

 You can now use the [Amazon S3 Transfer Manager \(Developer Preview\)](https://bit.ly/2WQebiP)  (<https://bit.ly/2WQebiP>) in the AWS SDK for Java 2.x for accelerated file transfers. Give it a try and [let us know what you think](https://bit.ly/3zT1YYM)  (<https://bit.ly/3zT1YYM>) !

Asynchronous programming

[PDF \(aws-sdk-java-dg-v2.pdf#asynchronous\)](#)

[Kindle \(https://www.amazon.com/dp/B07W6LPY3V\)](https://www.amazon.com/dp/B07W6LPY3V)


[RSS \(aws-sdk-java-dg-v2.rss\)](#)

The AWS SDK for Java 2.x features truly nonblocking asynchronous clients that implement high concurrency across a few threads. The AWS SDK for Java 1.x has asynchronous clients that are wrappers around a thread pool and blocking synchronous clients that don't provide the full benefit of nonblocking I/O.

Synchronous methods block your thread's execution until the client receives a response from the service. Asynchronous methods return immediately, giving control back to the calling thread without waiting for a response.

Because an asynchronous method returns before a response is available, you need a way to get the response when it's ready. The methods for asynchronous client in 2.x of the AWS SDK for Java return *CompletableFuture* objects that allow you to access the response when it's ready.

Non-streaming operations

For non-streaming operations, asynchronous method calls are similar to synchronous methods. However, the asynchronous methods in the AWS SDK for Java return a [CompletableFuture](#)  (<https://docs.oracle.com/javase/8/docs/api/index.html?java/util/concurrent/CompletableFuture.html>) object that contains the results of the asynchronous operation *in the future*.

Call the `CompletableFuture` `whenComplete()` method with an action to complete when the result is available. `CompletableFuture` implements the `Future` interface, so you can also get the response object by calling the `get()` method.

The following is an example of an asynchronous operation that calls a Amazon DynamoDB function to get a list of tables, receiving a `CompletableFuture` that can hold a [ListTablesResponse](http://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/services/dynamodb/model/ListTablesResponse.html) (<http://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/services/dynamodb/model/ListTablesResponse.html>) object. The action defined in the call to `whenComplete()` is done only when the asynchronous call is complete.

Imports

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import software.amazon.awssdk.services.dynamodb.model.ListTablesRequest;
import software.amazon.awssdk.services.dynamodb.model.ListTablesResponse;
import java.util.List;
import java.util.concurrent.CompletableFuture;
```

Code

```
public class DynamoDBAsyncListTables {

    public static void main(String[] args) throws InterruptedException {
```

```

// Create the DynamoDbAsyncClient object
Region region = Region.US_EAST_1;
DynamoDbAsyncClient client = DynamoDbAsyncClient.builder()
    .region(region)
    .build();

listTables(client);
}

public static void listTables(DynamoDbAsyncClient client) {

    CompletableFuture<ListTablesResponse> response =
client.listTables(ListTablesRequest.builder()
    .build());

    // Map the response to another CompletableFuture containing just the table
names
    CompletableFuture<List<String>> tableNames =
response.thenApply(ListTablesResponse::tableNames);

    // When future is complete (either successfully or in error) handle the
response
    tableNames.whenComplete((tables, err) -> {
        try {
            if (tables != null) {
                tables.forEach(System.out::println);
            } else {
                // Handle error
                err.printStackTrace();
            }
        } finally {
            // Lets the application shut down. Only close the client when you are
completely done with it.
            client.close();
        }
    });
    tableNames.join();
}
}

```

The following code example shows you how to retrieve an Item from a table by using the Asynchronous client. Invoke the `getItem` method of the `DynamoDbAsyncClient` and pass it a [GetItemRequest](http://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/services/dynamodb/model/GetItemRequest.html) (<http://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/services/dynamodb/model/GetItemRequest.html>) object with the table name and primary key value of the item you want. This is typically how you pass data that the operation requires. In this example, notice that a String value is passed.

Imports

```

import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.dynamodb.model.GetItemRequest;
import software.amazon.awssdk.services.dynamodb.model.AttributeValue;

```

