Gitea需求规格说明书

**[V1.0]**

|  |  |  |
| --- | --- | --- |
|  | 组员 | 日期 |
| 撰写 | 麦梓健 | 2020. |
| 校对 |  |  |

北京航空航天大学计算机学院

二〇一七年三月二十九日

**文档修改记录**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 版本号 | 日期 | 修改章节 | 修改说明 | 修改人 | 审核人 |
| 1.00 | 2020.3.21 |  | 初稿完成 | 麦梓健 | 麦梓健 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# 1 引言

Gitea是轻量级的代码托管平台。本文主要分析Gitea的核心功能模块、非功能模块和业务逻辑，同时介绍改进需求和拓展模块。

## 1.1 编写目的

本软件需求规格说明书，是为软件设计、软件测试人员和用户编写的。

本软件需求规格说明书的适用者，包括开发测试人员、Gitea技术人员、项目其他相关人员以及Gitea的使用群体。

## 1.2 背景

软件名称：Gitea

项目组织机构：Gitea项目开发组和社区组织

项目实施机构：Gitea社区上的开发组

项目背景：

1.Git的出现：高速、简单的设计、对非线性开发模式的强力支持（允许成千上万个并行开发的分支）、完全分布式、有能力高效管理类似 Linux 内核一样的超大规模项目（速度和数据量）。

2. GitHub、GitLab等基于Git的代码托管平台，尽管功能强大，但是过重、过于复杂了，对于许多项目而言并不需要如此庞大的体系，因而轻量级的框架gogs等相继出现。

3. Gogs简单、轻便、运行快，只需要廉价的树莓派即可满足需求。但是gogs由于“独裁”性质的维护，使得由社区维护的gitea出现了。

## 1.3 定义

|  |  |
| --- | --- |
| Git钩子 | Git钩子是在Git仓库中特定事件发生时自动运行的脚本。可以定制一些钩子，这些钩子可以在特定的情况下被执行，分为Client端的钩子和Server端的钩子。 |
| Wiki | Wiki是一种在网络上开放且可供多人协同创作的超文本系统，由沃德·坎宁安于1995年首先开发，这种超文本系统支持面向社群的协作式写作，同时也包括一组支持这种写作 |
| 镜像仓库 | 用来保存制作好的容器的一个仓库，只不过这里的容器被做成了镜像，所以我们称其为镜像仓库。 |
| SSH | SSH 为建立在应用层基础上的安全协议。SSH 是较可靠，专为远程登录会话和其他网络服务提供安全性的协议 |
| Session | 会话是一种持久网络协议，在用户端和服务器端之间建立关联，从而起到交换数据包的作用机制。 |
| pull | 将数据从远程分支拉取到本地，取回远程主机某个分支的更新，再与本地的指定分支合并。其完整命令：git pull <远程主机名> <远程分支名>:<本地分支名>。 |
| push | 将数据从本地仓库上传到远程仓库。git push的一般形式为 git push <远程主机名> <本地分支名> <远程分支名> ，例如 git push origin master：refs/for/master ，即是将本地的master分支推送到远程主机origin上的对应master分支， origin 是远程主机名。第一个master是本地分支名，第二个master是远程分支名。 |
| add | git add [参数] <路径>　作用就是将我们需要提交的代码从工作区添加到暂存区，就是告诉git系统，我们要提交哪些文件，之后就可以使用git commit命令进行提交了。 |
| merge | 将数据拉取到本地仓库，并与相应分支进行融合，不清除旧版本。开发分支（dev）上的代码达到上线的标准后，要合并到 master 分支或者当master代码改动了，需要更新开发分支（dev）上的代码 |
| cache | Git的数据缓存，用于撤销提交等 |
| migration | 数据迁移。 |
| mock | mock测试就是在测试过程中，对于某些不容易构造或者不容易获取的对象，用一个虚拟的对象来创建以便测试的测试方法。这个虚拟的对象就是mock对象。mock对象就是真实对象在调试期间的代替品。 |

