ARC320
基于.Net Framework2.0
企业应用框架的设计与实现

课程内容概述

我们试图通过实际的企业应用框架的设计开发过程，跟大家分享三件事情。

第一开发适合自己企业的基础框架，可以极大提的高生产率和软件质量；

第二讲述我们基于DotNet2.0开发自己企业应用框架的故事；

第三探讨企业应用框的架设计和实现。

课程内容安排

- 为什么我们需要应用框架
- 企业应用框架介绍
- 企业应用框架的设计
- 企业应用框架的实现
- 企业应用框架实施

为什么我们需要应用框架

问题？

在我们的项目开发中是否经常遇到这些问题：

- 一些基础的功能在不同的系统中总是重复开发？譬如：数据访问，事务管理，用户认证，权限管理等等。
- 不同的程序员总是出现同样的错误？譬如：没有释放资源，异常没有处理等等。
- 功能相似的代码在多个地方出现？
- 每个项目启动都要搭建一套基础框架？
- 在组织级没有技术积累，项目的成败很多时候决定于几个关键成员？

问题？（续）

- 没有体系结构的代码随处可见，大家风格各异，维护性差？
- 系统的扩展性差，没有成员愿意动以前的代码，常常更愿意重新做？
- 系统的模块、单元职责不清，紧密耦合，很难重用？
- ……

如何解决这些问题？

- 引入应用框架是一种行之有效的方法。

让软件开发工作变为一种荣誉感和成就感

企业应用框架介绍

框架

- 在建筑行业框架指的是支撑其它物体的结构，或者说是建筑物的基础支撑骨架。

软件的框架是一个软件的支撑结构，是经验的积累，是失败的教训，是一种创新，是一组可重用的设计和代码 ……

为何使用应用框架

- 从已有的框架分析：
  - MVC
  - MFC
  - DotNet
  - JAVA
  - Spring
  - ……
- 基于框架开发可以极大提高生产率和软件质量。
- 使用框架可以使我们摆脱前面所述“问题”的困扰。
- 使用框架还有更多的优点。

使用框架的优点（1）

- 模块化
  - 把应用分割成多个组件或者模块，分而治之。
  - 隔离变化的影响范围。
  - 降低系统的复杂性。
- 可重用
  - 框架为项目提供可重复使用的，稳定、成熟、可靠的组件。
  - 极大的提高系统的开发效率。
  - 不仅仅是代码，组件的重用，而且重用了以前的设计、经验、教训。
  - 站在巨人的肩上。

使用框架的优点（2）

- 简洁性
  - 不是框架简单了，是应用程序开发者的工作变得简单快乐。
  - 通过框架封装了基础的处理流程和控制逻辑，开发者可以透明的使用，极大的提高开发效率，节约成本。
- 可维护
  - 是我们的应用系统可以“随需而变”的一种能力。
  - 由于框架被多个应用所共享，代码已有一份，所以只需修改一次，所有的地方都保持一致。
  - 使用框架，业务规则被抽象出来作为一种扩展机制实现业务规则的修改只需要修改配置或者提供一个扩展实现即可。

使用框架的优点（3）

- 可靠性
  - 系统持续正确的运行，不间断的提供服务的一种能力。
  - 框架是开发工作长期的积累，是经过实践稳定下来的一组可重用的设计和代码，它是经过实践检验的，可以提供极高的可靠性。
- 可扩展
  - 框架可以非常方便增加自定义的功能，以适应特定的业务需求。
  - 在框架设计中很关键的是在应用可能发生变化的地方增加系统的扩展性和灵活性。
- ……

使用框架的不足

- 开发成本
  - 框架的设计开发需要大量的人力资源和物质资源。
  - 开发高可用性和高可靠性的框架，需要业务领域的专家和软件设计开发领域的专家。
  - 框架的开发是一种长线的投资，适用于有多个应用的大中型企业。
- 学习成本
  - 要在项目中应用框架，所有的开发者首先要学习框架的使用。框架提供了一种新的编程模型，新的API，新的服务和配置项，需要开发者掌握更多的开发技术，才可以高效的构建应用系统。

