N-Gitea

基于Gitea的通知扩展

第一次进度汇报

**1、实现功能说明**

* 1. 新的通知方式功能模块总体简介

对于使用者来说，我们可以在以下的两个方面进行通知的改进：通知形式和通知内容。通知形式上，Gitea已经为用户提供了邮件和网页消息两种通知方式。结合国内现状，越来越多的应用开始着力于移动端的功能开发，依托于现有的通用平台进行功能集成更是一种趋势。因此，我们结合移动端的优势，拓展通知的发布渠道。

1.2新的通知方式功能模块的加载和设置功能说明

通知方式功能模块的加载由部署者和管理者进行。部署者可以通过Gitea的安装界面完成模块的加载，管理者可以在管理面板Web界面完成模块加载。本地模块文件被选择并上传后，Gitea会将加载模块请求送至通知服务模块。通知服务模块加载模块文件内容后，校验模块文件的正确性和规范性。通知方式功能模块加载成功后，Gitea会向通知服务模块请求对应模块的全局设置页面代码，渲染后传给前端。部署者完成设置内容填写后，Gitea将收到的设置内容传给通知服务模块，通知服务模块将调用通知方式功能模块的对应方法完成全局设置。

除了全局设置以外，Gitea的每一个使用者都需要完成对应模块的个人设置才能正常使用模块功能。一般流程如下：Gitea向通知服务模块请求对应模块的个人设置界面代码，渲染后传递给前端。使用者填写完成后Gitea将数据回传给通知服务模块。

**2、实现进度**

2.1实现微信通知方式功能模块的加载和设置

2.1.1简要说明

获取个人微信的OpenID，完成Gitea的URL注册，实现Gitea的代理转发。

对于微信通知设置，Gitea需要使用者的OpenId完成通知的分发操作。Gitea需要访问微信服务号，以完成个人OpenId的获取操作。通知方式功能模块在全局设置时可向Gitea注册特定的URL连接，将Gitea作为代理，转发该URL请求中的数据至通知服务模块。这类特殊的功能模块能够通过注册URL的方式实现和外部的通信，完成使用者个人的通知设置。

