0 引言

测试驱动开发的基本思想是通过测试来推动整个软件开发流程,通过测试来明确软件需求、验证软件设计、保证软件质量。 在. net 开发平台上,目前有多套成熟的单元测试框架, 比如 MSTest, NUnit 和 XUnit 等。 在软件开发工作中,应用测试驱动开发技术可以提高软件开发效率,减少软件故障。

测试驱动开发的工作流程如下:首先,明确要完成的功能并整理出一张功能列表;其次,针对每个功能设计测试用例,测试用例应尽量简单;第三,编写功能代码,进行修改调试,以便通过测试;第四,进行代码检查,对不合理的地方进行代码重构,并保证通过测试;最后,重复以上流程开发其余功能模块,完成软件开发。

1 开发准备工作

本文以 Visual Studio 2013 作为开发工具,数据库采用 SQL Server 2012,以我们常见的订单操作为例,说明如何应用测试驱动开发技术。 订单的基本操作包括增加、删除、更新和查询,订单存在已提交、已付款、已审核、已发货、已完成、冻结等状态。 由于很多操作只能在特定状态下进行,因此就存在数据的并发控制问题。 假设经过业务分析,设计数据库见表1。

表 1 订单数据库设计

序号	字段名	中文名	类型	约束	说明
1	ID	唯一编号	uniqueidentifier	主键	GUID,作为操作条件
2	CUSTOMER_ID	客户编号	uniqueidentifier	非空	GUID,关联客户表
3	PRODUCT_ID	产品编号	uniqueidentifier	非空	GUID,关联产品表
4	QUANTITY	数量	money	非空	表示件数或者重量
5	AMOUNT	金额	money	非空	由单价、数量决定
6	STATUS	状态	tinyint	非空	记录、追踪订单状态
7	ACCOUNT_ID	录入人 Id	uniqueidentifier	非空	GUID,关联账号表
8	CREATE_TIME	录入时间	datetime	非空	新增时记录,禁止修改
9	TICKS	计时周期数	bigint	非空	数据并发控制

经过整理,确定实现以下两项功能:

1) 实现订单的增加、删除、更新和查询功能;实现乐观并发控制(First Win),同时读取数据后,只有第一个能保存。

2) 设计完成后, 打开 Visual Studio, 新建一个 Visual Studio 解决方案;然后在解决方案下新建 2 个项目,一个 Visual C#类库项目(BusinessLayer) 和一个单元测试项目(BusinessLayer. Tests);最后在单元

```
Guid CustomerId {get;set;}
Guid ProductId {get;set;}
Decimal Quantity {get;set;}
Decimal Amount {get;set;}
byte Status {get;set;}
Guid AccountId {get;set;}
DateTime CreateTime { get; set; }
long Ticks { get; set; }
```

测试项目中添加类库项目的引用。 完成后的解决方案如图 1 所示。

图 1 解决方案结构图

2 实现基本功能

2.1 定义操作接口

新建一个 IOrder 接口,定义订单属性;新建一个 IOrderProvider 接口,定义对订单数据的操作,代码如下所示:

```
public interface IOrder
{ / / 定义订单属性
Guid Id { get; set; }
}
public interface IOrderProvider
{ / / 定义订单操作
List < IOrder > Fetch();
IOrder Insert(IOrder order);
IOrder Update(IOrder order);
void Delete(IOrder order);
}
```

2.2 定义测试类

在测试项目中新建测试类 OrderTests,查询测试代码如下,新增、修改、删除的测试代码类似。 编译后运行全部测试,结果如图 2 所示。

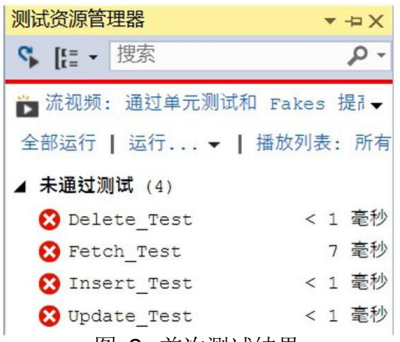

图 2 首次测试结果

```
[TestClass]
public class OrderTests
{
[TestMethod]
public void Fetch_Test()
{ /  / 查找功能测试
  Assert. Fail();
  }
}
```

