CompletableFuture

уже здесь

Дмитрий Чуйко
dmitry.chuyko@oracle.com

Содержание

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Введение

Параллелизм в Java: История

- Threads
  - Monitors
  - Locks and other primitives
- Executors
  - Runnable<T>, Callable<T> → lambdas
- Future<T>
- ForkJoinPool
- stream.parallel()
- Explicit → implicit

Параллелизм в Java: Future<T>

- Обёртка для результата, доступного в будущем
- Получение значения требует проверки исключений
  - ExecutionException содержит исходное unchecked исключение
- Исполняется в каком-то потоке и блокирует какой-то (другой) поток
- Можно проверить, доступно ли уже значение
- Можно отменить вычисление

```
Future f = executor.submit(() -> ...);
result = f.get();
```

Сложности: Обработка ошибок

```
try {
    try {
        return parse(fileObject, content.get());
    } catch (ExecutionException e) {
        unfold(e);
    }
} catch (IOException e) {
    log.error(e, fileObject);
} catch (InterruptedException e) {
    throw new RuntimeException(e);
}
```

Сложности: Обработка ошибок

```
private static void unfold(ExecutionException e) throws IOException,
        InterruptedException {
    Throwable cause = e.getCause();
    if (cause instanceof ExecutionException)
        unfold((ExecutionException) cause);
    if (cause instanceof IOException)
        throw (IOException) cause;
    if (cause instanceof InterruptedException)
        throw (InterruptedException) cause;
    if (cause instanceof Error)
        throw (Error) cause;
    if (cause instanceof RuntimeException)
        throw (RuntimeException) cause;
    throw new RuntimeException(e);
}
```

Сложности: Давайте добавим обработчики!

```
readFileAsync(file,
    content -> System.out.println(content);
    ex -> ex.printStackTrace();
);
```

Сложности: Давайте добавим обработчики!

```
readFileAsync(file,
    content -> System.out.println(content);
    ex -> ex.printStackTrace();
);
```

- ↓Добро пожаловать сюда↓
  http://callbackhell.com/

```
readFileAsync(file,
    content -> processContentAsync(content,
        c -> System.out.println(c),
        e -> e.printStackTrace()
    );
    ex -> ex.printStackTrace();
);
```

CompletableFuture: Боремся с вложенностью

```
CompletableFuture.supplyAsync(() -> readFile(file))
    .thenComposeAsync(content -> processContent(content))
    .whenComplete((result, ex) -> {
        if (ex == null) {
            System.out.println(result);
        } else {
            ex.printStackTrace();
        }
    });
```

CompletableFuture: Композиция

- Method chaining
  - Нет блокирующих вызовов
  - Знакомо и удобно (builders, mocks, streams)
- Работаем с несколькими CF

API

Что за класс: CompletableFuture<T>

- Core library class since Java SE 8
- implements Future<T>
- implements CompletionStage<T>
  - Элемент композиции
  - Вычисление функции
  - Связь через результаты
  - Методы преобразований
  - toCompletableFuture()

Методы: Создание

- CompletableFuture()
  - boolean complete(T value), успешно только для одного потока
- static <U> CompletableFuture<U> completedFuture(U value)
- static <U> CompletableFuture<U>
  supplyAsync(Supplier<U> supplier[, Executor executor])
- static CompletableFuture<Void>
  runAsync(Runnable runnable[, Executor executor])
- ForkJoinPool.commonPool() по умолчанию

Методы: Создание

```
CompletableFuture<Long> cf1 = CompletableFuture.supplyAsync(() -> 42L);
CompletableFuture<Long> start = CompletableFuture.completedFuture(42L);
```

Методы: *Отступление

Упрощённая запись generics

- Function<? super T,? extends U>
  → Function<T,U>
- Consumer<? super T> → Consumer<T>

Методы: Трансформации

- <U> CompletableFuture<U> thenApply(Function<T,U> fn)
  - Нет "Async"→ продолжаем в том же потоке
- <U> CompletableFuture<U>
  thenApplyAsync(Function<T,U> fn [, Executor executor])

Методы: Трансформации (map)

```
CompletableFuture<Long> cf2 = CompletableFuture
.supplyAsync(() -> 42L)
.thenApply(r1 -> r1 + 2015);
```