```
import software.amazon.awssdk.services.dynamodb.DynamoDbAsyncClient;
import java.util.HashMap;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;
import software.amazon.awssdk.services.dynamodb.model.DynamoDbException;
```

Code

```
public static void getItem(DynamoDbAsyncClient client, String tableName, String
key, String keyVal) {

    HashMap<String, AttributeValue> keyToGet =
        new HashMap<String, AttributeValue>();

    keyToGet.put(key, AttributeValue.builder()
        .s(keyVal).build());

    try {

        // Create a GetItemRequest instance
        GetItemRequest request = GetItemRequest.builder()
            .key(keyToGet)
            .tableName(tableName)
            .build();

        // Invoke the DynamoDbAsyncClient object's getItem
        java.util.Collection<AttributeValue> returnedItem =
client.getItem(request).join().item().values();

        // Convert Set to Map
        Map<String, AttributeValue> map =
returnedItem.stream().collect(Collectors.toMap(AttributeValue::s, s->s));
        Set<String> keys = map.keySet();
        for (String sinKey : keys) {
            System.out.format("%s: %s\n", sinKey, map.get(sinKey).toString());
        }

    } catch (DynamoDbException e) {
        System.err.println(e.getMessage());
        System.exit(1);
    }
}
```

See the [complete example](https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/example_code/dynamodbasync/src/main/java/com/example/dynamodbasync/DynamoDBAsync) (https://github.com/awsdocs/aws-doc-sdk-examples/blob/master/javav2/example_code/dynamodbasync/src/main/java/com/example/dynamodbasync/DynamoDBAsync on GitHub).

Streaming operations

For streaming operations, you must provide an [AsyncRequestBody](http://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/core/async/AsyncRequestBody.html) (http://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/core/async/AsyncRequestBody.html) to provide the content incrementally, or an [<https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/asynchronous.html#basics-async-streaming>](http://docs.aws.amazon.com/sdk-for-</p>
</div>
<div data-bbox=)

[java/latest/reference/software/amazon/awssdk/core/async/AsyncResponseTransformer.html](https://docs.aws.amazon.com/sdk-for-java/latest/reference/software/amazon/awssdk/core/async/AsyncResponseTransformer.html)) to receive and process the response.

The following example uploads a file to Amazon S3 asynchronously by using the PutObject operation.

Imports

```
import software.amazon.awssdk.core.async.AsyncRequestBody;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.model.PutObjectRequest;
import software.amazon.awssdk.services.s3.model.PutObjectResponse;
import java.nio.file.Paths;
import java.util.concurrent.CompletableFuture;
```

Code

```
/**
 * To run this AWS code example, ensure that you have setup your development
 * environment, including your AWS credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class S3AsyncOps {

    public static void main(String[] args) {

        final String USAGE = "\n" +
            "Usage:\n" +
            "    S3AsyncOps <bucketName> <key> <path>\n\n" +
            "Where:\n" +
            "    bucketName - the name of the Amazon S3 bucket (for example,
bucket1). \n\n" +
            "    key - the name of the object (for example, book.pdf). \n" +
            "    path - the local path to the file (for example,
C:/AWS/book.pdf). \n" ;

        if (args.length != 3) {
            System.out.println(USAGE);
            System.exit(1);
        }

        String bucketName = args[0];
        String key = args[1];
        String path = args[2];

        Region region = Region.US_WEST_2;
        S3AsyncClient client = S3AsyncClient.builder()
            .region(region)
            .build();
```

```

PutObjectRequest objectRequest = PutObjectRequest.builder()
    .bucket(bucketName)
    .key(key)
    .build();

// Put the object into the bucket
CompletableFuture<PutObjectResponse> future = client.putObject(objectRequest,
    AsyncRequestBody.fromFile(Paths.get(path))
);
future.whenComplete((resp, err) -> {
    try {
        if (resp != null) {
            System.out.println("Object uploaded. Details: " + resp);
        } else {
            // Handle error
            err.printStackTrace();
        }
    } finally {
        // Only close the client when you are completely done with it
        client.close();
    }
});

future.join();
}
}

```

The following example gets a file from Amazon S3 asynchronously by using the `GetObject` operation.

Imports

