内容

[1 基本概念说明 1](#_Toc408408523)

[2 发送预警 2](#_Toc408408524)

[2.1 固定接收人和无模板参数 2](#_Toc408408525)

[2.2 固定接收人和有模板参数 2](#_Toc408408526)

[2.3 动态接收人和有模板参数 2](#_Toc408408527)

[2.4 工作流中预警发送 2](#_Toc408408528)

[3 预警查询 3](#_Toc408408529)

[3.1 根据登陆用户查询 3](#_Toc408408530)

[3.2 根据登陆用户和其他过滤查询 3](#_Toc408408531)

[3.3 根据预警类型汇总 3](#_Toc408408532)

[3.4 根据预警分类进行汇总 4](#_Toc408408533)

[3.5 设置消息完成 4](#_Toc408408534)

[4 预警缓存 4](#_Toc408408535)

[4.1 接口定义 4](#_Toc408408536)

[4.2 缓存接口实现示例 5](#_Toc408408537)

[4.3 注册接口实现示例 6](#_Toc408408538)

[4.4 从缓存取数据示例 6](#_Toc408408539)

风险预警开发手册

# 基本概念说明

在某些业务场景，需要以UI消息、手机短信、邮件等形式，通知用户比较关心的相关事项，如员工合同到期提醒等。消息以不同的形式（如手机）发送给用户，就需要不同消息模板，我们称这些模板为通道。

在实际的业务场景中，发送一个消息给用户时，可能需要同时以两种以上形式发送，比如合同到期时，可能以UI消息和邮件两种形式同时发送，这种场景下，我们把UI通道和邮件通道打包，称之为一个预警类型，并给预警类型一个编码（如C00001）。那么在发送消息时，用C00001调用API发送消息。

消息最终需要发送到指定的接收人，接收人在系统层面用用户Id表示，在发送时，使用用户Id表示指定的接收人。接收人设定有两种方式固定和动态，固定是指预警类型在配置时，已经指定了接收人。动态指在业务系统发送消息时，才指定接收人。

在不同的业务场景下，同一种通道，而内容可能是不同的，并且内容绝大部分场景中是不固定的，如消息的消息发送日期，必须在发送时刻，由业务决定。因此在发送预警消息时，需要给定模板的参数。模板中动态内容部分，以形如{{Key}}来表示动态填充的部分，如{{SentDate}}表示此内容为发送日期。

# 发送预警

## 固定接收人和无模板参数

在接收人预先配置，并通道模板固定的情况下使用。调用示例代码：

//无模板参数，无接收人

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm= serviceAdapter.GetAlarmService();

Guid id=ialarm.SendAlarm(“C00001”);

## 固定接收人和有模板参数

在接收人预先配置，而模板有动态参数需要填充。调用示例代码：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

IDictionary<string, string> args = new Dictionary<string, string>();

args.Add("SentDate", DateTime.Now);

ialarm.SendAlarm(\_AlarmTypeCode, args);

## 动态接收人和有模板参数

业务系统在发送消息时动态指定接收人，并指定模板参数。调用示例：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

IDictionary<string, string> args = new Dictionary<string, string>();

List<ReceiverDescription> rds = new List<ReceiverDescription>();

rds.Add(new ReceiverDescription() { Param1 = "3223", PeopleEmail =

"57592856@qq.com", PeopleMobile = "13127665611", PeopleName = "安永", UserCode =

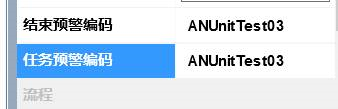
"4EF6BE7E-9300-416D-B390-EBB859A6D05B" });

args.Add("SentDate", DateTime.Now);

ialarm.SendAlarm(\_AlarmTypeCode, args, rds);

## 工作流中预警发送

在流程中，预警发送是有流程引擎调用上面提到的接口实现的。对于流程开发者来说，仅需要配置预警类型编码。工作流上的预警类型编码分两类，一是产生代办时，发送给代办人的消息。二是在流程完成时，发送给起草人的消息。因此，在流程设计器中可看到如下图的流程属性。



流程中的预警使用流程中的上下文参数作为预警模板的参数。在流程提交或审批的过程中，流程引擎会自动调用预警接口发送预警。示例代码：

//OMP可以不设置此属性，默认就是false

//SSP设置为true

paras.EMailCollector.IsTenantMode = true;

业务数据设置，与流程上下文数据设置一致，如：

//加入业务参数

paras.ProcessStartupParams.ApplicationRuntimeParameters["amount"] = expense.Amount;

//在需要改变模板定义的预警类型时，可根据场景，修改下面的预警编码。

//paras.EMailCollector.MailCompletedArguments.AlarmTypeCode = " C00001";

//paras.EMailCollector.MailTaskArguments.AlarmTypeCode = " C00001";

在流程引擎中，提供了一些系统参数供预警模板使用。这些参数有ApplicationName、ProgramName、TaskTitle、Url、ReceiverName、DeliverTime、CreatorName。

# 预警查询

## 根据登陆用户查询

查询当前登录用户的预警信息，此查询的结构是未完成的且不过期的所有消息。示例代码：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

var ais = ialarm.QueryAlarmByUserId("1a212508-3a81-4005-ba5c-0cdf9249ee07", 1, 1000);

