Tests in ABP

基本框架

1. 使用的框架（按照abp文档推荐的）：
    - xUnit（整体测试框架）
    - NSubstitute（mock用的包）
    - Shouldly（assert用的包）建议升级到4.0.1，低版本可能有问题。
2. 函数attribute
    - 使用`[Fact]`attribute来声明一个测试函数
3. 函数命名
    - 函数命名需要传达的信息
        - 该函数是个测试函数
        - 该函数测试的功能函数
        - 该函数所期望的表现结果
    - 建议的函数命名
        - Test_{Function_name_to_be_tested}_{Success/Fail_For_Some_Reason}
        - Examples
            - `public void Test_FreezeConfiguration_Success()`
            - `public void Test_EndDataProcess_DataProcessFailed()`
            - `public async Task Test_DeleteFlow_Fail_For_ActiveFlow()`

三a注释

1. 在测试函数中使用三a注释，分别表示测试的三个阶段
    - `// arrange`（准备阶段，准备一些测试所需的数据结构，比如DTO、request等）
    - `// act`（执行阶段，一般为执行具体需要测试的函数）
    - `// assert`（断言阶段，一般为检查函数的执行结果是否符合预期）

```
[Fact]
public void Test_FreezeConfiguration_Success()
{
    // arrange
    var fakeFlow = Flow.Create(Guid.NewGuid(), Guid.NewGuid(), 1);
    fakeFlow.Freezed.ShouldBe(false);

    // act
    fakeFlow.FreezeConfiguration();

    // assert
    fakeFlow.Freezed.ShouldBe(true;
}
```

在Test类中使用internal的函数/类

1. 比如src/Flow.Domain中有一个internal函数

```
internal static Flow Create(Guid id, Guid flowFamilyId, int version)
{
    return new Flow(id, flowFamilyId, version);
}
```

2. 为了在test/Flow.Domain.Tests中调用它，需要在src/Flow.Domain的csproj中加入

```
    <ItemGroup>
        <AssemblyAttribute
Include="System.Runtime.CompilerServices.InternalsVisibleTo">
            <_Parameter1>Flow.Domain.Tests</_Parameter1>
        </AssemblyAttribute>
    </ItemGroup>
```

（PS:建议对csproj autoformat一下，如果format不对的话会报错一片红，但是很难发现是format导致的载入问题）

3. 如果我想访问private呢？
    - unable and unreasonable
    - private函数应该是被public调用的，外部使用的入口也是public。你应当通过public来调用private。

使用NSubstitute来Mock

1. 基本语法
    1. 创建

```
public class FlowManagerTest : FlowDomainTestBase
{
    private readonly IRepository<Flow, Guid> fakeFlowRepo;

    public FlowManagerTest()
    {
        this.fakeFlowRepo = Substitute.For<IRepository<Flow, Guid>>();
    }
}
```

    2. 拟定返回值（arrange阶段）

```
this.flowRepository.GetAsync(flow.Id).Returns(flow);
```

    如同调用它一样的语法，后续跟Returns来表示当参数为`flow.Id`时返回值为`flow`这个实例。

    3. 更复杂点的用法
        1. `ReturnsForAnyArgs`对于任何参数，返回固定的值。

2. `ReturnsNullForAnyArgs`对于任何参数，返回Null
3. 当使用包含AnyArgs的句式时，前面的函数参数应当是些啥都无所谓。建议使用`default`。
    - 若有同名重载函数时，使用`default(你期望的参数类型)`
4. example：
   `this.fakeFlowRepo.InsertAsync(default).ReturnsNullForAnyArgs();`

   释义：对于一切参数的`insertAsync`都返回Null。
5. 更多用法请参考文档

4. 检查调用次数（assert阶段）
    1. 使用`Received`/`ReceivedWithAnyArgs`来判断目标函数被调用次数是否符合预期
    2. ```
       // 任意参数InsertAsync被调用一次
       await this.fakeFlowFamilyRepo
                       .ReceivedWithAnyArgs(1)
                       .InsertAsync(default).ConfigureAwait(false);
       // 参数为flow时的InsertAsync被调用一次
        await this.fakeFlowRepo
                       .Received(1)
                       .InsertAsync(flow).ConfigureAwait(false);
       ```
    3. async的函数需要加上`await`与`ConfigureAwait(false)`

