Asynchronous CompletableFuture

László-Róbert Albert

May 18th, Bucharest

Asynchronous CompletableFuture

László-Róbert Albert

May 18th, Bucharest

About me

róbert

Chief Software Architect @ Crossover
10+ years in Java
Speaker
MsC in Computer Science
Proud father of 3

j.u.c.CompletableFuture

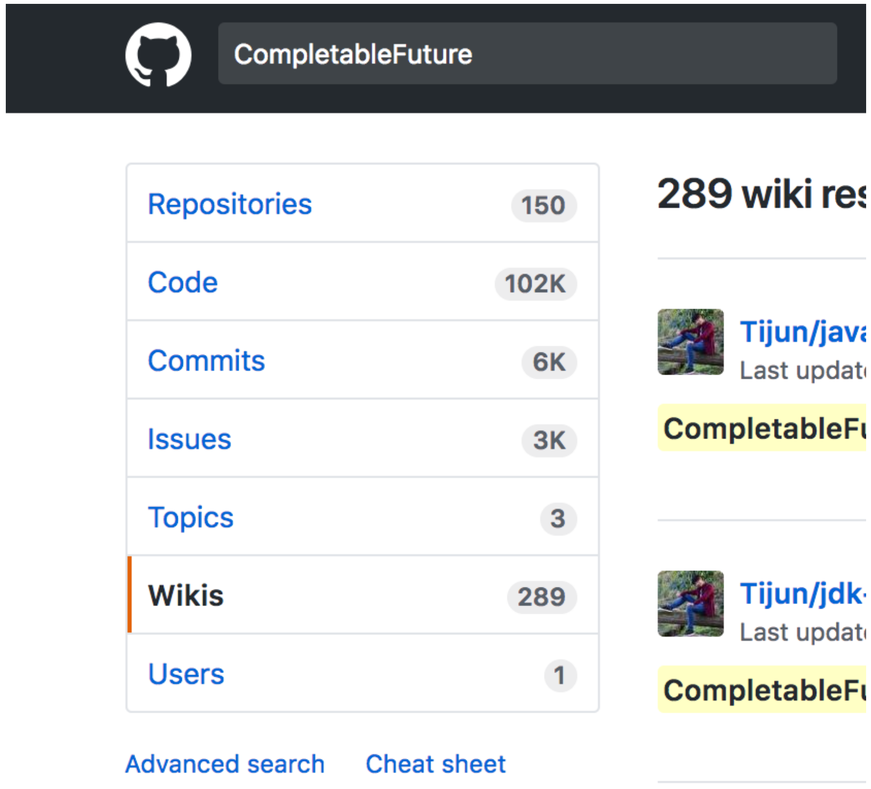
- @since Java 8

j.u.c.CompletableFuture

- @since Java 8
- not used in Java 8

j.u.c.CompletableFuture

- @since Java 8
- not used in Java 8



j.u.c.CompletableFuture

- Usage in Java 9

j.u.c.CompletableFuture

- Usage in Java 9
  - Process API
  - HttpClient

j.u.c.CompletableFuture

```
public class CompletableFuture<T> implements Future<T>, CompletionStage<T> {
    ...
}
```

j.u.c.Future

```
public interface Future<V> {
    boolean cancel(boolean mayInterruptIfRunning);
    boolean isCancelled();
    boolean isDone();
    V get() throws InterruptedException, ExecutionException;
    V get(long timeout, TimeUnit unit)
        throws InterruptedException, ExecutionException, TimeoutException;
}
```

