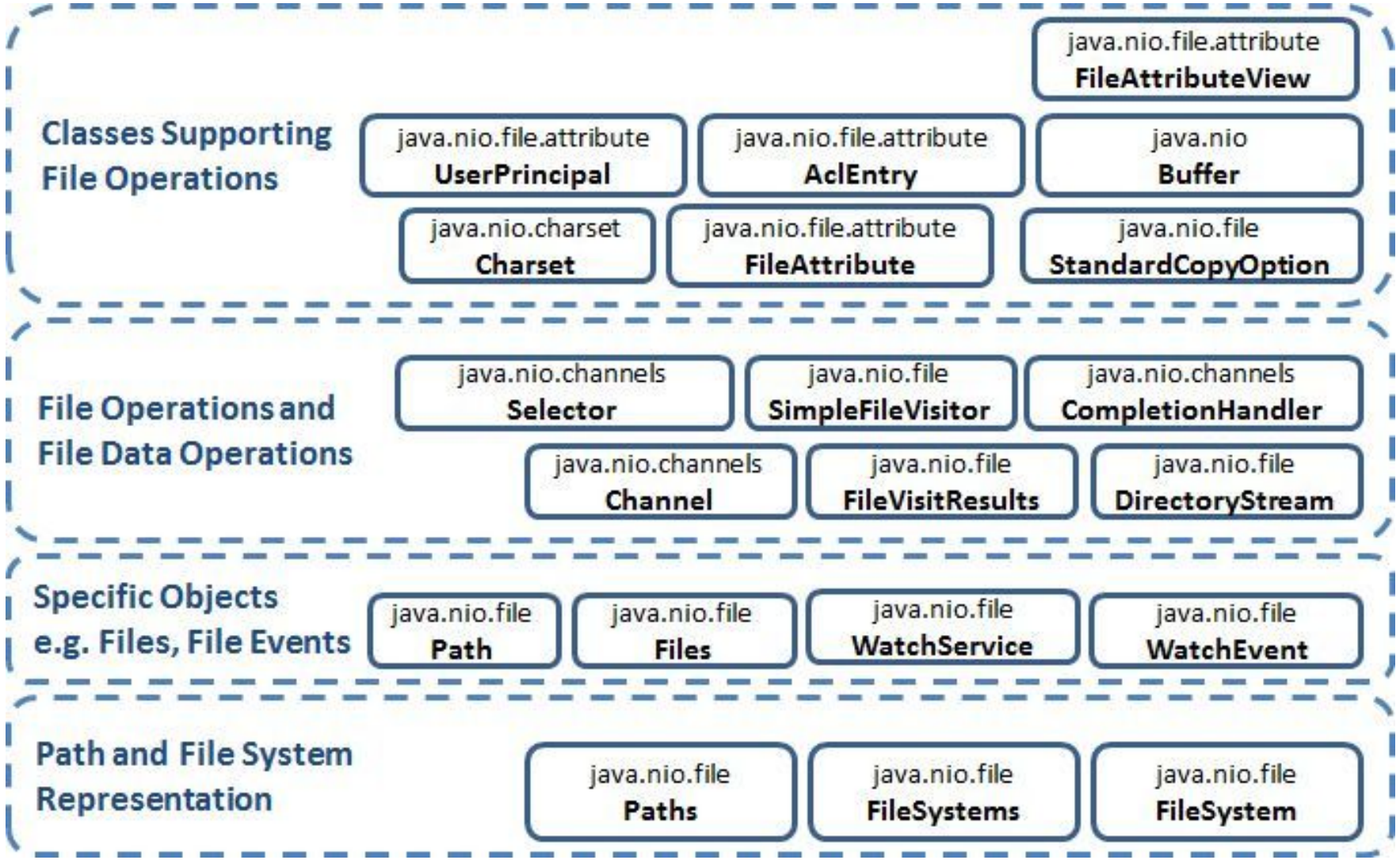
- Создание:
  private static final Logger log = LoggerFactory.getLogger(Slf4jExample.class);
- Логирование:
  log.info("method arguments with arg1 = {}, arg2 = {}", first, second);
- ***Handler*** -> ***Appender***
- ***Formatter*** -> ***Layout***
- Есть фильтры

