Async Tree Pattern

Guseyn Ismayylov

July 2018

1 Introduction

Async Tree Pattern is a declarative design pattern, the main purpose of which is to transform procedural code into the object-oriented(declarative) code in the asynchronous environment (in this work implementation of this pattern will be shown for Node.js).

Also **Async Tree** can be considered as a data structure.

2 Declarative vs Imperative

The conception of the **Async Tree** was created with assumption that declarative programming is much better than imperative style of writing code for big programs and systems. Declarative code is more readable, extensible and maintainable in general.

You may agree or disagree with this statement. It's only opinion of the author of this work.

Here it will be shown the difference between these two approaches by small example. Let's consider the following imperative code:

```
// Imperative
user = getUserFromDb(userId);
account = user.createNewAccount(accountInfo);
user.saveAccount(account);
```

It can be written in the declarative style:

```
// Declarative
SavedAccount(
  CreatedAccount(
    UserFromDb(userId), accountInfo
  )
).call();
```

The example is quite small to demonstrate why declarative programming is preferable. Nevertheless, you can see that declarative code always shows the result of program execution and what it requires to get this result. For readability it's more important to be able to see what result you get, not how exactly you get it.

3 Why Node.js

As it's been said the implementation of the **Async Tree** will be described for Node.js. The choice is very simple because Node.js is the most popular and stable asynchronous event driven runtime and it performs well. But as you will see, **Async Tree** can be applied not only in the asynchronous environment.

4 Async Tree and Async Object

Async Tree can be defined as a composition of objects and primitives. Each of objects in this composition that represents some other object or primitive type of data is called **Async Object**.

For example, in the previous example `SavedAccount` and `CreatedAccount` represent `Account`, `UserFromDb` represents `User`.

Async Object is an object that represents(computes) some other but similar(in terms of logic) object. So, for example, `SavedAccount`, `UpdatedAccount`, `DeletedAccount` are *async objects* because they represent `Account`, which is simple object. `UserFromDb`, `UserFromRequest` are also *async objects*, they represent simple object `User`.

Every **Async Object** can be constructed by other *async objects*, *simple objects* and *primitives*. For example, `SavedAccount` can be created by `UpdatedAccount`, `CreatedAccount` or any other `Async Object` that represents `Account`. Obviously, it can be created by `Account` itself.

That's the main idea of the **Async Tree Pattern**: to provide flexible way to create composition of objects via their representations.