文档编号 SS-STG-003

Sextant Tour Guide 软件设计报告

**系统组件化**

版本 1.0 （第1稿）

Sextant Studio

2013/04/06

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| **版本** | **日期** | **修订者** | **说明** |
| 1.0 | 2013/04/06 | 郭旭 | 创建 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[Sextant Tour Guide 软件设计报告 1](#_Toc353141555)

[修订记录 2](#_Toc353141556)

[目录 3](#_Toc353141557)

[**一、** **引言** 4](#_Toc353141558)

[1.1 编写目的 4](#_Toc353141559)

[1.2 项目背景 4](#_Toc353141560)

[1.3 项目开发情况简介 4](#_Toc353141561)

[**二、** **系统架构** 4](#_Toc353141562)

[2.1 架构 4](#_Toc353141563)

[2.2 数据建模 4](#_Toc353141564)

[2.3 引用关系 4](#_Toc353141565)

[图 4](#_Toc353141566)

[**三、** **组件化** 4](#_Toc353141567)

[3.1 组件化说明 4](#_Toc353141568)

[3.2 组件接口设计 4](#_Toc353141569)

[3.3 配置文件 8](#_Toc353141570)

1. **引言**
   1. 编写目的

本文档对Sextant Tour Guide旅游软件组件化设计进行详细说明。

* 1. 项目背景

本项目作为《软件工程技术与设计》课程的训练项目提出，由Sextant Studio小组进行实际开发。通过Sextant Tour Guide软件的设计开发，了解软件文档的编写原则，训练掌握软件各类文档写作的技巧，同时提高和增加C#开发的能力和经验。

项目开发平台为Visual Studio 2010 (C# .Net 3.5)， 单机数据库使用SQLite。

1. **系统架构**
   1. 架构

目前的系统虽然采用C/S架构，由于作业未作要求，因此服务端和客户端暂未分离，程序、服务、数据库共同在一台机器上运行。系统严格按照三层架构设计，包括数据连接层、服务（业务逻辑）层、用户界面层，各层之间通过接口进行访问，以降低耦合性。在各层的接口中一般使用简单工厂模式进行类的初始化，工厂类使用.NET的反射性质，使得dll文件在程序运行过程中根据配置文件进行动态加载。



图 1 架构与分工

* 1. 数据建模

数据建模是在项目初期需要解决的一个较麻烦的问题。考虑到小组成员对.NET开发环境比较陌生，对于复杂的面向对象开发也不是很了解，而且本系统的业务逻辑又比较简单，因此没有采用NHibernate等ORM工具，也放弃了最为“面向对象”的领域模型（Domain Model）的建模方式，而是采用比较简洁的活动记录（Active Record）方式，即模型中的实体类与数据库的表一一对应，实体类中的公共属性与表中的字段一一对应；并且使用贫血（失血）模型，即实体类中仅有属性，没有任何方法，以最大程度地降低与数据访问层和业务逻辑的耦合。实体类与数据库的映射采用SQL语句硬编码的方式进行映射。

* 1. 引用关系

图

1. **组件化**
   1. 组件化说明

本系统的组件化原则上按照系统的分层架构进行，主要包括：数据库访问组件、服务组件、日志组件，这几个组件均向外部提供接口以进行访问；此外，还包括一个封装了ADO.NET的数据库访问组件，以及在用户界面层实现的可编辑图片列表组件和只读图片列表组件，这几个组件由于不具有通用性，因此没有使用接口，而采用直接向外暴露公共方法以进行访问。在系统通过读取配置文件，确定实例化组件接口的方式，配置文件将在下文进行说明。