Методы: Подписка

- CompletableFuture<Void> thenAccept(Consumer<T> block);
- CompletableFuture<Void> thenRun(Runnable action);
- Async versions

```
CompletableFuture<Void> cf3 = CompletableFuture
.supplyAsync(() -> 42L)
.thenApply(r1 -> r1 + 2015)
.thenAccept(System.out::println);
```

Методы: Обработка ошибок

- CompletableFuture<T>
  exceptionally(Function<Throwable, T> fn)
- CompletableFuture<U>
  handle(BiFunction<T, Throwable, U> fn)
- Ошибка пробрасывается по цепочкам

```
CompletableFuture.supplyAsync(() -> readFile(file))
    .thenComposeAsync(content -> processContent(content))
    .thenAccept(System.out::println)
    .exceptionally(Throwable::printStackTrace);
```

Методы: Комбинация (reduce)

- <U,V> CompletableFuture<V> thenCombine (CompletionStage<U> other, BiFunction<T,U,V> fn)

```
CompletableFuture<Long> cf2 = CompletableFuture
.supplyAsync(() -> 42L)
.thenCombine(CompletableFuture.supplyAsync(() -> 2015), Math::min);
```

Методы: Для набора

- CompletableFuture<**Object**>
  anyOf(CompletableFuture<?>... cfs)
- CompletableFuture<**Void**>
  allOf(CompletableFuture<?>... cfs)

Методы: Композиция (flatMap)

- CompletableFuture<U>
  thenCompose(Function<T,CompletableFuture<U>> fn)

```
CompletableFuture<Long> cff = CompletableFuture
    .supplyAsync(() -> 42L)
    .thenCompose(x -> CompletableFuture.supplyAsync(() -> x + 2015));

CompletableFuture<CompletableFuture<Long>> cff = CompletableFuture
    .supplyAsync(() -> 42L)
    .thenApply(x -> CompletableFuture.supplyAsync(() -> x + 2015));
```

Методы: Get

- T get() throws InterruptedException, ExecutionException
- T get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException
- T getNow(T valueIfAbsent)
- T join()
  - Нет checked exceptions (CompletionException)

```
...
stream.map(x -> CompletableFuture...).map(CompletableFuture::join)
...
```

Накладные расходы

Бенчмарк: Окружение

- Intel Core i5-3320M (1x2x2, 3.3 GHz)
- Linux x64 (kernel 3.16)
- JDK 8u40
- OpenJDK JMH 1.8

Бенчмарк: Базовые операции

```
@Param({ "1024" })
public volatile int loada;

@Param({ "1024" })
public volatile int loadb;

Integer a() {
    Blackhole.consumeCPU(loada);
    return loada;
}

Integer b() {
    Blackhole.consumeCPU(loadb);
    return loadb;
}
```

Бенчмарк: Что сравниваем

Простой вариант

```
@Benchmark
public Integer ab() {
    return a() * b();
}

@Benchmark
public Integer stream() throws InterruptedException, ExecutionException {
    return IntStream.range(0, 2)
        .mapToObj((i) -> i == 0 ? a() : b())
        .reduce(1, (a, b) -> a * b);
}

@Benchmark
public Integer parstream() throws InterruptedException, ExecutionException {
    return IntStream.range(0, 2).parallel()
        .mapToObj((i) -> i == 0 ? a() : b())
        .reduce(1, (a, b) -> a * b);
}
```

Бенчмарк: Что сравниваем

Future

```
@Benchmark
public Integer future2() throws InterruptedException, ExecutionException {
    ExecutorService fjp = ForkJoinPool.commonPool();
    Future<Integer> fa = fjp.submit(() -> a());
    Future<Integer> fb = fjp.submit(() -> b());
    return fa.get() * fb.get();
}

@Benchmark
public Integer future3() throws InterruptedException, ExecutionException {
    ExecutorService fjp = ForkJoinPool.commonPool();
    Future<Integer> fa = fjp.submit(() -> a());
    Future<Integer> fb = fjp.submit(() -> b());
    return fjp.submit(() -> fa.get() * fb.get()).get();
}
```

