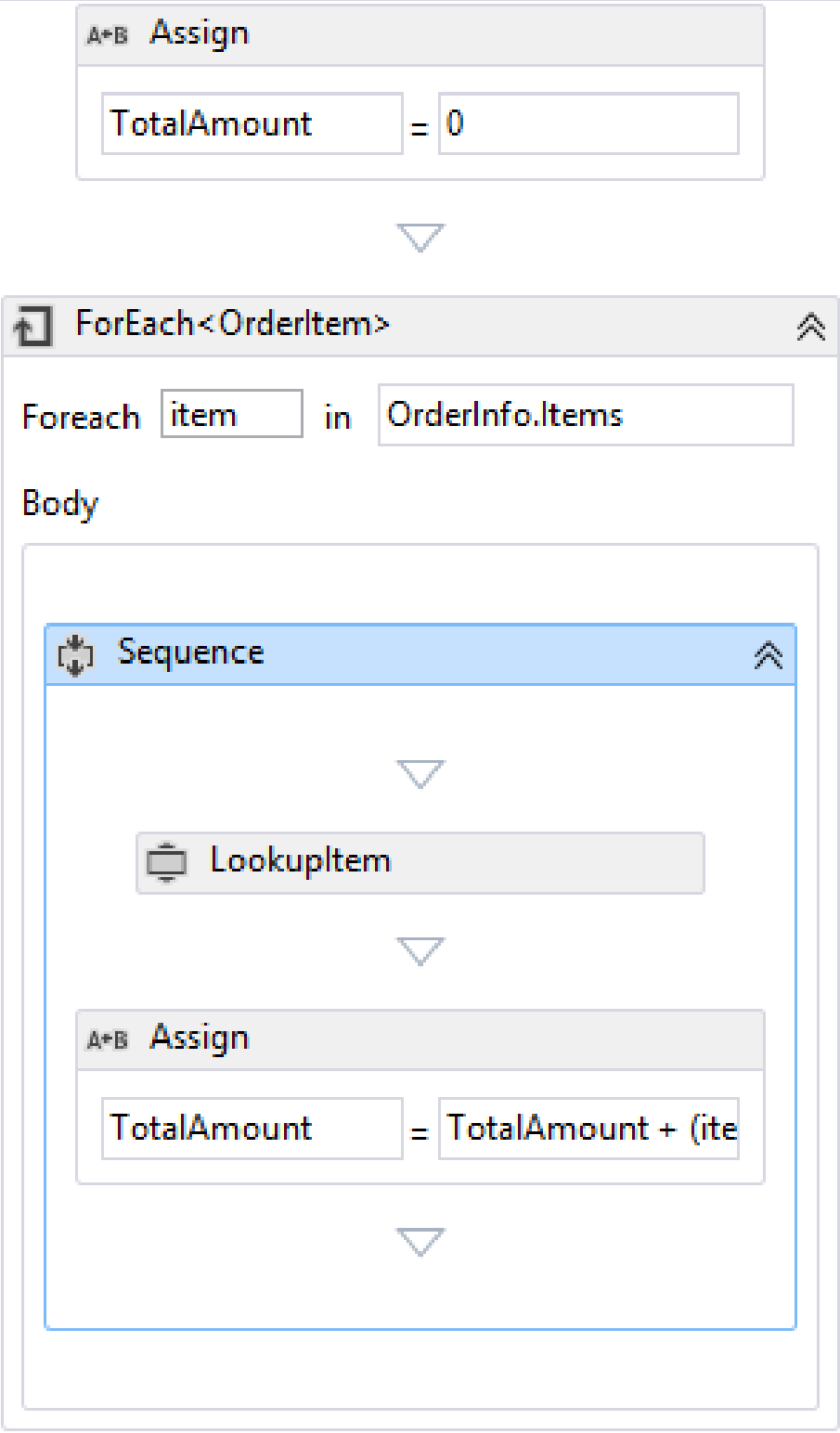
عنوان: WF:Windows Workflow #۶
نویسنده: محمد جواد تواضعی
تاریخ: ۱۳۹۱/۰۹/۱۲ ۲:۳۰
آدرس: www.dotnettips.info
گروه‌ها: WorkFlow Foundation, Workflow

در این قسمت به تکمیل مثالی که در قسمت قبل زده شد پرداخته می‌شود و همچنین کنترل‌های Foreach , Try Catch نیز بررسی خواهند شد.