框架的分类

- 业务应用（Business Application）
  - 实现具体业务应用的框架。
- 应用框架（Application Framework）
  - 应用系统的支撑体系结构，应用系统的半成品。
  - 特定领域框架
    - 针对特定领域的业务逻辑构建的专有组件。
  - 跨领域的框架
    - 不包括业务领域知识的共享组件。
- 基础框架（Foundation Framework）
  - 应用框架和业务系统构建的基础，软件开发商提供的开发工具：DotNet、Java、MFC，Spring，Struts。

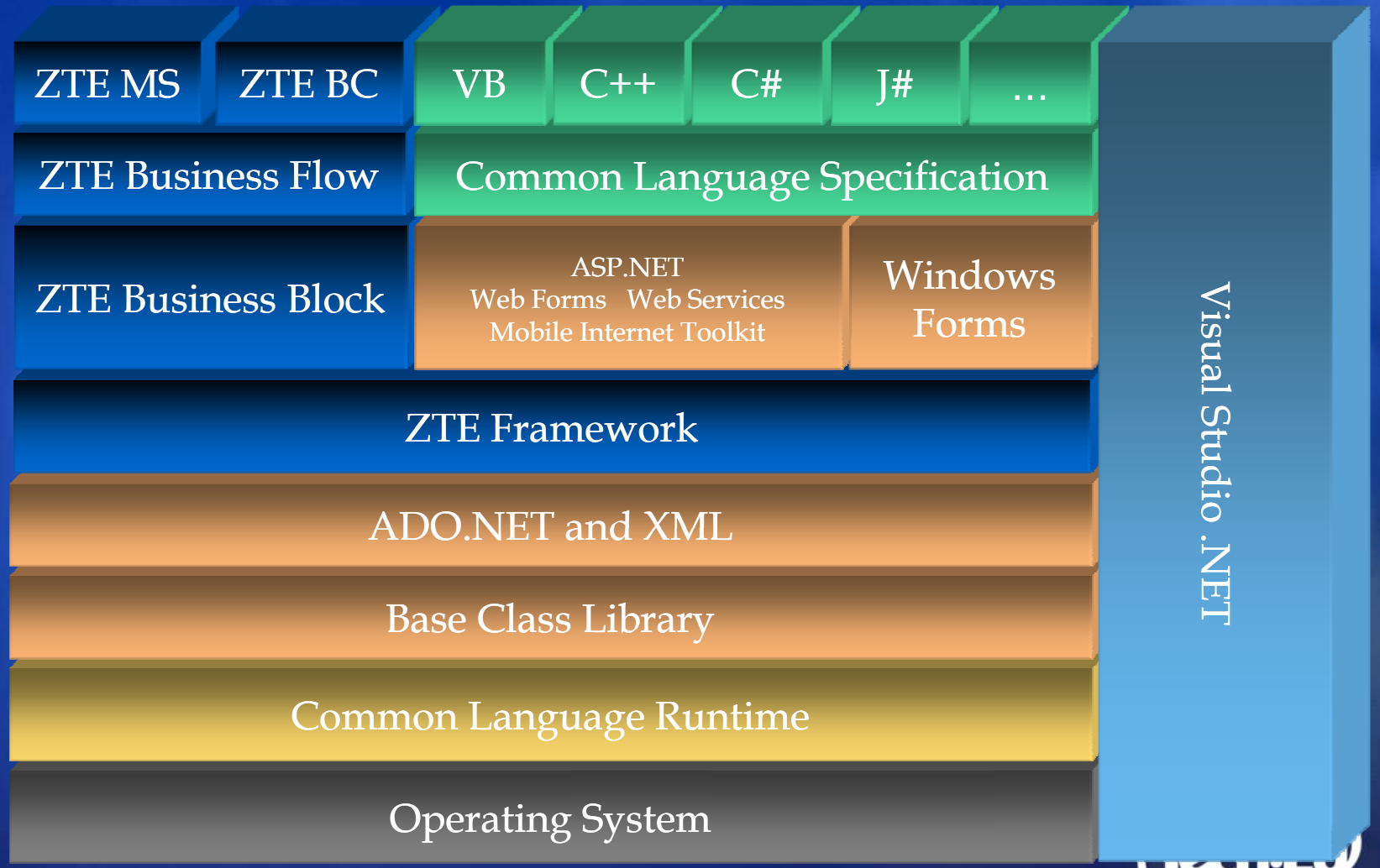
框架的开发过程

和其它项目一样，框架开发也包含下面几个阶段：

- 分析
  - 确定框架的目标和范围。
- 设计
  - 确定架构的通用点和扩展点，设计框架的蓝图。
- 实现
  - 编码实现框架。
- 稳定
  - 开发者验证框架实现，开发Demo项目和文档。