Бенчмарк: Что сравниваем

CompletableFuture

```
@Benchmark
public Integer cf2() throws InterruptedException, ExecutionException {
    CompletableFuture<Integer> cfa = CompletableFuture.supplyAsync(() -> a());
    CompletableFuture<Integer> cfb = CompletableFuture.supplyAsync(() -> b());
    return cfa.get() * cfb.get();
}

@Benchmark
public Integer cf3() throws InterruptedException, ExecutionException {
    return CompletableFuture.supplyAsync(() -> a())
        .thenCombine(CompletableFuture
            .supplyAsync(() -> b()), (a, b) -> a * b).get();
}
```

Бенчмарк: Результаты

Последовательная обработка лучше (loada=loadb=1024)

Вариант	μс/оп	
ab	4.7	± 0.1
cf2	7.9	± 0.8
cf3	11.4	± 0.7
future2	12.5	± 0.4
future3	13.3	± 0.7
parstream	9.9	± 0.5
stream	4.9	± 0.1

Бенчмарк: Результаты

Оценка пустой операции (loada=loadb=0)

Вариант	нс/оп	
ab	5.0	± 0.7
cf2	703.3	± 285.9
cf3	1647.2	± 523.2
future2	6315.4	± 632.8
future3	7591.9	± 1032.3
parstream	1234.4	± 202.8
stream	74.8	± 5.5

Бенчмарк: Результаты

Параллельная обработка лучше (loada=loadb=131070)

Вариант	μс/оп	
ab	596.5	± 2.3
cf2	360.0	± 17.3
cf3	342.6	± 13.5
future2	359.6	± 7.9
future3	342.5	± 9.8
parstream	334.8	± 18.4
stream	597.1	± 2.8

Пример. Rest-сервис

Сервис: Ингридиенты

- JDK 8u40
- JAX-RS 2.0
  https://jax-rs-spec.java.net/
- Jersey RI
  https://jersey.java.net/
- Grizzly NIO framework
  https://grizzly.java.net/

Сервис: Ингридиенты

- JDK 8u40
- JAX-RS 2.0
  https://jax-rs-spec.java.net/
- Jersey RI
  https://jersey.java.net/
- Grizzly NIO framework
  https://grizzly.java.net/
- CompletableFuture

Сервис: Запускалка

```
public class Main {
    public static final String BASE_URI = "http://localhost:8080/jpoint/";

    public static void main(String[] args) throws IOException {
     ResourceConfig rc = new ResourceConfig().packages("com.oracle.demo");
     HttpServer server = GrizzlyHttpServerFactory
             .createHttpServer(URI.create(BASE_URI), rc);
        System.out.println(String.format("Jersey␣app␣started␣"
                + "with␣WADL␣available␣at␣%sapplication.wadl\n"
                + "Hit␣enter␣to␣stop␣it...", BASE_URI));
        System.in.read();
        server.shutdownNow();
    }
}
```

Сервис: Endpoint

```
@Path("jpoint")
public class JPoint {
    @Inject
    DataService dataService;

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public void asyncGet(@Suspended final AsyncResponse asyncResponse) {
     dataService
     .findAudience()
     .thenCombine(dataService.findImpression(), (a, b) -> a + b)
     .thenApply(asyncResponse::resume)
            .exceptionally(asyncResponse::resume);

     // way #1
     asyncResponse.setTimeout(1, SECONDS);
        asyncResponse.setTimeoutHandler(ar -> ar.resume(new TimeoutException()));
    }
}
```

Сервис: Provider

Часть 1/2

```
@ManagedBean
@Path("data")
public class DataService {
    //@Inject
    ScheduledExecutorService shedPool = Executors.newScheduledThreadPool(1);

    public CompletableFuture<String> findAudience() {
        return find("audience");
    }

    public CompletableFuture<String> findImpression() {
        return find("impression");
    }

    @PreDestroy
    public void shutdown() {
        shedPool.shutdownNow();
    }
...
```

Сервис: Provider

Часть 2/2

```
public CompletableFuture<String> find(String path) {
    CompletableFuture<String> promise = new CompletableFuture<>();
    CompletableFuture
            .runAsync(() -> {
                try {
                    promise.complete(new String(Files.readAllBytes(Paths
                            .get(path))));
                } catch (IOException e) {
                    promise.completeExceptionally(e);
                }
            });
    // way #2
    shedPool.schedule(
            () -> promise.completeExceptionally(new TimeoutException()), 1,
            SECONDS);

    return promise;
}
```