## 项目经理/.NET高级开发工程师笔试题

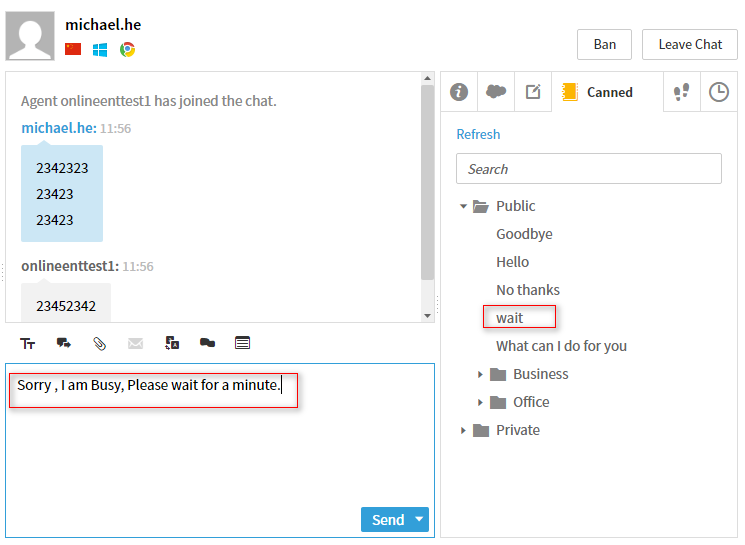
#### 一、C# &基础

某在线客服系统采用传统的三层架构

* **Data Access 层:**负责构建SQL脚本, 直接与数据库进行交互.
* **Business 层:** 封装了实体的业务逻辑, 需要调用Data Access 层的方法才能执行对应的数据库操作.
* **Process 层:**统一生成数据库连接、管理事务、调用Business 层的方法来处理UI层发送过来的请求.
* **UI层:**与用户进行交互, 调用Process 层的方法完成各种UI请求.

客服在线聊天时会使用到Canned Message

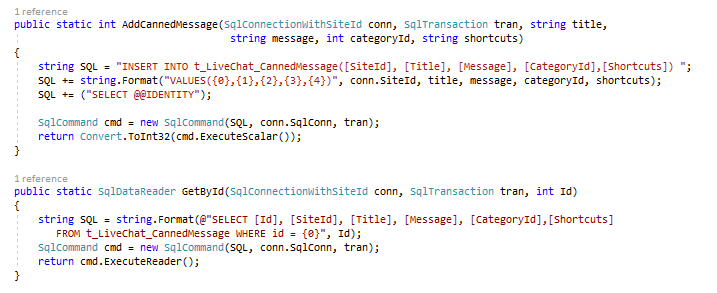
Canned Message : 预存储的常用聊天消息, 按Category 进行分类, 用于减轻客服在线聊天时的工作量. 包含Category, Title, Name, Message, Shortcuts 等属性.



举例: 客服在聊天过程中单击了Tab[Canned] 下Name为”wait”的canned message, 系统将会自动发送对应的消息: “Sorry, I am Busy, Please wait for a minute”.

以下是Canned Message 实体的相关代码:

* Data Access 层:



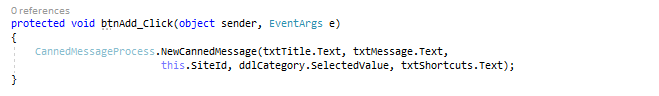
* Business 层：



* Process 层:



* UI层:



试找出上述代码的存在的错误和可优化点

1：Data Access 层，插入语句中，如果值是字符串类型，那么需要用到单引号括起来；新增之后返回Id，Select @@Identity，这句sql前面应该加上分号。应该写成如下：

SQL+=string.Format (“VALUES({0},'{1}','{2}',{3},'{4}')”,参数1,参数2,参数3,参数4,参数5)