در ابتدا دو کلاس به نام‌های ItemInfo و OutOfStockException را به برنامه اضافه می‌کنیم. کلاس اول برای ذخیره سازی مشخصات هر سفارش و کلاس دیگر برای مدیریت خطاها می‌باشد.

```
public class ItemInfo
    {
        public string ItemCode { get; set; }
        public string Description { get; set; }
        public decimal Price { get; set; }
    }

public class OutOfStockException : Exception
    {
        public OutOfStockException()
            : base()
        {

        }

        public OutOfStockException(string message)
            : base(message)
        {

        }
    }
```

در Workflow مورد نظر که به نام OrderWF.xaml می‌باشد, پس از کنترل Assign که برای صفر کردن مقدار متغیر TotalAmount از آن استفاده می‌شود, یک کنترل ForEach را به Flow جاری اضافه می‌کنیم. این کنترل دارای دو خاصیت به نام‌های Type Arguments و Values می‌باشد. اولین خاصیت که مقدار پیش فرض آن، مقدار عددی Int32 است, برای مشخص کردن نوع متغییر حلقه و دیگری برای مشخص کردن نوع منبع داده حلقه تعریف شده‌اند.



همانطور که در شکل بالا مشخص می‌باشد, Type Arguments حلقه برابر با کلاس OrderItem می‌باشد. Values هم برابر با OrderInfo.Items است. از این جهت نوع حلقه را از جنس کلاس OrderItem مشخص کرده‌ایم تا کنترل بر روی مقادیر Items اجرا شود (لیستی از کلاس مورد نظر).

حال همانند شکل بالا، در قسمت Body کنترل ForEach، یک کنترل Sequence را ایجاد کرده و سپس برای اینکه کنترل LookupItem را ایجاد کنیم, ابتدا باید یک Code Activity را به پروژه اضافه کنیم. به همین منظور پروژه جاری را انتخاب کرده و یک Code Activity به آن اضافه و نام آن را LookupItem می‌گذاریم. سپس کد زیر را به آن اضافه می‌کنیم:

```
public sealed class LookupItem : CodeActivity
    {
        // Define an activity input argument of type string
        public InArgument<string> ItemCode { get; set; }
        public OutArgument<ItemInfo> Item { get; set; }

        // If your activity returns a value, derive from CodeActivity<TResult>
        // and return the value from the Execute method.
        protected override void Execute(CodeActivityContext context)
        {
            // Obtain the runtime value of the Text input argument
            ItemInfo i = new ItemInfo();
            i.ItemCode = context.GetValue<string>(ItemCode);
            switch (i.ItemCode)
            {

                case "12345":
                    i.Description = "Widget";
                    i.Price = (decimal)10.0;
                    break;
                case "12346":
                    i.Description = "Gadget";
                    i.Price = (decimal)15.0;
                    break;
                case "12347":
                    i.Description = "Super Gadget";
                    i.Price = (decimal)25.0; break;
            }

            context.SetValue(this.Item, i);

        }
    }
```