j.u.c.CompletionStage

```
public interface CompletionStage<T> {
   public <U> CompletionStage<U> thenApply(Function<? super T,? extends U> fn);
   public <U> CompletionStage<U> thenApplyAsync(Function<? super T,? extends U> fn);
   public <U> CompletionStage<U> thenApplyAsync(Function<? super T,? extends U> fn, Executor executor);
   public CompletionStage<Void> thenAccept(Consumer<? super T> action);
   public CompletionStage<Void> thenAcceptAsync(Consumer<? super T> action);
   public CompletionStage<Void> thenAcceptAsync(Consumer<? super T> action, Executor executor);
   public CompletionStage<Void> thenRun(Runnable action);
   public CompletionStage<Void> thenRunAsync(Runnable action);
   public CompletionStage<Void> thenRunAsync(Runnable action, Executor executor);
   public <U,V> CompletionStage<V> thenCombine(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn);
   public <U,V> CompletionStage<V> thenCombineAsync(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn);
   public <U,V> CompletionStage<V> thenCombineAsync(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn, Executor executor);
   public <U> CompletionStage<Void> thenAcceptBoth(CompletionStage<? extends U> other, BiConsumer<? super T, ? super U> action);
   public <U> CompletionStage<Void> thenAcceptBothAsync(CompletionStage<? extends U> other, BiConsumer<? super T, ? super U> action);
   public <U> CompletionStage<Void> thenAcceptBothAsync(CompletionStage<? extends U> other, BiConsumer<? super T, ? super U> action, Executor executor);
   public CompletionStage<Void> runAfterBoth(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterBothAsync(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterBothAsync(CompletionStage<?> other, Runnable action, Executor executor);
   public <U> CompletionStage<U> applyToEither(CompletionStage<? extends T> other, Function<? super T, U> fn);
   public <U> CompletionStage<U> applyToEitherAsync(CompletionStage<? extends T> other, Function<? super T, U> fn);
   public <U> CompletionStage<U> applyToEitherAsync(CompletionStage<? extends T> other, Function<? super T, U> fn, Executor executor);
   public CompletionStage<Void> acceptEither(CompletionStage<? extends T> other, Consumer<? super T> action);
   public CompletionStage<Void> acceptEitherAsync(CompletionStage<? extends T> other, Consumer<? super T> action);
   public CompletionStage<Void> acceptEitherAsync(CompletionStage<? extends T> other, Consumer<? super T> action, Executor executor);
   public CompletionStage<Void> runAfterEither(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterEitherAsync(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterEitherAsync(CompletionStage<?> other, Runnable action, Executor executor);
   public <U> CompletionStage<U> thenCompose(Function<? super T, ? extends CompletionStage<U>> fn);
   public <U> CompletionStage<U> thenComposeAsync(Function<? super T, ? extends CompletionStage<U>> fn);
   public <U> CompletionStage<U> thenComposeAsync(Function<? super T, ? extends CompletionStage<U>> fn, Executor executor);
   public CompletionStage<T> exceptionally(Function<Throwable, ? extends T> fn);
   public CompletionStage<T> whenComplete(BiConsumer<? super T, ? super Throwable> action);
   public CompletionStage<T> whenCompleteAsync(BiConsumer<? super T, ? super Throwable> action);
   public CompletionStage<T> whenCompleteAsync(BiConsumer<? super T, ? super Throwable> action, Executor executor);
   public <U> CompletionStage<U> handle(BiFunction<? super T, Throwable, ? extends U> fn);
   public <U> CompletionStage<U> handleAsync(BiFunction<? super T, Throwable, ? extends U> fn);
   public <U> CompletionStage<U> handleAsync(BiFunction<? super T, Throwable, ? extends U> fn, Executor executor);
   public CompletableFuture<T> toCompletableFuture();
}
```