SQL+=(“;Select @@Identity”) （+3）

2：Business层，应该增加非事物的新增方法，在聊天的场景中，如果仅仅只是单表插入时不应使用事物，使用事物会造成事物锁。（+3）

3：整个新增消息的逻辑中，应该将零散的实体参数对象化，直接传入实体对象，方便后期可读性性和维护工作。（+3）

4：UI层应该对保存结果进行判断，一般会将保存结果返回给UI层展示。（+3）

5：Process层抛出的异常没有处理，前台UI将出现异常。（+3）

（总分 15 分）

#### 二、数据库

有一Ticket（个案或工单）系统，用于处理来自于客户的邮件、在线表单等问题。其中，个案的编号、标题、描述、创建时间、状态、负责人、部门等固定字段30个。用户会根据自己的业务需求添加或删除其它自定义字段，比如产品、渠道、咨询类别等。

自定义字段类型包括：文本框，下拉框，复选框，复选框列表等。其中，下拉框/复选框列表是指用户可以预定义数据，然后在具体新建或编辑Ticket的时候，客服可以选择其中的一个值。比如，用户可以加一咨询类别，预定义类别为：售前咨询，产品质量，支付问题等。在客服在新建或修改产品的时候，客服可以选择一个或多个选项。

业务系统的使用或数据量需求：

1. 我们假定系统有非常多的Ticket，比如>100万个Ticket。
2. 系统设计应该支持客户针对Ticket字段的频繁查询。比如客户定义自己的查询为“部门为客服，负责人是自己，状态为Open，自定义字段-资讯类别为产品质量”的Ticket，经常使用
3. 用户一般在使用初期会根据业务需求自定义字段，之后在系统使用过程自定义字段新建和修改会比较少；
4. 用户查询和修改一月之前的Ticket的概率会比最近时间的Ticket的概率低很多，大概1：100；

**问题：**在关系型数据库的基础上，请定义能实现自定义和存储Ticket的表结构；

数据量大，对频繁查询的字段增加索引，扩自定义字段后期维护比较少，分表处理，与工单表建立外键关系。对于旧数据的处理方式为：按创建时间字段，每月进行分区存储。

建立三个表：WorkOrder（工单表），WorkOrderDefine（工单自定义字段），SysDictionarye（字典表，ps:工单自定义时选择下拉框时用到）,建表字段说明前面两个表没有给出，最后一个已给出。

1.WorkOrder（工单表）结构如下：

CREATE TABLE [dbo].[WorkOrder] (

[Id] int NOT NULL IDENTITY(1,1) ,

[Title] varchar(50) COLLATE Chinese\_PRC\_CI\_AS NULL ,--标题

[ServiceManId] int NULL ,-- 负责人Id

[ServiceManIdName] varchar(50) COLLATE Chinese\_PRC\_CI\_AS NULL ,-- 负责人名称

[DepartmentId] int NULL ,--部门Id

[DepartmentName] varchar(50) COLLATE Chinese\_PRC\_CI\_AS NULL ,--部门名称

[State] smallint NULL ,--状态

[CreateDate] datetime NULL ,--创建时间

CONSTRAINT [PK\_\_WorkOrde\_\_3214EC077F60ED59] PRIMARY KEY ([Id])

)

ON [PRIMARY]

GO

CREATE INDEX [WorkOrder\_INDEX] ON [dbo].[WorkOrder]

([ServiceManId] ASC, [DepartmentId] ASC, [State] ASC)

ON [PRIMARY]

GO

2.WorkOrderDefine（工单自定义字段）结构如下：

CREATE TABLE [dbo].[WorkOrderDefine] (

[Id] int NOT NULL IDENTITY(1,1) ,

[WorkOrderId] int NOT NULL ,--外键工单Id

[Type] int NOT NULL ,-- 自定义类型 1.产品、2渠道、3咨

[Field] varchar(50) COLLATE Chinese\_PRC\_CI\_AS NOT NULL ,-- 自定义字段

[FieldName] varchar(50) COLLATE Chinese\_PRC\_CI\_AS NULL ,-- 自定义名称

[FieldType] int NULL ,-- 字段类型 1.文本 2.日期 3.时间 4.字典单选 5.字典多选

[DictionaryCode] varchar(50) COLLATE Chinese\_PRC\_CI\_AS NULL ,-- 字典对应的Code（编号）

[IsEnable] bit NOT NULL ,-- 是否启用

CONSTRAINT [FK\_\_WorkOrder\_\_WorkO\_\_023D5A04] FOREIGN KEY ([WorkOrderId]) REFERENCES [dbo].[WorkOrder] ([Id]) ON DELETE NO ACTION ON UPDATE NO ACTION