slf4j. Bridging



Java NIO

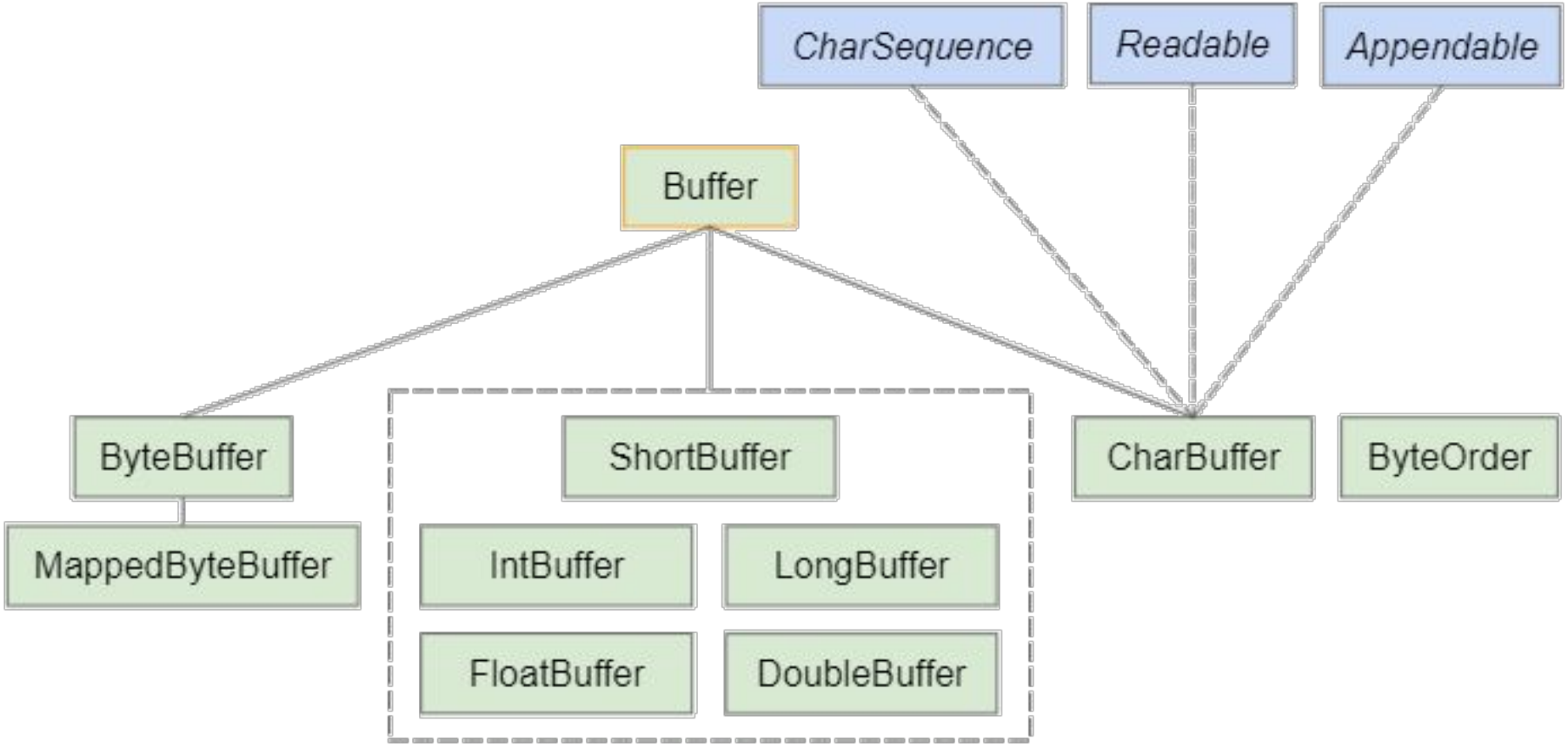
New Input Output



Что не так с блокирующим IO

- Проблемный класс ***File***
- Отсутствие кодировок
- Блокировка потока при чтении
- Отсутствие удобной буферизации