使用Shouldly来assert

1. 基本用法：

   `Something.ShouldBe(...)`
2. 常用用法

   ```
   general:
       ShouldBe
       ShouldBeNull
   Bool:
       ShouldBeTrue
       ShouldBeFalse
   集合类:
       ShouldBeEmpty
       ShouldBeEquivalentTo(集合相等)
   int:
       ShouldBeGreaterThan
   ```
3. 判断proto中的`RepeatedField<AlgoConfigDto>`与c#中的`List<AlgoConfig>`集合相等
    1. RepeatedField中包含的为Dto，而List中包含的是具体的类，因此需要先用mapper转换
    2. 集合相等不能直接使用`ShouldBe`，需要使用`ShouldBeEquivalentTo`
    3. example:

       ```
        this.objectMapper.Map<RepeatedField<AlgoConfigDto>, List<AlgoConfig>>
       (flowConfig.DataProcessAlgos)
                       .ShouldBeEquivalentTo(flow.DataProcessAlgos);
       ```
       ```

4. 更多用法参考文档

单测和集成测试

1. 单测（一般为Domain层）：仅针对具体的功能函数，一切外界因素均使用mock。
2. 集成测试（一般为application层）：针对业务功能入口函数，数据库使用真实的（mongo2Go），eventBus使用mock的，其他酌情考虑。
    - 如何使用真实的：`GetRequiredService<T>`
    - 为什么eventBus使用mock的？
        1. 我们业务侧保证调用到abp框架的publish函数即可，因此检查publish函数是否被调用即可，至于消息是否到达rabbitmq是abp框架的部分。
        2. 如果使用真实的，我们也难以观察到event是否真实发送到了rabbitmq中。

           （只能打开webUI看或者使用rabbitMQ的sdk来自己实现连接rabbitmq并调用函数，复杂度飙升）

Mongo2Go

1. ABP使用了mongo2Go，会为每个测试用例起一个临时的mongo实例，开在27xxx端口。
    - 每个test的mongo独立隔离，拥有同样的预置数据。
    - 临时实例访问不了，测试结束就没了。
2. 每个临时mongo2GO都会插入预置的Data Seed数据。
3. 如何强制tests 时使用本地自己起的mongo而不是mongo2GO的临时实例？
    - 在src/Flow.MongoDB/FlowMongoDbModule.cs的ConfigureServices函数中加入以下代码
    - ```
      // uncomment this will use local mongoDB
      /*this.Configure<AbpDbConnectionOptions>(options =>
      {
         options.ConnectionStrings[FlowDbProperties.ConnectionStringName] =
      "mongodb://localhost:27017/CTA";
      });
      */
      ```

设置Mongo的Guid类型

1. 在src/Flow.MongoDB/FlowMongoDbModule.cs中加入以下函数

```
 public override void
OnPreApplicationInitialization(ApplicationInitializationContext context)
        {
            // every test will run this function, which leads to duplicate
registration.
            // recreate BsonSerializer for tests use.
            // and use test/Flow.Application.Tests/xunit.runner.json &
test/Flow.Domain.Tests/xunit.runner.json to make tests parallel
            Type staticType = typeof(BsonSerializer);
            ConstructorInfo ci = staticType.TypeInitializer;
```
```

```
            object[] parameters = new object[0];
            ci.Invoke(null, parameters);

            // Enable UUID storing and parsing in/from mongodb as UUID
subtype 4.
            BsonDefaults.GuidRepresentationMode = GuidRepresentationMode.V3;
            BsonSerializer.RegisterSerializer(new
GuidSerializer(GuidRepresentation.Standard));
        }
```

- 真正起作用的只有最后两句，但是因为每个测试都会跑这个函数，而BsonSerializer类不允许duplicate registration，因此需要用点魔法。
- 为了不引发报错，需要
  - 使用发射在每个测试用例执行前销毁BsonSerializer类，然后又初始化它。（如同上述前四行）
  - 使tests顺序执行。在test/Flow.Domain.Tests与test/Flow.Application.Tests项目下加入文件`xunit.runner.json`
  - xunit.runner.json:

```
{
  "parallelizeAssembly": false,
  "parallelizeTestCollections": false
}
```

Test文件树

- test
  - /Flow.Application.Tests (application层测试)
    - /Flows（包含FlowsAppService的集成测试）
    - /EventHandler（包含eventbus的测试）
    - xunit.runner.json（xunit框架的配置文件，为了使测试顺序执行）
  - /Flow.Domain.Tests (domain层测试)
    - /Flows（包含src/Flow.Domain的测试）
    - xunit.runner.json（xunit框架的配置文件，为了使测试顺序执行）

预置测试数据

1. 位置：test/Flow.TestBase/FlowTestData.cs
2. 放什么？
   - 预置的测试数据，用来在data seed时被插入数据库。
3. **注意**：所有测试用例共享一个static类。
4. best practice:

```
public static class FlowTestData
{
    // 声明一些想插入的实例
```

```
        public static readonly FlowFamily FlowFamily1;
        public static readonly Flow Flow1;
        public static readonly Flow Flow2;

        public static readonly int FlowFamily1ChildrenNum = 2;

    // 准备一个ArrayList，用来给data seed函数调用
        public static readonly ArrayList FlowList = new ArrayList();
        public static readonly ArrayList FlowFamilyList = new ArrayList();

        static FlowTestData()
        {
            /*
            1. 初始化Flow1, Flow2等实例，并给他们配上期望的属性
            （具体方法下述）
            */

            // 2. 放入ArrayList，在data seed中调用，插入该arrayList中每一个实例到
repo。
            FlowList.Add(Flow1);
            FlowList.Add(Flow2);
        }
 }
```

使用反射来设置测试数据

1. 因为Flow类不是所有成员变量都有public setter，因此使用了反射来构造测试实例。
2. 使用一个工具函数

```
  public static class ReflectionSetter
    {
        public static bool SetValue(object targetObj, string fieldName,
object fieldValue)
        {
            PropertyInfo prop = targetObj.GetType().GetProperty(fieldName);
            if (prop != null)
            {
                if (prop.PropertyType.IsGenericType &&
prop.PropertyType.GetGenericTypeDefinition() == typeof(Nullable<>))
                {
                    dynamic objValue =
System.Activator.CreateInstance(prop.PropertyType);
                    objValue = fieldValue;
                    prop.SetValue(targetObj, (object)objValue, null);
                }
                else
                {
                    prop.SetValue(targetObj, fieldValue, null);
                }

                return true;
            }

            return false;
```

```
        }
    }
```

3. 创建Flow的Builder类，使用反射来赋值。（FlowCenter放在了test/Flow.TestBase/Builder下）

```
public class FlowBuilder
    {
        private Flow flow;

        private FlowBuilder()
        {

        }

     // New()初始化Builder内部的private flow实例
        public static FlowBuilder New()
        {
            var fb = new FlowBuilder();
            fb.flow = Flow.Create(Guid.NewGuid(), Guid.NewGuid(), 1);
            return fb;
        }

     /*
     assign函数，一般为使用上述的反射工具函数（ReflectionSetter）来设置成员flow实例的
属性，返回它本身以实现链式编程
     */
        public FlowBuilder AssignVersion(int version)
        {
            ReflectionSetter.SetValue(this.flow, "Version",
version).ShouldBeTrue();
            return this;
        }

        public FlowBuilder AssignFamilyId(Guid familyId)
        {
            ReflectionSetter.SetValue(this.flow, "FlowFamilyId",
familyId).ShouldBeTrue();
            return this;
        }




     // 将builder内的成员flow导出
        public Flow Gen()
        {
            return this.flow;
        }
    }
```

4. 在FlowTestData.cs中这样使用

```
        // Flow1
            Flow1 = FlowBuilder
                .New()
                .AssignFamilyId(flowFamily1Id)
```