企业应用框架的设计

框架设计蓝图



框架设计解析（1）

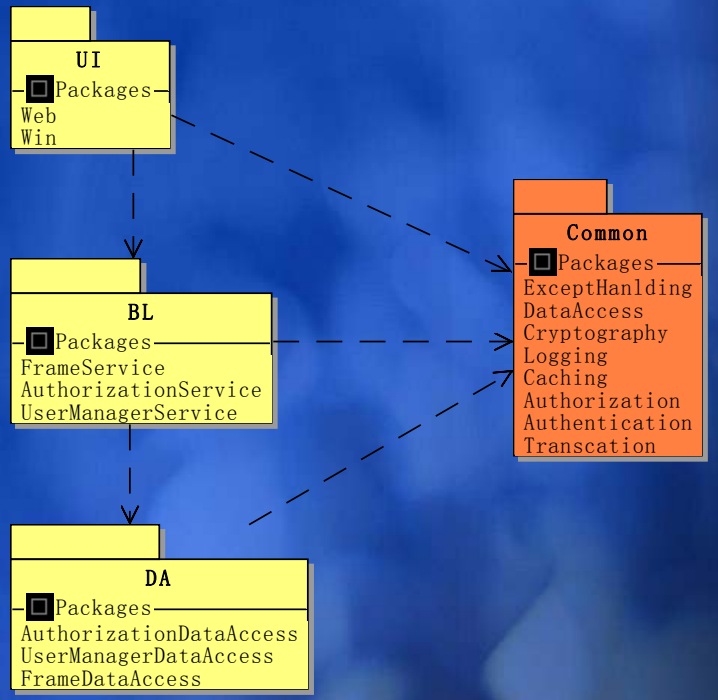
- **ZTE Framework** 统一的基础平台，包括系统各个层次需要的基础功能。如：数据访问组件，事务管理组件，日志记录组件，异常处理组件等。
- **ZTE Business Block** 各种商务应用模块集合。例如：采购管理模块，合同管理模块，库存管理模块，财务管理模块，计划模块，MRP模块，车间管理模块，供应商管理模块，销售管理模块，客户管理模块等等。每个块都是产品化的系统模块，可以方便的Plug-In到系统框架，通过ZTE Management Studio 配置后，可以满足各种典型的商务应用。

框架设计解析（2）

- **ZTE Business Flow** 工作流组件。支持ZTE Business Block之间的工作流控制和管理。
- **ZTE Management Studio** 框架产品的管理、配置、监控平台。
- **ZTE Business Client** ZTE业务客户应用模块，用户可按CS模式使用，也可按BS模式使用。
- **Visual Studio.Net** 框架的开发工具，支持用户使用Visual Studio.Net开发新的模块，注册到 ZTE Business Block中，扩展系统的功能，满足自定义的业务功能扩展。

ZTE Framework设计

- 通用类库（ Common ）
- 应用层类库（ Application ）
- 业务层类库（ Business ）
- 资源层类库（ DataAccess ）
- 工具库（ Tools ）



通用类库（ Common ）

- 系统通用的组件集合
  - 数据访问组件
  - 事务管理组件
  - 异常管理组件
  - 日志管理组件
  - 缓存管理组件
  - 数据加密组件
  - 服务通讯组件
  - 自动更新组件