Buffer

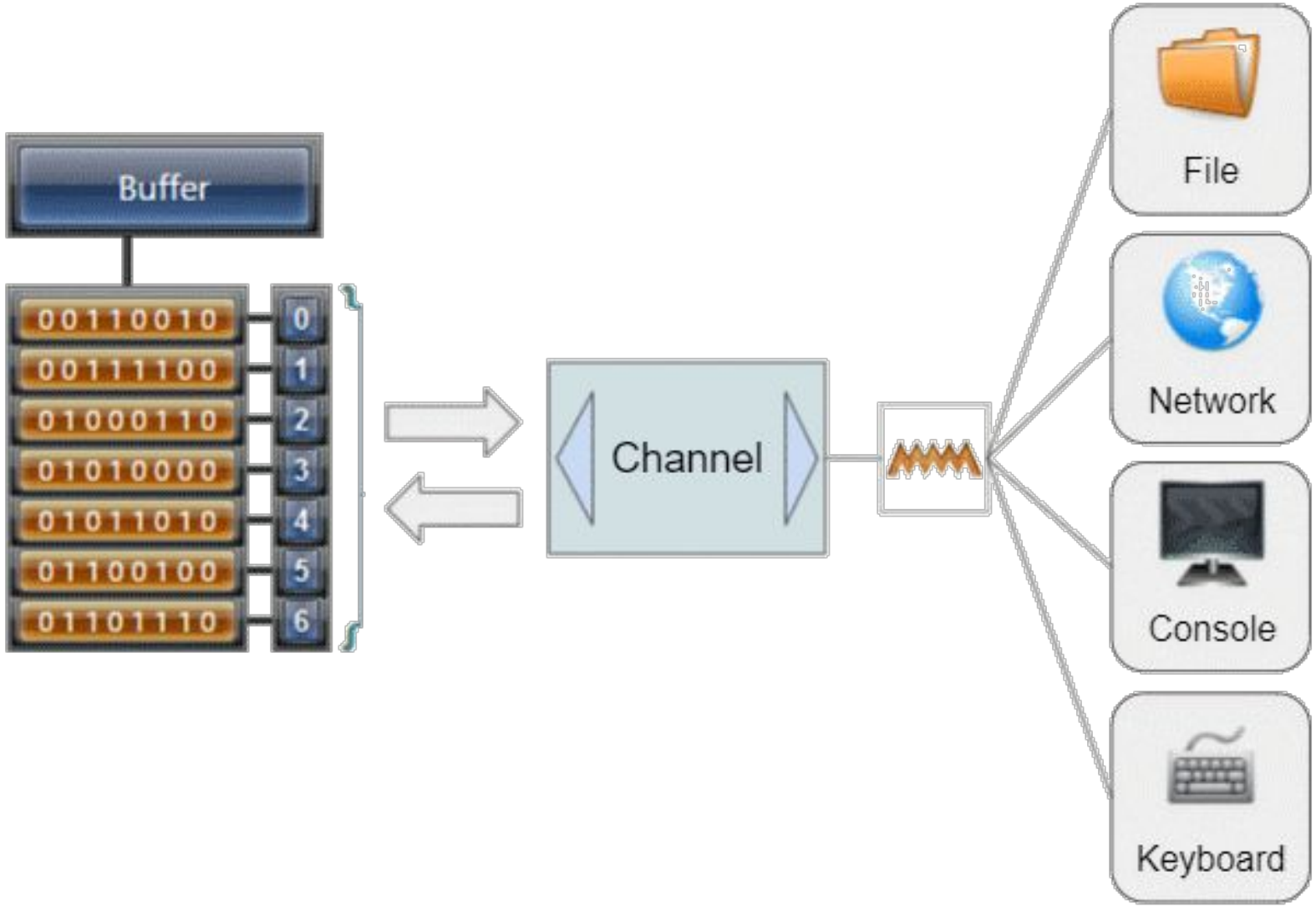


- В буфер можно писать
- Из буфера можно читать

Buffer. Основные методы

- **Capacity**, **limit**, **position** и **mark** — основные указатели буфера
- `Buffer clear()` — сбрасываем все указатели, данные не трогаем
- `Buffer flip()` — **limit -> position**, **position** -> 0
  Используется обычно после завершения записи
- `Buffer rewind()` — Перематывает буфер
  position -> 0, **mark** сбрасывается
- `int remaining()` — количество элементов между **position** и **limit - 1**
- `CharBuffer allocate(int capacity)`, `ByteBuffer allocateDirect(int capacity)` — в каждом буфере есть свой фабричный метод по его созданию