```
                .AssignVersion(1)
                .AssignTrainDatasetId(Guid.NewGuid())
                .AssignTestDatasetId(Guid.NewGuid())
                .AssignOneDataProcessAlgo()
                .AssignDataProcessExperimentId()
                .AssignStep(Shared.FlowStep.RunFlow)
                .AssignOneModelAlgo(Shared.ModelTrainingStatus.Finished)
                .AssignOneModelAlgo(Shared.ModelTrainingStatus.Training)
                .AssignOneModelAlgo(Shared.ModelTrainingStatus.Stopped)
                .AssignOneModelAlgo(Shared.ModelTrainingStatus.Notstart)
                .AssignRunningStatus(Shared.FlowRunningStatus.Training)
                .AssignModelTrainFile("train_file")
                .AssignModelTestFile("test_file")
                .Gen();
            FlowList.Add(Flow1);
```

Data seed

1. 即在tests时，给每个测试用例起的临时mongo中加入的预置数据。
    - PS：它在integrate数据库时也会被调用。
2. 位置：tests/Flow.TestBase/FlowTestData.cs

```
 public class FlowDataSeedContributor : IDataSeedContributor,
ITransientDependency
{
        private readonly IGuidGenerator guidGenerator;
        private readonly IRepository<Flow, Guid> flowRepo;
        private readonly IRepository<FlowFamily, Guid> flowFamilyRepo;

     // 准备好repo
        public FlowDataSeedContributor(
            IGuidGenerator guidGenerator,
            IRepository<Flow, Guid> flowRepo,
            IRepository<FlowFamily, Guid> flowFamilyRepo)
        {
            this.guidGenerator = guidGenerator;
            this.flowRepo = flowRepo;
            this.flowFamilyRepo = flowFamilyRepo;
        }

        public async Task SeedAsync(DataSeedContext context)
        {
            // 理论上mongo2GO应该不需要这一句，但是使用自己的mongo时，每个tests都会调
用一次data
            // seed，所以需要这一句。
            if (await this.flowRepo.GetCountAsync().ConfigureAwait(false) >
0)
            {
                return;
            }
            /*
             在此为repo插入数据。每个tests可视的数据是独一份的。
             */
```

```
            // 这里我直接使用了FlowTestData中准备完毕的arrayList
             foreach (Flow f in FlowTestData.FlowList)
            {
                await this.flowRepo.InsertAsync(f).ConfigureAwait(false);
            }

            foreach (FlowFamily fm in FlowTestData.FlowFamilyList)
            {
                await
this.flowFamilyRepo.InsertAsync(fm).ConfigureAwait(false);
            }

        }
}
```

EventBus测试

- 发出event：使用mock类来检验eventBus.PublishAsync的调用次数。
- 接收event：接受event的函数在abp中表现为override一个abp内部的函数，因此可以像其他普通函数一样写测试。

将测试按照分散到多个文件中

- 在集成测试中，需要测试的函数很多，全挤在一个文件里看不下去。
- 解决方法：使用partial关键字来表示类的一部分。
- 文件树：
  - test/Flow.Application.Tests/Flows
    - FlowAppServiceTest.cs（仅包含private成员声明与构造函数）
    - FlowAppServiceTest_Create.cs（按照功能拆分）
    - FlowAppServiceTest_Update.cs
    - FlowAppServiceTest_Delete.cs
    - FlowAppServiceTest_Control.cs
- 每个文件中类的声明：

  `public partial class FlowAppServiceTest : FlowApplicationTestBase`

注意点：集成测试中因共享Static类导致的同步问题

1. 出现原因
   - 每个测试虽然有独立的mongodb，但是他们共享一个static类（FlowTestData），因此前一个测试修改了static类中的成员的话，很可能导致下一个测试失败。
2. 解决方法
   - 从repo中获取测试实例而不是直接使用TestData类中的。

```
Bad:
    var flow = FlowTestData.Flow1;

good:
    var flow = await
this.getFlowInstanceAsync(FlowTestData.Flow1.Id).ConfigureAwait(false);

// getFlowInstanceAsync是包装了一层repo.GetAsync
```

3. 当然，如果你保证该测试用例中不会修改实例的属性，那么直接使用Static中的成员也是ok的。