j.u.c.CompletionStage

```
public interface CompletionStage<T> {
   public <U> CompletionStage<U> thenApply(Function<? super T,? extends U> fn);
   public <U> CompletionStage<U> thenApplyAsync(Function<? super T,? extends U> fn);
   public <U> CompletionStage<U> thenApplyAsync(Function<? super T,? extends U> fn, Executor executor);
   public CompletionStage<Void> thenAccept(Consumer<? super T> action);
   public CompletionStage<Void> thenAcceptAsync(Consumer<? super T> action);
   public CompletionStage<Void> thenAcceptAsync(Consumer<? super T> action, Executor executor);
   public CompletionStage<Void> thenRun(Runnable action);
   public CompletionStage<Void> thenRunAsync(Runnable action);
   public CompletionStage<Void> thenRunAsync(Runnable action, Executor executor);
   public <U,V> CompletionStage<V> thenCombine(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn);
   public <U,V> CompletionStage<V> thenCombineAsync(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn);
   public <U,V> CompletionStage<V> thenCombineAsync(CompletionStage<? extends U> other, BiFunction<? super T,? super U,? extends V> fn, Executor executor);
   public <U> CompletionStage<Void> thenAcceptBoth(CompletionStage<? extends U> other, BiConsumer<? super T, ? super U> action);
   public <U> CompletionStage<Void> thenAcceptBothAsync(CompletionStage<? extends U> other, BiConsumer<? super T, ? super U> action);
   public <U> CompletionStage<Void> thenAcceptBothAsync(CompletionStage<? extends U> other, BiConsumer<? super T, ? super U> action, Executor executor);
   public CompletionStage<Void> runAfterBoth(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterBothAsync(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterBothAsync(CompletionStage<?> other, Runnable action, Executor executor);
   public <U> CompletionStage<U> applyToEither(CompletionStage<? extends T> other, Function<? super T, U> fn);
   public <U> CompletionStage<U> applyToEitherAsync(CompletionStage<? extends T> other, Function<? super T, U> fn);
   public <U> CompletionStage<U> applyToEitherAsync(CompletionStage<? extends T> other, Function<? super T, U> fn, Executor executor);
   public CompletionStage<Void> acceptEither(CompletionStage<? extends T> other, Consumer<? super T> action);
   public CompletionStage<Void> acceptEitherAsync(CompletionStage<? extends T> other, Consumer<? super T> action);
   public CompletionStage<Void> acceptEitherAsync(CompletionStage<? extends T> other, Consumer<? super T> action, Executor executor);
   public CompletionStage<Void> runAfterEither(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterEitherAsync(CompletionStage<?> other, Runnable action);
   public CompletionStage<Void> runAfterEitherAsync(CompletionStage<?> other, Runnable action, Executor executor);
   public <U> CompletionStage<U> thenCompose(Function<? super T, ? extends CompletionStage<U>> fn);
   public <U> CompletionStage<U> thenComposeAsync(Function<? super T, ? extends CompletionStage<U>> fn);
   public <U> CompletionStage<U> thenComposeAsync(Function<? super T, ? extends CompletionStage<U>> fn, Executor executor);
   public CompletionStage<T> exceptionally(Function<Throwable, ? extends T> fn);
   public CompletionStage<T> whenComplete(BiConsumer<? super T, ? super Throwable> action);
   public CompletionStage<T> whenCompleteAsync(BiConsumer<? super T, ? super Throwable> action);
   public CompletionStage<T> whenCompleteAsync(BiConsumer<? super T, ? super Throwable> action, Executor executor);
   public <U> CompletionStage<U> handle(BiFunction<? super T, Throwable, ? extends U> fn);
   public <U> CompletionStage<U> handleAsync(BiFunction<? super T, Throwable, ? extends U> fn);
   public <U> CompletionStage<U> handleAsync(BiFunction<? super T, Throwable, ? extends U> fn, Executor executor);
   public CompletableFuture<T> toCompletableFuture();
}
```

j.u.c.CompletionStage

- Contains 38 methods

j.u.c.CompletionStage

- Contains 38 methods
- 36 of them has 3 forms

j.u.c.CompletionStage

- Contains 38 methods
- 36 of them has 3 forms:

```
public CompletionStage<?> somethingAsync(..., Executor executor);
public CompletionStage<?> somethingAsync(...);
public CompletionStage<?> something(...);
```

j.u.c.CompletionStage

```
public CompletionStage<?> somethingAsync(..., Executor executor);
```

- runs action chain in executor

j.u.c.CompletionStage

```
public CompletionStage<?> somethingAsync(..., Executor executor);
```

- runs action chain in executor

```
public CompletionStage<?> somethingAsync(...);
```

- Same as `somethingAsync(..., ForkJoinPool.commonPool())`

j.u.c.CompletionStage

```
public CompletionStage<?> somethingAsync(..., Executor executor);
```

- runs action chain in executor

```
public CompletionStage<?> somethingAsync(...);
```

- Same as `somethingAsync(..., ForkJoinPool.commonPool())`

```
public CompletionStage<?> something(...);
```

- default execution

j.u.c.CompletionStage

- 12 methods remain

j.u.c.CompletionStage

- 12 methods remain
- 9 of them has 3 forms

j.u.c.CompletionStage

- 12 methods remain
- 9 of them has 3 forms:

  `Apply` - function from input to *`R`*, result is a ***`CompletableFuture`***`<`*`R`*`>`

j.u.c.CompletionStage

- 12 methods remain
- 9 of them has 3 forms:

  `Apply` - function from input to *`R`*, result is a ***`CompletableFuture`***`<`*`R`*`>`

  `Accept` - consumer of input, result is a ***`CompletableFuture`***`<`**`Void`**`>`

j.u.c.CompletionStage

- 12 methods remain
- 9 of them has 3 forms:

  `Apply` - function from input to *`R`*, result is a ***`CompletableFuture`***`<`*`R`*`>`

  `Accept` - consumer of input, result is a ***`CompletableFuture`***`<`**`Void`**`>`

  `Run` - just execute a ***`Runnable`***, result is a ***`CompletableFuture`***`<`***`Void`***`>`

j.u.c.CompletionStage

- single input

```
thenApply, thenAccept, thenRun
```

j.u.c.CompletionStage

- single input

  `thenApply, thenAccept, thenRun`

- binary `<<or>>`

  `applyToEither, acceptEither, runAfterEither`

j.u.c.CompletionStage

- single input

  `thenApply, thenAccept, thenRun`

- binary `<<or>>`

  `applyToEither, acceptEither, runAfterEither`

- binary `<<and>>`

  `thenCombine, thenAcceptBoth, runAfterBoth`

j.u.c.CompletionStage

- 3 methods remained:

```
thenCompose

handle

whenComplete
```

j.u.c.CompletionStage

`thenCompose`

- Function from input to ***`CompletableFuture`***`<R>`, result ***`CompletableFuture`***`<R>`
- a.k.a `flatMap`

j.u.c.CompletionStage

`handle`

- Function from input and exception to `R`, result ***`CompletableFuture`***`<R>`

j.u.c.CompletionStage

`whenComplete`

- Consumer from input and exception
- Similar to **`Accept`** methods above
- Result - the same as input

j.u.c.CompletionStage

- 2 methods remained (doesn't have async versions):

```
public CompletionStage<T> exceptionally(Function<Throwable, ? extends T> fn);

public CompletableFuture<T> toCompletableFuture();
```

j.u.c.CompletableFuture

```java
public class CompletableFuture<T> implements Future<T>, CompletionStage<T> {
    public CompletableFuture() { .. }

    public static CompletableFuture<Void> allOf(CompletableFuture<?>... cfs) { .. }
    public static CompletableFuture<Object> anyOf(CompletableFuture<?>... cfs) { .. }
    public static <U> CompletableFuture<U> completedFuture(U value) { .. }
    public static CompletableFuture<Void> runAsync(Runnable runnable) { .. }
    public static CompletableFuture<Void> runAsync(Runnable runnable, Executor executor) { .. }
    public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) { .. }
    public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier, Executor executor) { .. }

    public boolean cancel(boolean mayInterruptIfRunning) { .. }
    public boolean complete(T value) { .. }
    public boolean completeExceptionally(Throwable ex) { .. }
    public CompletableFuture<T> exceptionally(Function<Throwable, ? extends T> fn) { .. }
    public T get() throws InterruptedException, ExecutionException { .. }
    public T get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException { .. }
    public T getNow(T valueIfAbsent) { .. }
    public int getNumberOfDependents() { .. }
    public boolean isCancelled() { .. }
    public boolean isCompletedExceptionally() { .. }
    public boolean isDone() { .. }
    public T join() { .. }
    public void obtrudeException(Throwable ex) { .. }
    public void obtrudeValue(T value) { .. }

    ...
}
```