```

import software.amazon.awssdk.core.async.AsyncResponseTransformer;
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.s3.S3AsyncClient;
import software.amazon.awssdk.services.s3.model.GetObjectRequest;
import software.amazon.awssdk.services.s3.model.GetObjectResponse;
import java.nio.file.Paths;
import java.util.concurrent.CompletableFuture;

```

Code

```

/**
 * To run this AWS code example, ensure that you have setup your development
 * environment, including your AWS credentials.
 *
 * For information, see this documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-started.html
 */

public class S3AsyncStreamOps {

```

```

public static void main(String[] args) {

    final String USAGE = "\n" +
        "Usage:\n" +
        "    S3AsyncStreamOps <bucketName> <objectKey> <path>\n\n" +
        "Where:\n" +
        "    bucketName - the name of the Amazon S3 bucket (for example,
bucket1). \n\n" +
        "    objectKey - the name of the object (for example, book.pdf). \n"
+
        "    path - the local path to the file (for example,
C:/AWS/book.pdf). \n" ;

    if (args.length != 3) {
        System.out.println(USAGE);
        System.exit(1);
    }

    String bucketName = args[0];
    String objectKey = args[1];
    String path = args[2];

    Region region = Region.US_WEST_2;
    S3AsyncClient client = S3AsyncClient.builder()
        .region(region)
        .build();

    GetObjectRequest objectRequest = GetObjectRequest.builder()
        .bucket(bucketName)
        .key(objectKey)
        .build();

    CompletableFuture<GetObjectResponse> futureGet =
client.getObject(objectRequest,
    AsyncResponseTransformerToFile(Paths.get(path)));

    futureGet.whenComplete((resp, err) -> {
        try {
            if (resp != null) {
                System.out.println("Object downloaded. Details: "+resp);
            } else {
                err.printStackTrace();
            }
        } finally {
            // Only close the client when you are completely done with it
            client.close();
        }
    });
    futureGet.join();
}
}

```

Advanced operations

The AWS SDK for Java 2.x uses [Netty](https://netty.io) (<https://netty.io>), an asynchronous event-driven network application framework, to handle I/O threads. The AWS SDK for Java 2.x creates an `ExecutorService` behind Netty, to complete the futures returned from the HTTP client request through to the Netty client. This abstraction reduces the risk of an application breaking the async process if developers choose to stop or sleep threads. By default, 50 Threads are generated for each asynchronous client, and managed in a queue within the `ExecutorService`.

Advanced users can specify their thread pool size when creating an asynchronous client using the following option when building.

Code

```
S3AsyncClient clientThread = S3AsyncClient.builder()
    .asyncConfiguration(
        b -> b.advancedOption(SdkAdvancedAsyncClientOption
            .FUTURE_COMPLETION_EXECUTOR,
            Executors.newFixedThreadPool(10)
        )
    )
    .build();
```

To optimize performance, you can manage your own thread pool executor, and include it when configuring your client.

```
ThreadPoolExecutor executor = new ThreadPoolExecutor(50, 50,
    10, TimeUnit.SECONDS,
    new LinkedBlockingQueue<>(10_000),
    new ThreadFactoryBuilder()
        .threadNamePrefix("sdk-async-response").build());
```

```
// Allow idle core threads to time out
executor.allowCoreThreadTimeOut(true);
```

```
S3AsyncClient clientThread = S3AsyncClient.builder()
    .asyncConfiguration(
        b -> b.advancedOption(SdkAdvancedAsyncClientOption
            .FUTURE_COMPLETION_EXECUTOR,
            executor
        )
    )
    .build();
```

If you prefer to not use a thread pool, at all, use `Runnable::run` instead of using a thread pool executor.

```
S3AsyncClient clientThread = S3AsyncClient.builder()
    .asyncConfiguration(
        b -> b.advancedOption(SdkAdvancedAsyncClientOption
            .FUTURE_COMPLETION_EXECUTOR,
            Runnable::run
        )
    )
    .build();
```