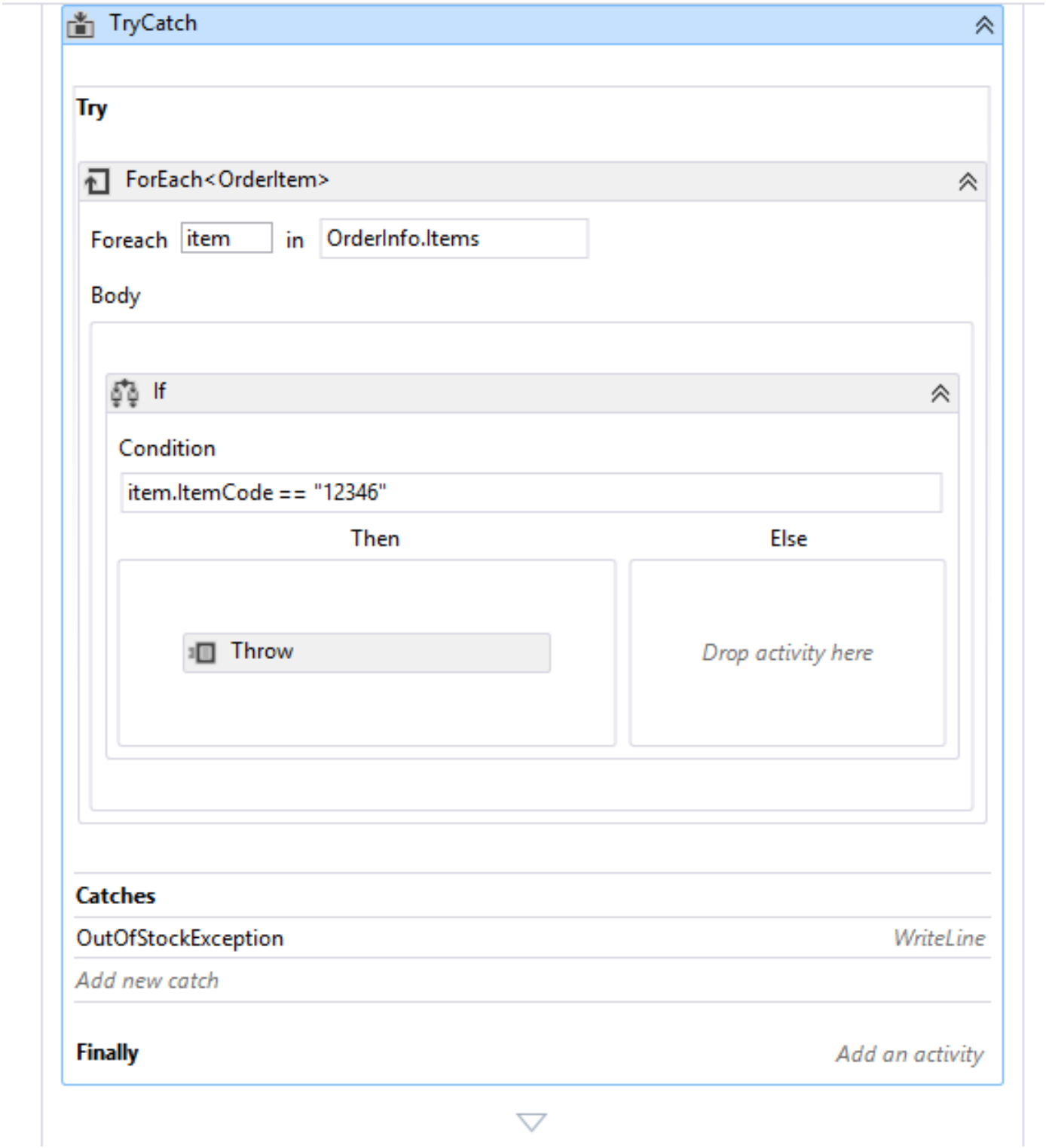
در این کد، دو متغییر تعریف شده‌اند؛ یکی از نوع رشته بوده و از طریق آن، دستور Switch تصمیم می‌گیرد که کلاس ItemInfo را با چه مقادیری پرکند. متغییر دیگر از نوع کلاس ItemInfo می‌باشد و برای گرفتن مقدار کلاس از دستور Switch تعریف شده است.
حال برای اینکه بتوانیم از Code Activity مورد نظر استفاده کنیم, ابتدا باید پروژه را یکبار Build کنیم. اکنون در قسمت Toolbox یک, Tab ایی به نام پروژه ایجاد شده است و در آن یک کنترل به نام LookupItem موجود می‌باشد. آن را گرفته و به درون Sequence انتقال می‌دهیم.
سپس برای مقدار دادن به متغیرهای تعریف شده در Code Activity، کنترل LookupItem را انتخاب کرده و در قسمت Properties به خصوصیت ItemCode، کد زیر را اضافه می‌کنیم:

```
item.ItemCode
```

نکته : از کلاس Code Activity برای ارسال و دریافت مقادیر به درون Workflow استفاده می‌شود.