j.u.c.CompletableFuture

```java
public class CompletableFuture<T> implements Future<T>, CompletionStage<T> {
    public CompletableFuture() { .. }

    public static CompletableFuture<Void> allOf(CompletableFuture<?>... cfs) { .. }
    public static CompletableFuture<Object> anyOf(CompletableFuture<?>... cfs) { .. }
    public static <U> CompletableFuture<U> completedFuture(U value) { .. }
    public static CompletableFuture<Void> runAsync(Runnable runnable) { .. }
    public static CompletableFuture<Void> runAsync(Runnable runnable, Executor executor) { .. }
    public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier) { .. }
    public static <U> CompletableFuture<U> supplyAsync(Supplier<U> supplier, Executor executor) { .. }

    public boolean cancel(boolean mayInterruptIfRunning) { .. }
    public boolean complete(T value) { .. }
    public boolean completeExceptionally(Throwable ex) { .. }
    public CompletableFuture<T> exceptionally(Function<Throwable, ? extends T> fn) { .. }
    public T get() throws InterruptedException, ExecutionException { .. }
    public T get(long timeout, TimeUnit unit) throws InterruptedException, ExecutionException, TimeoutException { .. }
    public T getNow(T valueIfAbsent) { .. }
    public int getNumberOfDependents() { .. }
    public boolean isCancelled() { .. }
    public boolean isCompletedExceptionally() { .. }
    public boolean isDone() { .. }
    public T join() { .. }
    public void obtrudeException(Throwable ex) { .. }
    public void obtrudeValue(T value) { .. }

    ...
}
```

j.u.c.CompletableFuture

- Contains 38 methods inherited from `CompletionStage`

j.u.c.CompletableFuture

- Contains 38 methods inherited from `CompletionStage`

  and

- 14 other instance methods
- 7 static methods

j.u.c.CompletableFuture

- 6 ways to complete future

```
complete/completeAsync/completeExceptionally

cancel

obtrudeValue/obtrudeException
```

j.u.c.CompletableFuture

- 4 ways to get value

`get/join`	- blocking
`get(timeout, timeUnit)`	- not so blocking
`getNow(valueIfAbsent)`	- non-blocking

j.u.c.CompletableFuture

- 3 ways to know status

```
isDone

isCompletedExceptionally

isCancelled
```

j.u.c.CompletableFuture

- 5 static methods to create future

```
completedFuture

runAsync(Runnable, Executor) -> CompletableFuture<Void>

supplyAsync(Supplier<U>, Executor) -> CompletableFuture<U>
```

j.u.c.CompletableFuture

- 2 static methods to chain futures

```
allOf(CompletableFuture<?>...) -> CompletableFuture<Void>

anyOf(CompletableFuture<?>...) -> CompletableFuture<Object>
```

Blocking or asynchronous?

Blocking or asynchronous?

- Blocking

```
R doSomething(...);
```

- Asynchronous:

```
CompletionStage<R> doSomethingAsync(..., Executor executor);
CompletionStage<R> doSomethingAsync(...);
```