)

ON [PRIMARY]

GO

CREATE INDEX [WorkOrderDefine\_INDEX] ON [dbo].[WorkOrderDefine]

([FieldName] ASC)

ON [PRIMARY]

GO

3.SysDictionarye（字典表）结构如下：

CREATE TABLE [dbo].[SysDictionarye] (

[Id] int NOT NULL IDENTITY(1,1) ,

[Code] nvarchar(50) COLLATE Chinese\_PRC\_CI\_AS NOT NULL ,

[Name] nvarchar(50) COLLATE Chinese\_PRC\_CI\_AS NOT NULL ,

[IsEnable] bit NOT NULL ,

[IsDelete] bit NOT NULL ,

CONSTRAINT [PK\_\_SysDicti\_\_3214EC070519C6AF] PRIMARY KEY ([Id])

)

ON [PRIMARY]

GO

IF ((SELECT COUNT(\*) from fn\_listextendedproperty('MS\_Description',

'SCHEMA', N'dbo',

'TABLE', N'SysDictionarye',

'COLUMN', N'Code')) > 0)

EXEC sp\_updateextendedproperty @name = N'MS\_Description', @value = N'编号'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'Code'

ELSE

EXEC sp\_addextendedproperty @name = N'MS\_Description', @value = N'编号'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'Code'

GO

IF ((SELECT COUNT(\*) from fn\_listextendedproperty('MS\_Description',

'SCHEMA', N'dbo',

'TABLE', N'SysDictionarye',

'COLUMN', N'Name')) > 0)

EXEC sp\_updateextendedproperty @name = N'MS\_Description', @value = N'名称'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'Name'

ELSE

EXEC sp\_addextendedproperty @name = N'MS\_Description', @value = N'名称'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'Name'

GO

IF ((SELECT COUNT(\*) from fn\_listextendedproperty('MS\_Description',

'SCHEMA', N'dbo',

'TABLE', N'SysDictionarye',

'COLUMN', N'IsEnable')) > 0)

EXEC sp\_updateextendedproperty @name = N'MS\_Description', @value = N'是否启用'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'IsEnable'

ELSE

EXEC sp\_addextendedproperty @name = N'MS\_Description', @value = N'是否启用'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'IsEnable'

GO

IF ((SELECT COUNT(\*) from fn\_listextendedproperty('MS\_Description',

'SCHEMA', N'dbo',

'TABLE', N'SysDictionarye',

'COLUMN', N'IsDelete')) > 0)

EXEC sp\_updateextendedproperty @name = N'MS\_Description', @value = N'是否删除'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'IsDelete'

ELSE

EXEC sp\_addextendedproperty @name = N'MS\_Description', @value = N'是否删除'

, @level0type = 'SCHEMA', @level0name = N'dbo'

, @level1type = 'TABLE', @level1name = N'SysDictionarye'

, @level2type = 'COLUMN', @level2name = N'IsDelete'

**考点1**: 提供的方案是否能够实现需求 (15分)

A)完全不知道要如何实现(0分)

B)思路上能get几个点，但是给不出具体完整的实现方案(1-5分) **（+5）**

C)能够给出具体完整的实现方案，根据方案的优劣打分（6-15分）

1. 修改表结构，自定义字段跟系统字段一样以列的方式，能够最大化查询效率；

如果提供的是这个方案或者其它高效的方案(12分以上)

b.如果仅仅是把自定义字段的配置信息和用户填写的自定义字段信息用单独的表存起来(8-12分)

c.其它方案，虽然能实现，但是效率低、实现复杂度高(6-8分)