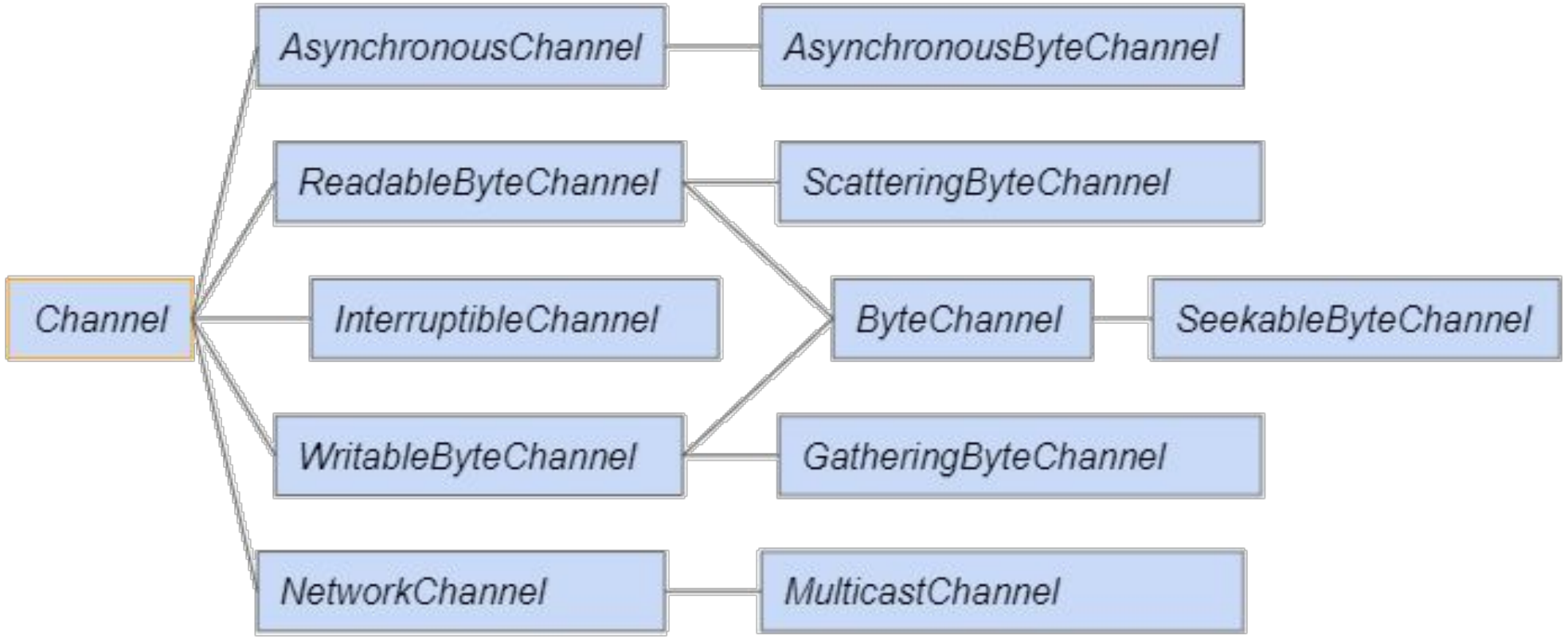
Channel


Buffer
00110010
00111100
01000110
01010000
01011010
01100100
01101110
0
1
2
3
4
5
6
Channel
File
Network
Console
Keyboard

Channel vs Stream

- В ***Channel*** можно писать и из него можно читать.
- В ***Stream*** для чтения и записи существуют отдельные классы
- ***Channel*** может считываться и записываться асинхронно.
- В ***Channel*** вы в основном манипулируете с буфером, а в ***Stream*** — непосредственно со стримом