2.1.2架构图

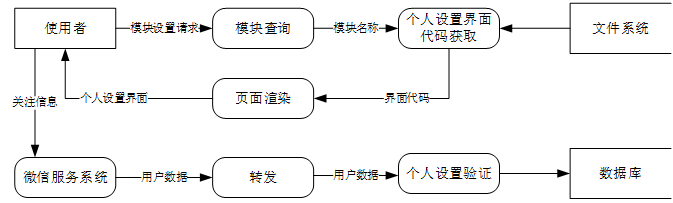


图1微信通知个人设置数据流图

2.1.3具体实现

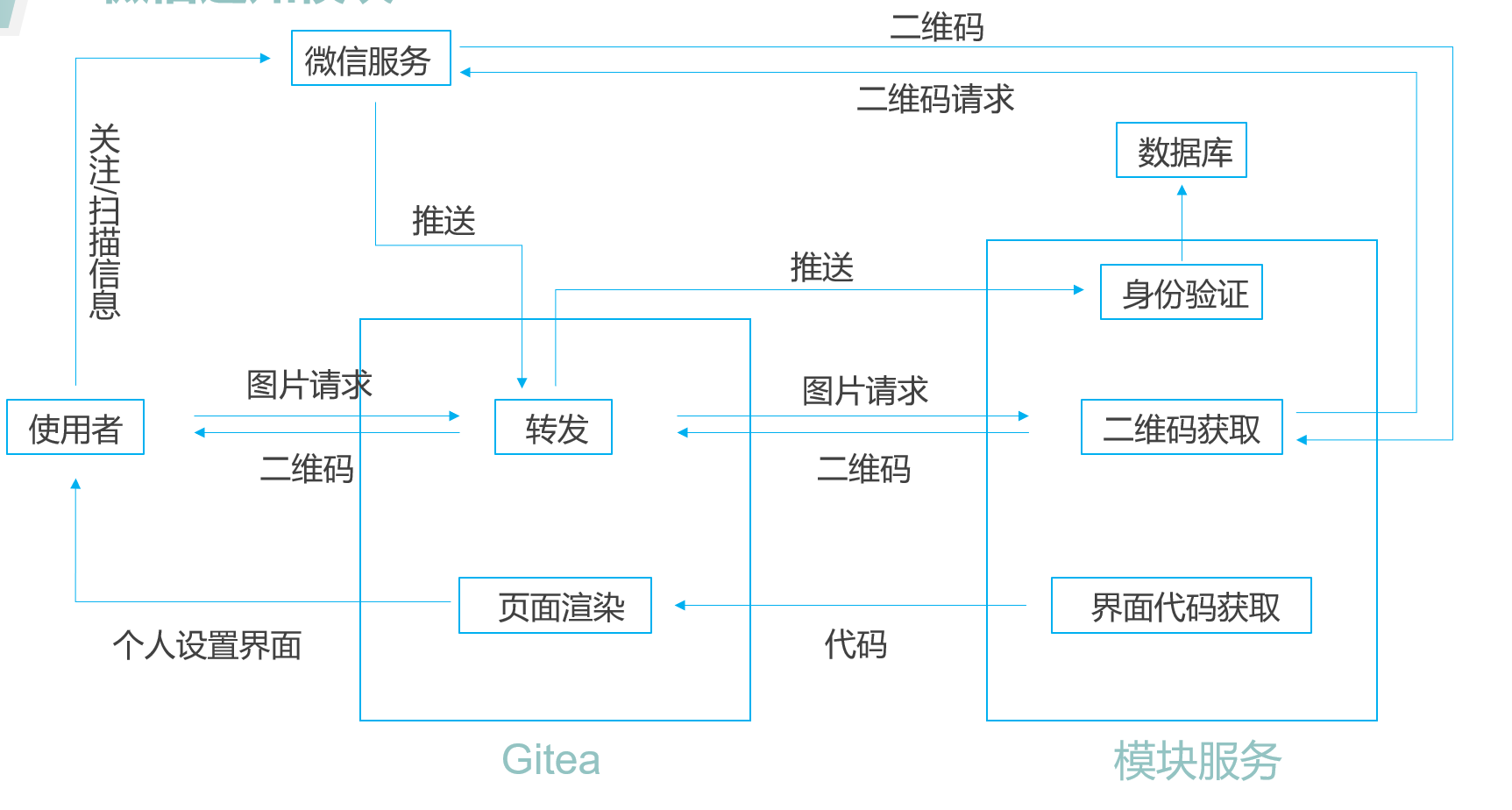


图2 Gitea绑定微信实现图

首先获取设置界面模块代码，在Web界面上渲染生成二维码，用户使用微信对该二维码扫描，就会使用该微信用户登录Gitea，Gitea访问微信服务号，获取OpenId。模块向Gitea注册特定的URL连接，将Gitea作为代理，转发该URL请求中的数据至通知服务模块。这样Gitea便与微信的服务器绑定成功。

2.2实现通知事件分发流程

2.2.1简要说明

通知事件是由使用者动态产生的。使用者可以在通知事件设置界面中针对仓库激活对应的通知事件，并选择Gitea已经注册的通知方式完成通知。当有通知事件时，Gitea会根据事件类型和设置选择需要通知的使用者和对应的通知方式，然后将通知事件以及相应操作发送至通知服务模块。通知服务模块读取个人设置信息后相继调用对应功能模块的格式化和发送代码完成通知的发送。