Try Catch

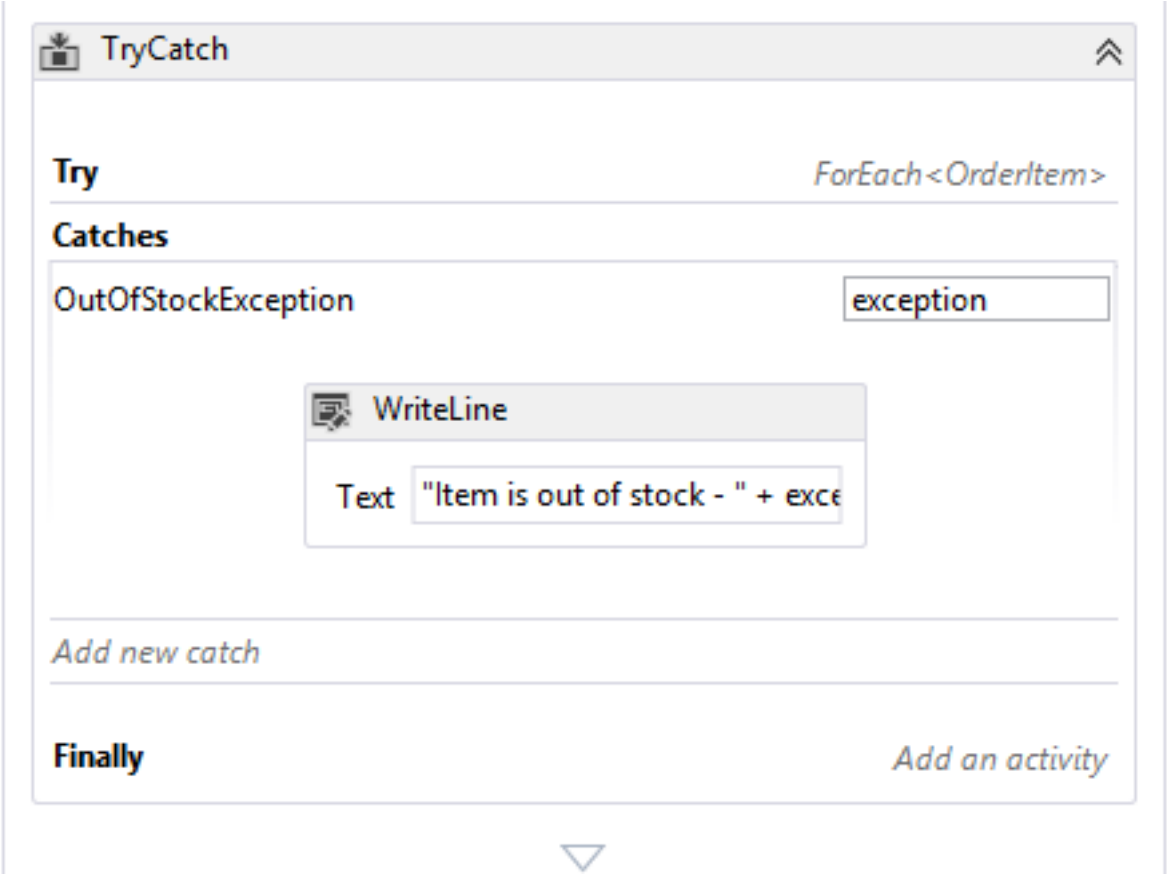
از این کنترل برای مدیریت خطاها استفاده می‌شود.
ابتدا یک کنترل Try Catch را به Workflow اضافه کرده، مانند شکل زیر:



در بدنه Try می‌توان از کنترل‌های مورد نظر استفاده کنیم و همانطور که در شکل بالا مشخص است, از کنترل Throw برای ایجاد خطا استفاده شده‌است. کنترل جاری را انتخاب کرده و از قسمت Properties در خاصیت Exception کد زیر را اضافه می‌کنیم:

```
new OutOfStockException("Item Code"+item.ItemCode)
```

این دستور باعث ایجاد یک خطا از نوع کلاس OutOfStockException می‌شود. برای کنترل خطای مورد نظر در قسمت Catches مانند شکل زیر عمل می‌کنیم.