**考点2:** ticket 要根据时间进行分表或把历史数据归档(5分) **（+5）**

**考点3**: 要在经常查询以及要进行关联查询的字段上加索引（5分）**（+5）**

**考点4:** 对于选择类型的字段，在Ticket表应该存放结果值，而不是选项值的Id。提高查询效率（10分）

**总分15 分**

#### 三、设计

在线客服系统是一款基于Web的实时聊天系统，用户帮助企业的客服人员回答网站访客的问题。请根据以下需求场景描述，对系统进行类设计：

1. 聊天（Chat）的双方为访客（Visitor）和客服（Agent）；
2. 客服为系统预设，属性包含Id、Name；
3. 一个聊天只能有一个访客参加，可以有多个客服参加；
4. 聊天由访客请求发起，客服人员接受请求后，双方可以聊天；
5. 聊天过程中，另外的客服人员可以加入；
6. 聊天中访客和客服都可以发送和接收聊天消息（Message）；

类设计要求体现需求描述内容和对象关系，要求包含类的定义和成员；

不限制语言，不要求实现函数体内容，不要求体现需求未描述内容；

类设计如下：

涉及四个类，Chat（聊天实体类），Message（会话消息实体类），Visitor（访客实体类），Agent（客服实体类）

namespace ChatSys

{

/// <summary>

/// 聊天实体类

/// </summary>

public class Chat

{

public Chat()

{

mainagent = new Agent();

listagent = new List<Agent>();

visitor = new Visitor();

listmessage = new List<Message>();

}

/// <summary>

/// 会话Id

/// </summary>

public string chatid { get; set; }

/// <summary>

/// 负责本次会话的客服

/// </summary>

public Agent mainagent { get; set; }

/// <summary>

/// 客服列表

/// </summary>

public List<Agent> listagent { get; set; }

/// <summary>

/// 访问者信息

/// </summary>

public Visitor visitor { get; set; }

/// <summary>

/// 会话开始时间

/// </summary>

public DateTime starttime { get; set; }

/// <summary>

/// 访客是否离开

/// </summary>

public bool isleave { get; set; }

/// <summary>

/// 会话结束时间

/// </summary>

public DateTime endtime { get; set; }

/// <summary>

/// 聊天记录列表

/// </summary>

public List<Message> listmessage { get; set; }

}

/// <summary>

/// 会话消息实体类

/// </summary>

public class Message

{

/// <summary>

/// 消息类型：1客服,2访客

/// </summary>

public int msgtype { get; set; }

/// <summary>

/// msgtype=1 客服Id，msgtype=2 访客Id

/// </summary>

public int Id { get; set; }

/// <summary>

/// msgtype=1 客服名称，msgtype=2 访客名称

/// </summary>

public int Name { get; set; }

/// <summary>

/// 消息内容

/// </summary>

public string Content { get; set; }

/// <summary>

/// 消息时间

/// </summary>

public DateTime CreateDate { get; set; }

}

/// <summary>

/// 访客实体类

/// </summary>

public class Visitor

{

/// <summary>

/// 访客Id

/// </summary>

public int V\_Id { get; set; }

/// <summary>

/// 访客名称

/// </summary>

public string V\_Name { get; set; }

}

/// <summary>

/// 客服实体类

/// </summary>

public class Agent

{

/// <summary>

/// 客服Id

/// </summary>

public int A\_Id { get; set; }

/// <summary>

/// 客服名称

/// </summary>

public string A\_Name { get; set; }

}

}

**考点1:** 有完整的类设计: 包括Chat, Agent, Visitor, Message （10分）**（+10）**

**考点2:** 体现正确的类关系 **（+10）**

* 1. Visitor包含Chat成员 （5分）
  2. Chat包含Agent对象列表，包含ChatMessage列表 （5分）

**考点3:** 聊天由visitor发起,所以visitor应该要有类似StartChat 一类的方法 （5分）

**考点4:** 聊天过程中可以Add Agent, 所以Chat类要有AddAgent 一类的方法 （5分）

**考点5：**Message可抽象为基类，Agent发送的消息和Visitor发送的消息可继承（5分）；

**总分 20 分**