应用层类库（ Application ）

- 系统应用层通用组件集合
  - 客户端主框架
  - Web主框架
  - WinForm用户管理
  - WebForm用户管理

业务逻辑层类库（ Business ）

- 系统业务层可重用组件集合
  - 系统主框架服务
  - 用户认证服务
  - 用户管理服务

数据访问层类库（ DataAccess ）

- 系统资源访问层可重用的组件集合
  - 系统主框架数据访问组件
  - 用户认证数据访问组件
  - 用户管理数据访问组件

工具库（Tools）

- 自动更新配置工具
- 文档生成工具（NDoc2.0）
- 查询生成工具
- 密钥生成工具

企业应用框架的实现

通用类库实现（1）

- 数据访问组件
  - 扩展Enterprise Library中的DataAccess模块
  - 添加SqlMapper功能
  - 添加DbRefCursor功能Oracle返回游标。
- 事务管理组件
  - 基于数据访问组件实现，使用线程静态的上下文类保持用户的数据库链接状态和事务状态。
  - 自动绑定参数。

通用类库实现（2）

- 异常管理组件
  - 扩展Enterprise Library中的ExceptionHandling模块
  - 定制了两种类型的异常处理策略
  - 定义系统基础异常
  - 处理数据库异常（常见的错误码替换为用户异常）
- 日志管理组件
  - 扩展Enterprise Library中的Logging模块
  - 定义自己的日志配置，简化了日志写出动作。

通用类库实现（3）

- 缓存管理组件
  - 直接使用Enterprise Library中的Cache模块。
- 数据加密组件
  - 直接使用Enterprise Library中的Cryptography模块。

通用类库实现（4）

- 服务通讯组件
  - 包装DotNet中的Remoting和Web服务的类库，通过配置实现切换。
  - 添加服务基础类，用户上下文信息。
- 自动更新组件
  - 扩展Update Application Block模块。
  - 添加按定义的模块，角色下载部分的应用程序。
  - 自动更新配置工具。

应用层类库实现（1）

- 客户端主框架
  - 包括系统主界面，用户登陆窗，基础窗体，版本信息等
  - 用户登陆全过程（认证，系统初始化）
  - 用户权限透明校验
  - 用户菜单加载
  - 通过配置文件动态加载业务窗体
- Web主框架
  - MasterPage、基础WebForm，用户登陆，系统版本信息
  - 用户登陆认证过程。
  - 用户权限数据的加载，系统初始化
  - 通过配置动态加载业务应用界面。

应用层类库实现（2）

- WinForm用户管理
  - 用户创建，修改，分配角色等UI界面实现
  - 角色创建，修改，授权的UI界面实现
- WebForm用户管理
  - 用户创建，修改，分配角色等Web窗体实现。
  - 角色创建，修改，授权的Web页面实现。

业务层类库实现

- 系统主框架服务
  - 按用户的权限取用户的菜单项，构建用户的菜单树。
  - 取的用户Profile数据。
  - 用户的自定义的桌面，与我有关。
- 用户认证服务
  - 提供内部认证和外部认证方式。
  - 认证的方法可扩展。
- 用户管理服务
  - 用户权限模型实现
  - RBAC0模型实现
  - RBAC2模型实现

数据访问层类库实现

- 系统主框架数据访问组件
  - 用户权限数据读取，用户菜单数据，角色数据等。
  - 用户Profile数据，用户桌面数据。
- 用户认证数据访问层
  - 访问数据库读取用户身份认证的数据。
- 用户管理数据访问层
  - 用户数据读取，修改。
  - 角色数据读取，修改。
  - 用户角色关联数据读取，修改。
  - 角色权限数据读取，修改。

工具库（1）

- 自动更新配置工具
  - 使用UAB 中的配置工具扩展，生成按模块或者角色的方式下载部指定部分的模块。
- 文档生成工具（NDoc2.0）
  - 当时没有NDoc2.0版本，我们用NDoc提供的源码，升级到DotNet2.0版本。
  - 修改中文XML处理乱码问题。

工具库（2）

- 查询生成工具
  - 用户使用工具定义界面元素，指定数据库查询过程。
  - 使用数据库保存用户定义的元数据。
  - 使用解析引擎解析界面的元数据，动态生成界面元素。
  - 绑定数据库过程返回结果集到界面。

企业应用框架的实施

框架给大家的是什么？

- 一个安装包
  - 基础类库集合。
  - 系统开发、支持的一些工具。
- 一份开发说明文档
  - 开发规范。
  - 用户手册。
  - SDK
- 一个Demo项目
  - 一个Demo项目提供给开发者学习用。