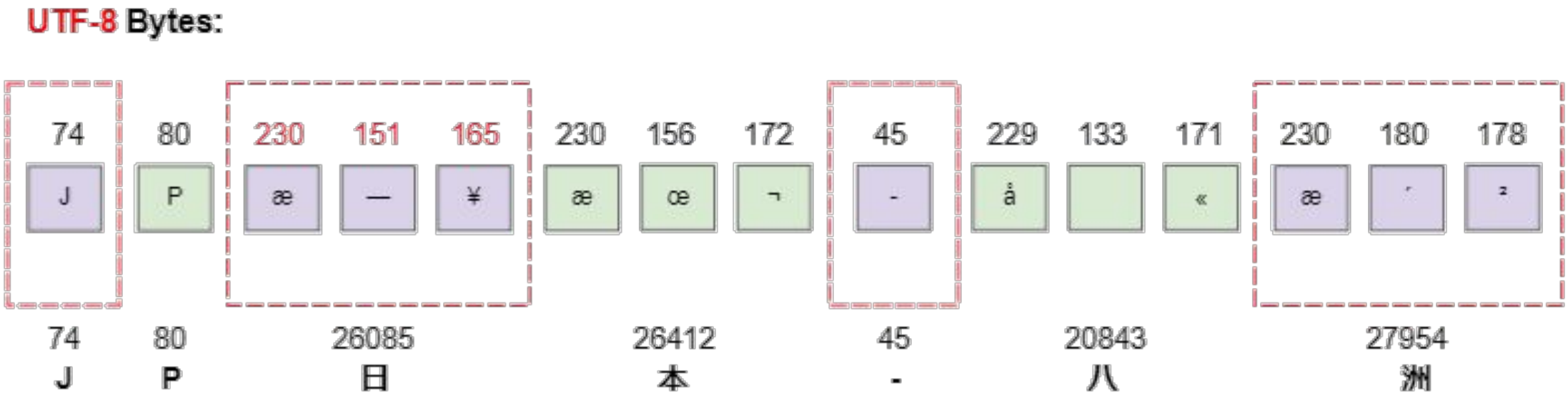
Channel. Иерархия



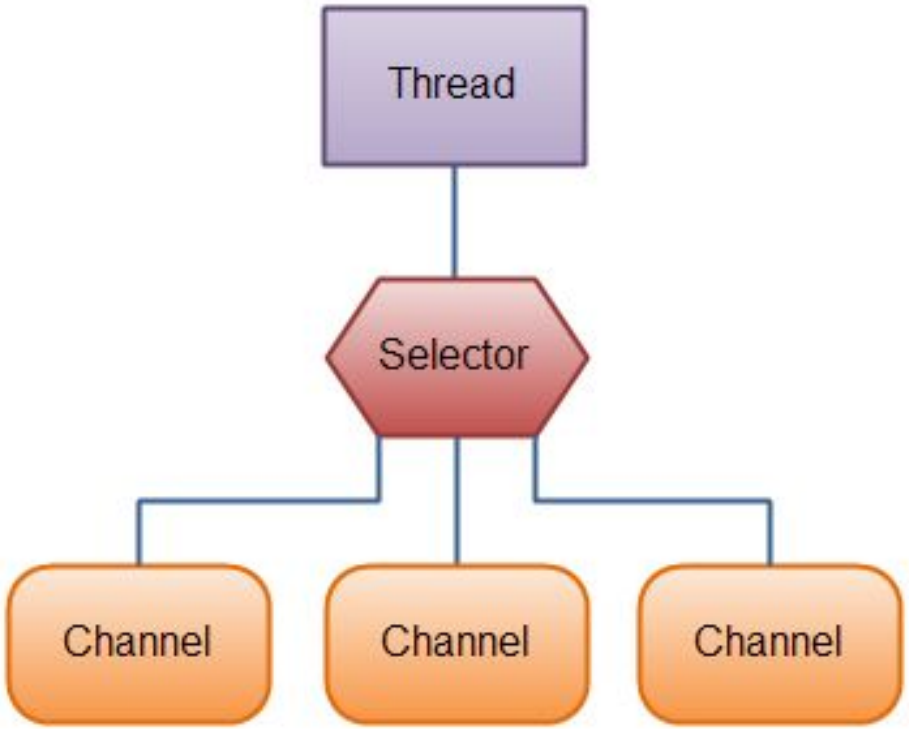
Channel. Основные методы

- `boolean isOpen()` — открыт ли канал
- `int read(ByteBuffer dst)` — читает данные из канала
- `long read(ByteBuffer[] dsts, int offset, int length)` — читает данные в несколько буфферов
- `int write(ByteBuffer src)` — пишет данные в канал
- `long write(ByteBuffer[] srcs, int offset, int length)` — пишет данные в канал из нескольких буфферов

Channel. Примеры



Selector



Path. Доступ к файловой системе

- Регистрация неблокирующий каналов на заданное событие.
  Класс `SelectionKey`
  - `static final int OP_READ = 1 << 0;`
  - `static final int OP_WRITE = 1 << 2;`
  - `static final int OP_CONNECT = 1 << 3;`
  - `static final int OP_ACCEPT = 1 << 4;`
- `int select()` — возвращает количество каналов, готовых к работе
- `Set<SelectionKey> selectedKeys()` — возвращает список ключей, готовых к работе

Спасибо!

образование

ПОЛИТЕХ

одноклассники
экосистема