TryCatch
Try
ForEach<OrderItem>
Catches
OutOfStockException
exception
WriteLine
Text
"Item is out of stock - " + exce
Add new catch
Finally
Add an activity

نظرات خوانندگان

نویسنده: حامد
تاریخ: ۱۳۹۱/۰۹/۱۵ ۷:۲۶

با سلام
مطالب بسیار آموزنده ایه و خوبی اون اینه که همه‌ی برنامه نویسان از اون استفاده می‌برند.
لطفاً ادامه بدید.

نویسنده: محمد جواد تواضعی
تاریخ: ۱۳۹۱/۰۹/۱۸ ۳:۵۳

سلام آقا حامد
حتما , من اینجا از همه دوستان عذر خواهی میکنم که در ارسال مطالب وقفه ایجاد شد .

نویسنده: ترابی
تاریخ: ۱۳۹۲/۰۳/۲۲ ۱۱:۱۲

باتشکر از مطالب مفیدتون

منتظر ادامه‌ی آموزش‌ها هستیم

موفق باشید

نویسنده: سروش شیرالی
تاریخ: ۱۳۹۴/۰۳/۲۲ ۲۰:۲۰

سلام و تشکر به خاطر مطالب مفیدتان
شما پروژه ای از نوع workflow console application را به پروژه اضافه نموده اید. اگر من بخواهم در یک پروژه وب از wf استفاده کنم نیز باید پروژه ای از همین نوع را به solution برنامه اضافه نمایم؟
در یکسری ویدیوهای آموزشی من پروژه هایی از نوع sequential و غیره دیده ام اما من که از vs 2013 استفاده می‌کنم این گزینه‌ها را ندارم . ممکن است راهنمایی بفرمایید؟

نویسنده: محمد جواد تواضعی
تاریخ: ۱۳۹۴/۰۳/۲۳ ۱:۰

می توانید پروژه wf را به صورت WCF WorkFlow Service Application در Solution مورد نظر اضافه کنید پس از ان سرویس را بر روی ویندوز سرور هاست کنید به کمک برنامه AppFabric که می‌توانید ان را از لینک زیر دانلود کنید .
http://msdn.microsoft.com/appfabric

روش دیگر این است که شما مستقیما از کلاس‌های WF در پروژه خود استفاده کنید و Activityهای خود را تولید کنید بدون اینکه احتیاج به Model Designer داشته باشید مانند کد زیر:

```
namespace LeadGenerator
{

public sealed class CreateLead : CodeActivity
{
public InArgument<string> ContactName { get; set; }
public InArgument<string> ContactPhone { get; set; }
public InArgument<string> Interests { get; set; }
public InArgument<string> Notes { get; set; }
```

```
public InArgument<string> ConnectionString { get; set; }
public OutArgument<Lead> Lead { get; set; }
protected override void Execute(CodeActivityContext context)
{
// Create a Lead class and populate it with the input arguments
Lead l = new Lead();
l.ContactName = ContactName.Get(context);
l.ContactPhone = ContactPhone.Get(context);
l.Interests = Interests.Get(context);
l.Comments = Notes.Get(context);
l.WorkflowID = context.WorkflowInstanceId;
l.Status = "Open";
// Insert a record into the Lead table
LeadDataDataContext dc =
new LeadDataDataContext(ConnectionString.Get(context));
dc.Leads.InsertOnSubmit(l);
dc.SubmitChanges();
// Store the request in the OutArgument
Lead.Set(context, l);
}
}
}
```