通用类库使用（1）

- 事务管理组件
- 异常管理组件
- 日志管理组件
- 缓存管理组件
- 数据加密组件
- 服务通讯组件
- 自动更新组件

通用类库使用（2）

```
/// <summary>
/// 保存扩展信息。
/// </summary>
private void SaveCustomerExtendInfo()
{
    CustomerExtendInfo extInfo = new CustomerExtendInfo(this._CustomerId,
        this.txtContact.Text,
        long.Parse(this.ddlType.SelectedValue),
        this.ddlProvince.SelectedIndex == 0 ? (long?)null : long.Parse(this.ddlProvince.SelectedValue),
        this.txtDescription.Text,
        this.txtBank.Text,
        this.txtAccountNo.Text,
        this.txtAccountName.Text,
        this.txtDebtUpLimit.Text.Trim() == string.Empty ? (double?)null : double.Parse(this.txtDebtUpLimit.Text.Trim()),
        this.txtDebtPeriod.Text.Trim() == string.Empty ? (int?)null : int.Parse(this.txtDebtPeriod.Text.Trim()),
        this.ddlPriceType.SelectedIndex > 0 ? short.Parse(this.ddlPriceType.SelectedValue) : (short?)null,
        this.txtPriceRate.Text.Trim() == string.Empty ? (double?)null : double.Parse(this.txtPriceRate.Text.Trim()),
        this.txtPriceDesc.Text,
        this.LastUpdateDate);

    extInfo.ContactList = this._ContactList;

    new CustomerService().SaveCustomerExtentInfo(extInfo);

    this.LastUpdateDate = extInfo.LastUpdateDate;
}
```

通用类库使用（3）

```
/// <summary>
/// 保存客户扩展信息。
/// </summary>
/// <param name="customerExtendInfo"></param>
public void SaveCustomerExtentInfo(CustomerExtendInfo customerExtendInfo)
{
    try
    {
        this.BeginTransaction();

        CustomerDao.UpdateCustomerExtendInfo(customerExtendInfo);

        CustomerDao.DeleteCustomerAllContact(customerExtendInfo.CustomerId);

        foreach (CustomerContact c in customerExtendInfo.ContactList)
        {
            if (c.IsValid)
            {
                CustomerDao.InsertCustomerContact(customerExtendInfo.CustomerId, c);
            }
        }

        this.Commit();
    }
    catch (Exception ex)
    {
        this.Rollback();
        throw this.HandleException(ex);
    }
}
```

通用类库使用（4）

```
/// <summary>
/// 更新客户扩展信息，如果没有扩展信息，新增记录。
/// </summary>
/// <param name="customerExtendInfo"></param>
public static void UpdateCustomerExtendInfo(CustomerExtendInfo customerExtendInfo)
{
    DbInstance db = DbManager.GetAutoParameterDbInstance();

    DateTime now = DateTime.Now;

    // 更新扩展信息，如果没有记录，新增。
    db.ExecuteNonQuery("$BasisInfo.UpdateCustomerExtendInfo",
        customerExtendInfo.CustomerId,
        customerExtendInfo.Contact,
        customerExtendInfo.TypeId,
        customerExtendInfo.ProvinceId,
        customerExtendInfo.Description,
        customerExtendInfo.AccountBank,
        customerExtendInfo.AccountName,
        customerExtendInfo.AccountNo,
        customerExtendInfo.DebtUpLimit,
        customerExtendInfo.DebtPeriod,
        customerExtendInfo.PriceType,
        customerExtendInfo.PriceRate,
        customerExtendInfo.PriceDescription,
        Context.UserId,
        now,
        customerExtendInfo.LastUpdateDate);

    // 传回之后更新时间，为了避免输出参数，用web服务器的时间。
    customerExtendInfo.LastUpdateDate = now;
}

/// <summary>
/// 删除所有客户的联系人。
/// </summary>
/// <param name="customerId"></param>
public static void DeleteCustomerAllContact(long customerId)
{
    DbInstance db = DbManager.GetAutoParameterDbInstance();

    db.ExecuteNonQuery("$BasisInfo.DeleteCustomerContact", customerId);
}
```

