ToiletPaper #147

Easy in-memory caching for AWS Lambda

Author: Robert Gruner / Software Engineer / Office Leipzig

✕ Problem

It is known that when an AWS Lambda function is called, a new instance is created, which is not immediately purged after the request has been processed. AWS keeps this runtime for a certain amount of time for reusing in later requests or calls. AWS recommends using the runtime's *execution environment* to make the Lambda function more performant. But how does it work?

✓ Solution

Put simply, in a Node.js Lambda there is only the handler function itself and a global JavaScript context. Each time the lambda is called, the handler function is executed – which makes caching within the function context impossible. However, this function has access to the global context. Any variables or objects can be stored in it. A subsequent call has access to this data as long as this runtime exists. If you combine this behavior with a dependency injection solution like tsyringe, you get instance caching for database/HTTP clients "for free". It is also very easy to cache static configurations in memory.

➔ Example

The following 2 variants are offered by tsyringe to cache the instance of a dependency in the global context. For class instances, the *singleton* decorator works well.

```
import { singleton } from "tsyringe";

@singleton()
class HttpClient {}
```

As the name suggests, the instance is created only once and is then stored in the dependency container. For simple objects or dependencies that are created using a factory, the helper function *instanceCachingFactory* is suitable.

```
import { container, instanceCachingFactory } from "tsyringe";

export interface CoffeeCache {
  configs?: CoffeeConfig[];
}

container.register("CoffeeCache", {
  useFactory: instanceCachingFactory<CoffeeCache>(() => ({
    /* "configs" key must be filled later */
  })),
});
```

With this setup, you can get the instance of the client e.g. by a **container.resolve(HttpClient)**. As this works everywhere in the code, it also does within a handler function. Once you have retrieved the configurations via HTTP, you can store them in the CoffeeCache object. If you call the Lambda function again, the configurations are already in the cache and you can save the HTTP request. Within other classes, the dependencies can also be used via constructor injection.

```
import { inject } from "tsyringe";

class CoffeeHandler {
  constructor(@inject("CoffeeCache") cache: CoffeeCache) {}
}
```

+ Further Aspects

- Documentation of AWS Lambda execution context: https://docs.aws.amazon.com/lambda/latest/dg/runtimes-context.html
- AWS Lambda best practices: https://docs.aws.amazon.com/lambda/latest/dg/best-practices.html
- Source on recommended DI solution *tsyringe:* https://github.com/microsoft/tsyringe