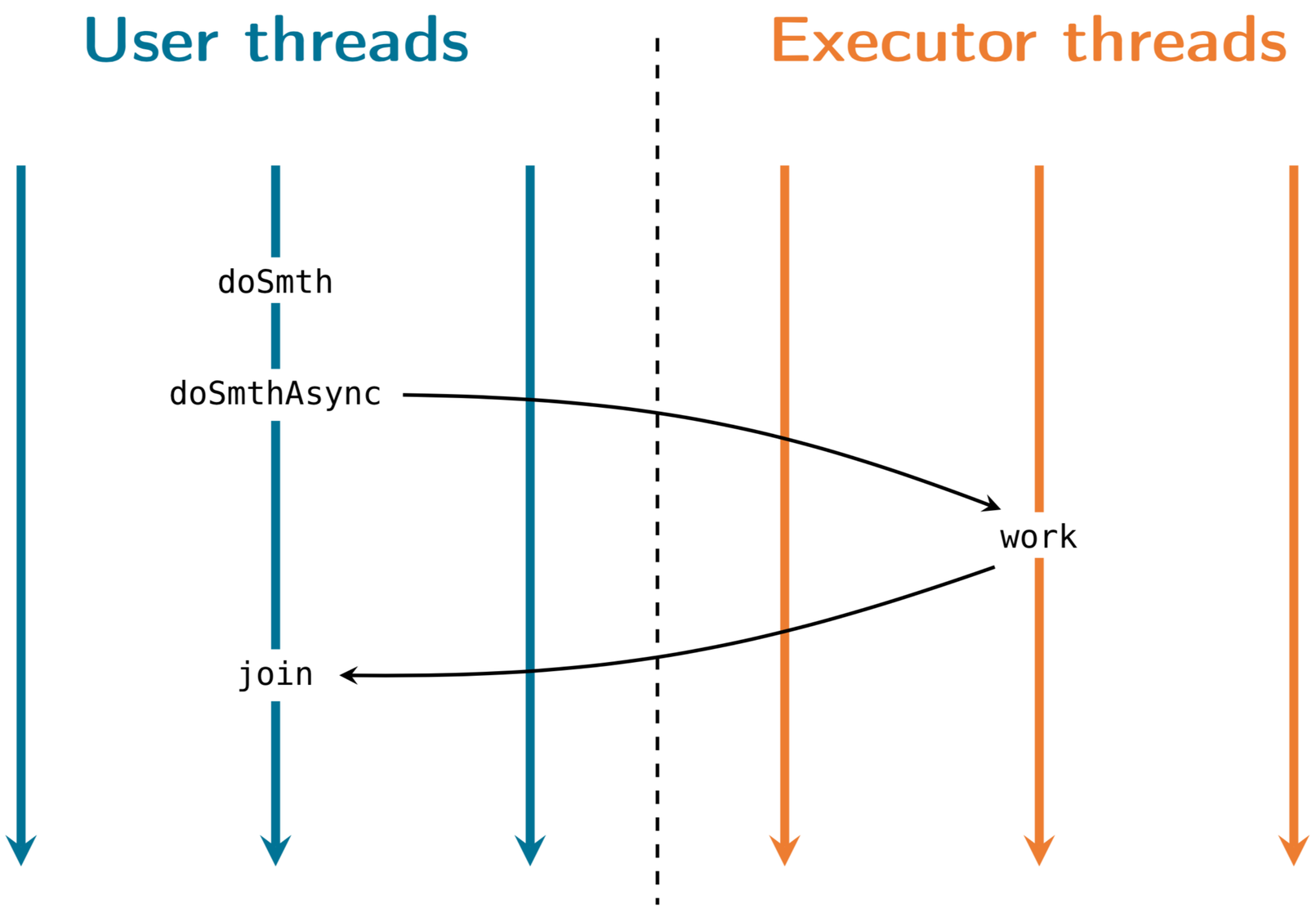
Blocking or asynchronous?

```
R doSmth(...) {
    return doSmthAsync(...).join();
}
```

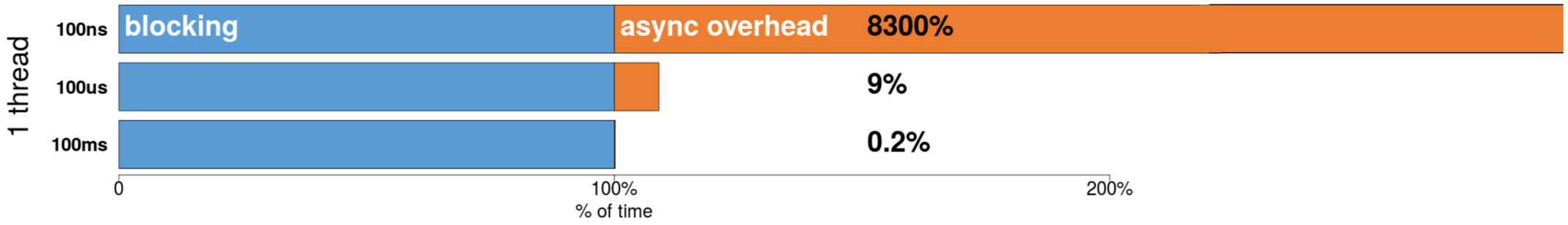
Blocking or asynchronous?

```
R doSmth(...) {
    return doSmthAsync(...).join();
}
```

Blocking or asynchronous?



Let's measure



Reference

- JavaDoc & the JDK
  - https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/CompletableFuture.html
  - HttpClient - http://openjdk.java.net/groups/net/httpclient/intro.html
    - https://docs.oracle.com/javase/9/docs/api/jdk/incubator/http/HttpClient.html
  - Process API - http://openjdk.java.net/jeps/102
  - practice 😀

Thank you

Asynchronous CompletableFuture

László-Róbert Albert

May 18th, Bucharest