2. 3 实现业务功能

在类库项目中新建 Order 类,实现 IOrder 接口;新建 OrderProvider 类, 实现 IOrderProvider 接口; 并修改测试代码,测试结果如图 3 所示。

```
public void Fetch_Test()
{ / /          查找功能测试
var list = new OrderProvider() . Fetch();
Assert. IsTrue(list. Count > 0);

}
```

图 3 实现查找功能后测试结果

从图 3 可以看到测试断言 Assert. IsTrue 失败,原因是数据库订单表中没有数据。 执行 SQL 语句插入一条数据后,测试结果如图 4 所示。

图 4 插入数据后测试结果

依次实现新增、修改、删除功能,并修改测试方法,测试结果如图 5 所示。

图 5 测试结果

测试类如下所示

```
:
[TestClass]
public class OrderTests
{
private static Guid _accountId;
[ClassInitialize]
public static void ClassInitialize( TestContext context)
{ / / 测试类初始化,新增多个订单是为了保证后续测试有数据可用
_accountId = new Guid("11111111" );
for (int i = 0; i       < 5; i + + )
{
var data  = new Order ( ) { AccountId   = _ accountId };
new OrderProvider(). Insert(data);
}
}
[TestMethod]
public void Fetch_Test()
{ / / 查找功能测试,列表条数 > 0 则通过
var list = new OrderProvider()
. Fetch();
Assert. IsTrue(list. Count > 0);
}
[TestMethod]
public void Insert_Test()
{ / / 新增功能测试, 新增后列表条数更大则通过
var list = new OrderProvider() . Fetch();
```

法。 对功能代码进行调试修改,保证通过全部测试

var data = new Order ( ) { AccountId = _ ac-

万方数据

```
countId };
    new OrderProvider() . Insert(data);
    var newList = new OrderProvider() . Fetch();
    Assert. IsTrue(newList. Count > list. Count);
    }
    [TestMethod]
    public void Update_Test()
    { / / 更新功能测试,更新满足条件的全部订单
    foreach (var item in new OrderProvider( ) . Fetch
())
    {
    if (item. AccountId ! = _accountId) continue;
    item. Quantity = 99. 9M;
    item. Amount = 10. 2M;
    new OrderProvider() . Update(item);
    }
    }
    [TestMethod]
    public void Delete_Test()
    { / / 删除功能测试, 删除满足条件的第一个
订单
    foreach (var item in new Order() . Fetch())
    if (item. AccountId = = _accountId)
    {
    new OrderProvider() .
    Delete(item); break;
    }
    }
    [ClassCleanup]
    public static void ClassCleanup()
    { / / 类清理,删除测试数据,恢复数据库
    foreach (var item in new Order() . Fetch())
    if (item. AccountId = = _accountId)
    new OrderProvider() . Delete(item);
    }
    }
```

3 实现并发控制

在测试类中新增两个测试方法,分别测试并发更新和并发删除。 并发更新测试方法如下所示;并发删除类似,区别是将第二个更新改为删除操作。

```
    [TestMethod,ExpectedException(typeof(Applica-
tionException))]
    public void UpdateException_Test()
    { / / 并发更新测试,连续更新必须抛出异常
    foreach (var item in new Order() . Fetch()) if
    (item. AccountId = = _accountId)
    {
    new OrderProvider(). Update(item);
    new OrderProvider(). Update(item);
    break;
    }
    }
```

运行全部测试,只有两个新增测试方法未通过。将更新操作的 SQL 语句改为:

@ " update Orders set quantity = @ quantity, a-mount = @ amount,status = @ status,ticks = @ current where id = @ id and ticks = @ ticks"

并将当前时钟周期数(DateTime. Now. Ticks) 和上一次时钟周期数作为参数传入;

将最后语句改为“ if(cmd. ExecuteNonQuery( )! = 1) throw new ApplicationException();”

更改以上操作后即通过测试。

删除功能的修改与更新功能类似。

在实际代码中,利用数据库提供的事务机制,才能保证数据的一致性。 在. net 平台, 推荐使用 TransactionScope 类。

4 结语

测试驱动开发是一种先进的设计方法论,主要应用于编码阶段,也可以推广到软件的整个开发过程,将软件需求变成一套可以执行的代码,便于迅速发现故障、定位故障,有利于提高程序员的工作信心。 测试驱动开发的缺点是增加了代码输入量,但是 Visual Studio 有插件(Unit Test Generator),可以根据函数自动产生单元测试代码。