2.2.2架构图

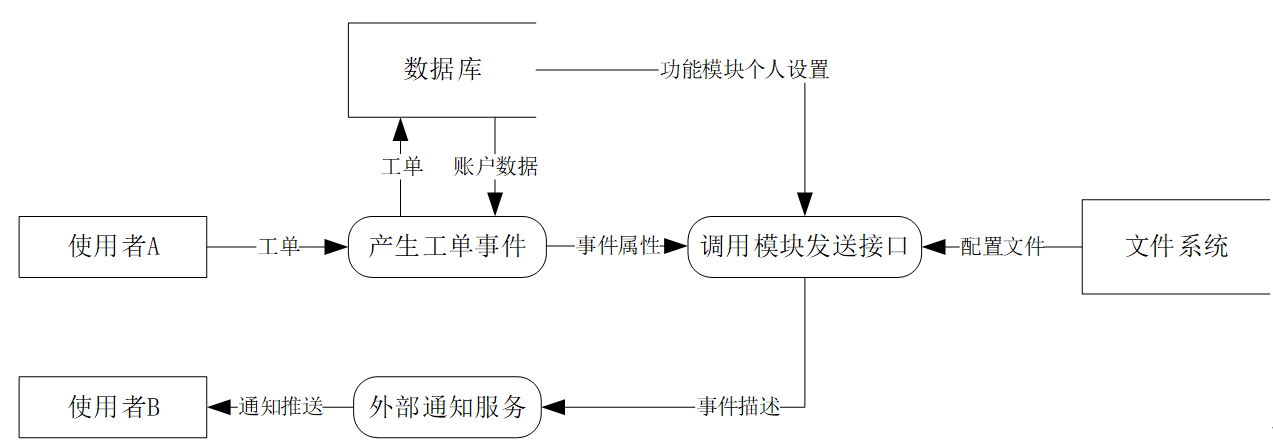


图3通知事件数据流图

2.2.3具体实现

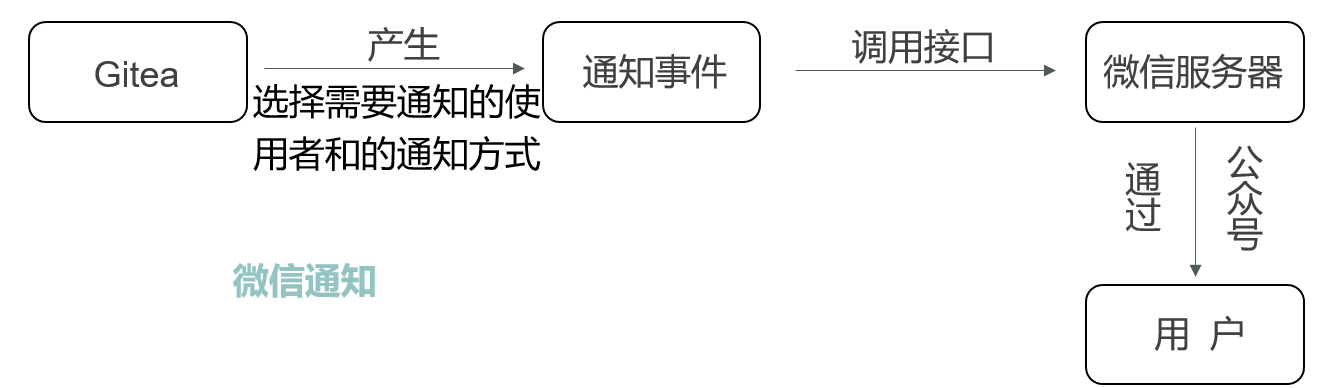


图4微信通知实现图

当有消息通知的时，会触发消息通知过程，会在Gitea端通过notification判断通知的类型及需要通知的使用者，然后调用消息发送模块发送信息，将消息传递到微信服务器，微信服务器使用外部接口通过公众号推送给关注公众号的用户。

**3、附录**

核心代码块

package wechat

import (

    "code.gitea.io/gitea/models"

    "code.gitea.io/gitea/modules/git"

    "code.gitea.io/gitea/modules/log"

    "code.gitea.io/gitea/modules/notification/base"

    "code.gitea.io/gitea/services/wechat"

    "fmt"

)

type (

    notificationService struct {

        base.NullNotifier

        issueQueue chan issueNotificationOpts

    }

    issueNotificationOpts struct {

        issueID              int64

        commentID            int64

        notificationAuthorID int64

    }

)

var (

    \_ base.Notifier = &notificationService{}

)

func NewNotifier() base.Notifier {

    return &notificationService{

        issueQueue: make(chan issueNotificationOpts, 100),

    }

}

func (ns \*notificationService) Run() {

    for opts := range ns.issueQueue {

        if err := sendNotification(opts); err != nil {

            log.Error("Was unable to create issue notification: %v", err)