## 根据登陆用户和其他过滤查询

除了用户Id，加入更多的查询条件，示例代码：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

MsgSearchCondition msc = new MsgSearchCondition();

msc.IsExpired = false;

msc.FilterBySubsciption = true;

var ais = ialarm.QueryAlarmByUserId("1a212508-3a81-4005-ba5c-0cdf9249ee07", msc, 1, 1000);

## 根据预警类型汇总

以预警类型进行汇总，并可以指定过滤条件，示例代码：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

MsgSearchCondition msc = new MsgSearchCondition();

msc.IsExpired = false;

var ais = ialarm.GetAlarmSummaryByUserId("4EF6BE7E-9300-416D-B390-EBB859A6D05B", msc, 1, 1000);

## 根据预警分类进行汇总

根据预警类型汇总预警消息，并可以指定过滤条件，示例代码：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

//查询Top 无订阅的情况

MsgSearchCondition msc = new MsgSearchCondition();

msc.Status = "000002";

//是否关联订阅查询

msc.FilterBySubsciption = false;

//是否关联订阅租查询

msc.FilterByGroup = false;

List<AlarmDicItem> categories = new List<AlarmDicItem>();

AlarmDicItem a3 = new AlarmDicItem();

a3.CategoryCode = "000002";

AlarmDicItem a4 = new AlarmDicItem();

a4.CategoryCode = "000003";

AlarmDicItem a5 = new AlarmDicItem();

a5.CategoryCode = "000004";

AlarmDicItem a6 = new AlarmDicItem();

a6.CategoryCode = "000005";

categories.Add(a3);

categories.Add(a4);

categories.Add(a5);

categories.Add(a6);

List<AlarmGroupSummary> testResult = ialarm.GetAlarmGroupSummaryByUserId("071FE281-3326-4689-86C1-74327EDB571B", msc, categories, true);

## 设置消息完成

当一个消息被用户处理后，可以将标志设置为已完成，示例代码：

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

ialarm.FinishAlarmByAlarmId(Guid.Parse("E5C3D628-9481-4B40-94E3-2FB74600C165"));

# 预警缓存

有时需要将预警的查询结果缓存，以避免对数据库的频繁访问。预警提供了一个统一的机制，可以缓存所有想要缓存的预警消息，接口名为IAlarmDbLoader。

## 接口定义

interface IAlarmDbLoader

{

/// <summary>

/// 过期时间

/// </summary>

TimeSpan ExpiredTimeSpan { get; set; }

/// <summary>

/// 缓存Key

/// </summary>

string ChacheKey { get; set; }

/// <summary>

/// 在缓存中无数据时，需要加载的业务数据接口

/// </summary>

/// <typeparam name="T">数据类型</typeparam>

/// <param name="userId">用户Id</param>

/// <param name="tenantCode">租户编码</param>

/// <param name="IsTenantMode">是否是多租户模式</param>

/// <returns>预警信息</returns>

T LoadDataFromDb<T>(string userId, string tenantCode, bool IsTenantMode)

}

## 缓存接口实现示例

class OMPHomeTop10AlarmDbLoader : IAlarmDbLoader

{

private string \_CacheKey = CommonConfig.CacheKey\_OMPHomeAlarmTop10;

private TimeSpan \_ExpiredTimeSpan = new TimeSpan(0, 10, 0);

public string ChacheKey

{

get

{

return \_CacheKey;

}

set

{

\_CacheKey = value;

}

}

public TimeSpan ExpiredTimeSpan

{

get

{

return \_ExpiredTimeSpan;

}

set

{

\_ExpiredTimeSpan = value;

}

}

public T LoadDataFromDb<T>(string userId, string tenantCode, bool IsTenantMode)

{

AlarmGroupSummaryEntity model = new AlarmGroupSummaryEntity();

//按照订阅的预警组显示

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

ISubscriptionBizlet ialarmType = serviceAdapter.GetSubscriptionService();

List<AlarmDicItem> categories = ialarmType.GetAvailibleCategories(true, userId);

model.AlarmCategories = categories;

MsgSearchCondition msc = new MsgSearchCondition();

msc.ChannelCode = AlarmItemCode.ChannelCodeForOMPUI;//OMPUI

msc.FilterByGroup = true;//是否关联订阅租查询

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

model.AlarmGroupSummaryList = ialarm.GetAlarmGroupSummaryByUserId(userId, msc, categories, false);

return model as T;

}

}

在LoadDataFromDb中，编写业务欲要调用的预警接口即可，在运行时，预警会自动管理起来这个调用，并将其缓存。

## 注册接口实现示例

在Portal和OugService的Ioc中注册：

sl.Register<IAlarmDbLoader, OMPHomeTop10AlarmDbLoader>(alwaysNew: true);

## 从缓存取数据示例

ServiceAdapterBizlet serviceAdapter = ServiceAdapterBizlet.CreateInstance();

IAlarmBizlet ialarm = serviceAdapter.GetAlarmService();

List<AlarmGroupSummary> alarmSummaries =

ialarm.GetAlarmCached<List<AlarmGroupSummary>>(CacheKeys.HomeTopN,userId);