* 1. 组件接口设计

（1）数据连接层接口

|  |  |
| --- | --- |
| 类 | 接口说明 |
| 说明 | DALFactory |
| 接口 | /// <summary>  /// 创建一个数据连接层的实例  /// </summary>  /// <typeparam name="T">指定的数据类型</typeparam>  /// <returns>指定数据类型的实例</returns>  T CreateDAL<T>() where T : IBaseDAL |
|  |  |
| 类 | IBlogDAL |
| 说明 |  |
| 接口 | /// <summary>  /// 获取全部日志  /// </summary>  /// <returns>日志列表</returns>  List<Blog> GetBlogs();  /// <summary>  /// 获取指定用户的日志  /// </summary>  /// <param name="userId">用户ID</param>  /// <returns>日志列表</returns>  List<Blog> GetBlogsByUserId(string userId);  /// <summary>  /// 获取指定景点的日志  /// </summary>  /// <param name="sightsId">景点ID</param>  /// <returns>日志列表</returns>  List<Blog> GetBlogsBySightsId(string sightsId);  /// <summary>  /// 获取指定用户与景点的日志  /// </summary>  /// <param name="userId">用户ID</param>  /// <param name="sightsId">景点ID</param>  /// <returns>日志列表</returns>  List<Blog> GetBlogsByUserIdAndSightsId(string userId, string sightsId);  /// <summary>  /// 获取指定旅行的日志  /// </summary>  /// <param name="tourId">旅行ID</param>  /// <returns>日志列表</returns>  List<Blog> GetBlogsByTourId(string tourId);  /// <summary>  /// 获取指定旅行中一次子旅行的日志  /// </summary>  /// <param name="tourId">旅行ID</param>  /// <param name="subTourId">子旅行ID</param>  /// <returns>日志列表</returns>  List<Blog> GetBlogsByTourIdAndSubTourId(string tourId, string subTourId);  /// <summary>  /// 获取指定日志  /// </summary>  /// <param name="blogId">日志ID</param>  /// <returns>日志</returns>  Blog GetBlogByBlogId(string blogId);  /// <summary>  /// 向数据库插入一篇日志  /// </summary>  /// <param name="blog">日志</param>  /// <param name="trans">数据库事务</param>  /// <returns>数据库中发生变更的行数</returns>  int InsertBlog(Blog blog, DbTransaction trans);  /// <summary>  /// 更新数据库中的一篇日志  /// </summary>  /// <param name="newBlog">新的日志</param>  /// <param name="oldBlog">旧的日志</param>  /// <param name="trans">数据库事务</param>  /// <returns>数据库中发生变更的行数</returns>  int UpdateBlogFromOld(Blog newBlog, Blog oldBlog, DbTransaction trans);  /// <summary>  /// 删除数据库中的一条日志  /// </summary>  /// <param name="blog">日志</param>  /// <param name="trans">数据库事务</param>  /// <returns>数据库中发生变更的行数</returns>  int DeleteBlog(Blog blog, DbTransaction trans); |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

（2）服务层接口

（3）日志接口

（4）UI层User Control组件接口

|  |  |
| --- | --- |
| 类 | STGReadonlyPictures |
| 说明 | 只读图片列表组件 |
| 接口 | /// <summary>  /// 设置或获取控件中的所有图片  /// </summary>  List<Picture> Pictures { get; private set; }  /// <summary>  /// 设置控件，显示指定用户上传的关于指定景点所有的图片  /// </summary>  /// <param name="sightsId">景点ID</param>  /// <param name="uploaderId">用户ID</param>  /// <param name="allowComment">是否允许添加图片评论</param>  void SetPicturesForSights(string sightsId, string uploaderId, bool allowComment);  /// <summary>  /// 设置控件，显示指定用户上传的关于指定旅行的所有图片  /// </summary>  /// <param name="tourId">旅行ID</param>  /// <param name="allowComment">是否允许添加图片评论</param>  void SetPicturesForTour(string tourId, bool allowComment);  /// <summary>  /// 重置控件，清除所有图片  /// </summary>  void ResetList(); |
|  |  |
| 类 | STGPictures |
| 说明 | 图片列表组件 |
| 接口 | /// <summary>  /// 设置或获取控件中的所有图片  /// </summary>  List<Picture> Pictures { get; private set; }  /// <summary>  /// 设置或获取控件中所有已删除的图片  /// </summary>  List<Picture> RemovedPictures { get; private set; }  /// <summary>  /// 设置控件，显示指定用户上传的关于指定景点所有的图片  /// </summary>  /// <param name="sightsId">景点ID</param>  /// <param name="uploaderId">用户ID</param>  void SetPicturesForSights(string sightsId, string uploaderId);  /// <summary>  /// 设置控件，显示指定用户上传的关于指定景点和指定旅行的所有图片  /// </summary>  /// <param name="sightsId">景点ID</param>  /// <param name="tourId">旅行ID</param>  /// <param name="subTourId">子旅行ID</param>  /// <param name="uploaderId">用户ID</param>  void SetPicturesForTour(string sightsId, string tourId, string subTourId, string uploaderId);  /// <summary>  /// 重置控件，清除所有图片  /// </summary>  void ResetList(); |

* 1. 配置文件

配置文件通过.NET提供的CongfigurationManager类进行访问，按照作用分为以下几部分：数据库Provider配置，数据库连接字符串配置，数据连接层接口配置，服务层接口配置，日志接口配置以及其它相关配置。

数据库Provider配置，是对.NET默认提供的几种连接数据库的Provider的补充，主要是使用其中的SQLite配置；数据库连接字符串配置记录了连接数据库的字符串；数据连接层接口配置，记录了程序在运行中使用的数据连接层，通常针对不同的数据库开发不同的数据连接层；服务层接口和日志接口配置同上所属，记录了程序运行过程中使用的服务层和日志。