        }

    }

}

// TODO: generate URL for message

func doSend(issue \*models.Issue, user \*models.User) {

    if user.WechatOpenId != "" {

        msg := fmt.Sprintf("issue: %d, author: %s", issue.ID, user.Name)

        wechat.SendWechatMsg(msg, user.WechatOpenId, "")

    }

}

func sendNotification(opts issueNotificationOpts) error {

    issue, err := models.GetIssueByID(opts.issueID)

    if err != nil {

        return err

    }

    user, err := models.GetUserByID(opts.notificationAuthorID)

    if err != nil {

        return err

    }

    issueWatches, err := models.GetIssueWatchers(opts.issueID)

    if err != nil {

        return err

    }

    watches, err := models.GetWatchers(issue.RepoID)

    if err != nil {

        return err

    }

    alreadyNotified := make(map[int64]struct{}, len(issueWatches)+len(watches))

    notifyUser := func(userID int64) error {

        if userID == opts.notificationAuthorID {

            return nil

        }

        if \_, ok := alreadyNotified[userID]; ok {

            return nil

        }

        alreadyNotified[userID] = struct{}{}

        doSend(issue, user)

        return nil

    }

    for \_, issueWatch := range issueWatches {

        if !issueWatch.IsWatching {

            alreadyNotified[issueWatch.UserID] = struct{}{}

            continue

        }

        if err := notifyUser(issueWatch.UserID); err != nil {

            return err

        }

    }

    if err = issue.LoadRepo(); err != nil {

        return err

    }

    for \_, watch := range watches {

        issue.Repo.Units = nil

        if issue.IsPull && !issue.Repo.CheckUnitUser(watch.UserID, false, models.UnitTypePullRequests) {

            continue

        }

        if !issue.IsPull && !issue.Repo.CheckUnitUser(watch.UserID, false, models.UnitTypeIssues) {

            continue

        }

        if err := notifyUser(watch.UserID); err != nil {

            return err

        }

    }

    return nil

}

func (ns \*notificationService) NotifyCreateIssueComment(doer \*models.User, repo \*models.Repository,

    issue \*models.Issue, comment \*models.Comment) {

    var opts = issueNotificationOpts{

        issueID:              issue.ID,

        notificationAuthorID: doer.ID,

    }

    if comment != nil {

        opts.commentID = comment.ID

    }

    ns.issueQueue <- opts

}

func (ns \*notificationService) NotifyNewIssue(issue \*models.Issue) {

    ns.issueQueue <- issueNotificationOpts{

        issueID:              issue.ID,

        notificationAuthorID: issue.Poster.ID,

    }

}

func (ns \*notificationService) NotifyIssueChangeStatus(doer \*models.User, issue \*models.Issue, actionComment \*models.Comment, isClosed bool) {

    ns.issueQueue <- issueNotificationOpts{

        issueID:              issue.ID,

        notificationAuthorID: issue.Poster.ID,

    }

}

func (ns \*notificationService) NotifyNewPullRequest(pr \*models.PullRequest) {

    ns.issueQueue <- issueNotificationOpts{

        issueID:              pr.Issue.ID,

        notificationAuthorID: pr.Issue.PosterID,

    }

}

func (ns \*notificationService) NotifyPullRequestReview(pr \*models.PullRequest, r \*models.Review, comment \*models.Comment) {

    var opts = issueNotificationOpts{

        issueID:              pr.Issue.ID,

        notificationAuthorID: r.Reviewer.ID,

    }

    if comment != nil {

        opts.commentID = comment.ID

    }

    ns.issueQueue <- opts

}

func (ns \*notificationService) NotifyIssueChangeAssignee(doer \*models.User, issue \*models.Issue, assignee \*models.User, removed bool, comment \*models.Comment) {

    ns.issueQueue <- issueNotificationOpts{

        issueID:              issue.ID,

        notificationAuthorID: doer.ID,

    }

}

func (ns \*notificationService) NotifyMergePullRequest(pr \*models.PullRequest, doer \*models.User, baseRepo \*git.Repository) {

    ns.issueQueue <- issueNotificationOpts{

        issueID:              pr.Issue.ID,

        notificationAuthorID: doer.ID,

    }

}