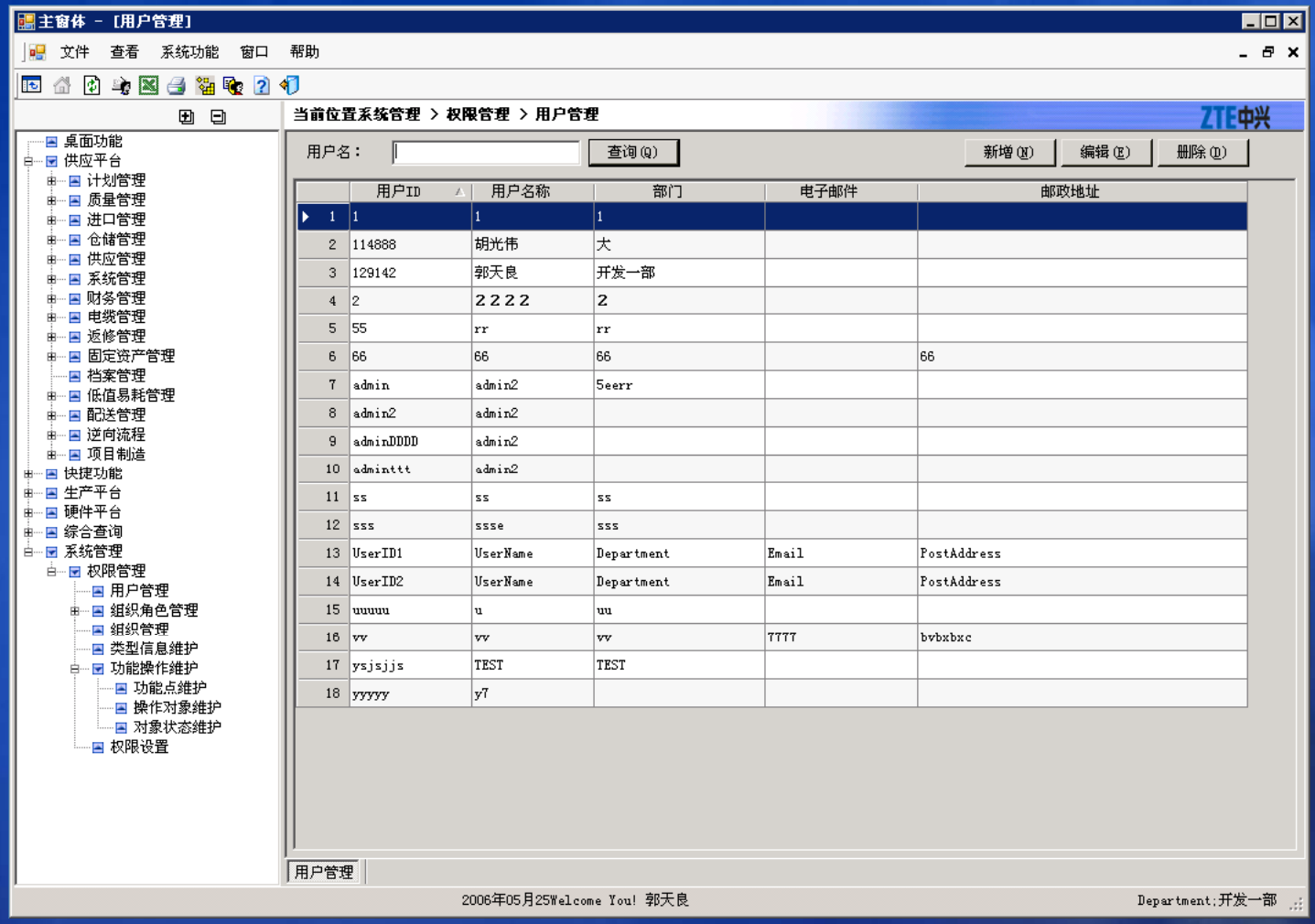
应用层类库使用

- 客户端主框架。
- Web主框架。
- WinForm用户管理。
- WebForm用户管理。
- “即拆即用”，拿来即可使用。

业务层类库使用

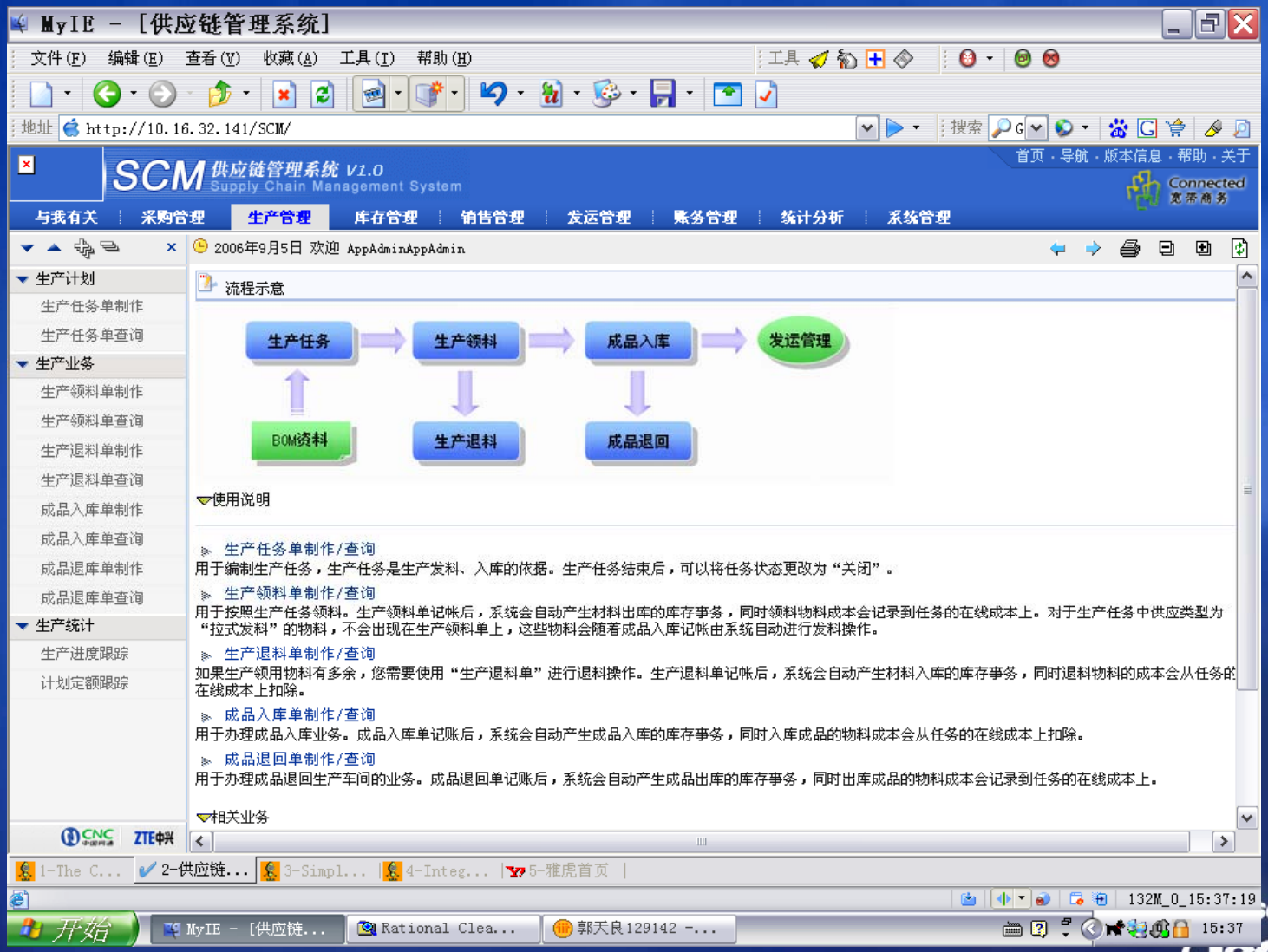
- 系统主框架服务。
- 用户认证服务。
- 用户管理服务。
- “即拆即用”，拿来即可。

框架实施项目一



Microsoft Tech·Ed 2006中国

框架实施项目二



参考资料

- 《Developing Application Frameworks *in .Net*》 Xin Chen
- 《Applying UML and Patterns》 Craig Larman
- Enterprise Library

填反馈表

Microsoft Tech·Ed 2006 中国


Microsoft®
您的潜力，我们的动力
Microsoft
Tech·Ed
2006中国