## 1.4 参考技术标准

参照如下标准：

* GB/T 19003-2008 软件工程
* GB/T 5538-1995 软件工程标准分类法
* GB/T 9385-2008 计算机软件需求规格说明
* GB/T 5532-2008 计算机软件测试规范
* GB/T 8567-2006 计算机软件文档编制规范

# 2 项目内容

## 2.1 目标

Gitea的出现是为了方便个人或者小型团队开发时进行方便快捷的代码管理和托管。一方面，Gitea同GitHub和GitLab相比减去了大量极少使用的代码模块，使得部署和运行要求大大下降，速度大为上升。另一方面，Gitea作为社区维护的开源项目，不仅可以依据需求选择版本，还可以根据个人和团队的要求进行方便的扩展开发。

## 2.2 用户特点

1. 有代码管理要求的个人和小型团队，且同时对服务器部署有速度等要求。
2. 对代码管理有独特要求、需要对已有框架进行扩展的开发团队

# 3 需求分析

## 3.1 需求

### 3.1.1 业务需求

### 3.1.2 管理需求

### 3.1.3 非功能需求

## 3.2 框架及组件描述

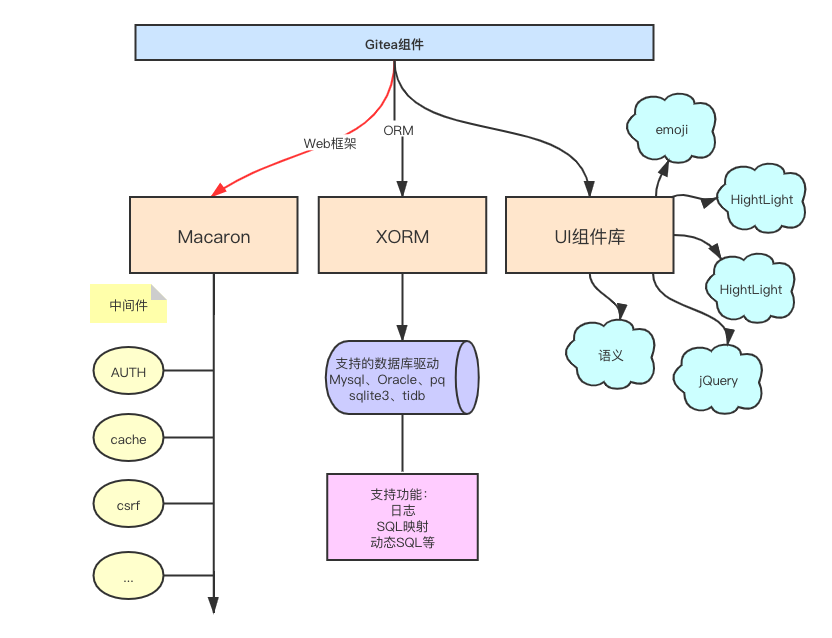


图 1 Gitea框架和组件图

参考文档：

<https://go-macaron.com/middlewares>

<https://docs.gitea.io/en-us/>

### 3.2.1 Macaron框架

Macaron框架是Go专用的一个快速开发工具，其具有强大功能：

1. 支持子路由的强大路由设计
2. 支持灵活多变的路由组合
3. 支持无限路由组的无限嵌套
4. 支持直接集成现有的服务
5. 支持运行时动态设置需要渲染的模板集
6. 支持对模块的轻松接入与解除
7. 采用 inject 提供的便利的依赖注入
8. 采用更好的路由层和更少的反射来提升执行速度

同时，Macaron本身已经集成了大量的中间件，包括

1. auth - HTTP 基本身份验证
2. bindata –静态文件和模版文件
3. binding – 数据绑定和验证
4. cache – 高速缓存服务
5. captcha – 验证码服务
6. csrf – 生成和验证令牌
7. i18n – 国际化和本地化
8. inject – 依赖注入
9. permissions2 - Cookies, users 和permissions
10. pongo2 - Pongo2 引擎
11. renders – 动态渲染
12. session – 会话管理
13. sockets – 套接字管理

除此之外，还有压缩、Oauth等等的中间件。

### 3.2.2 XORM库

xorm是一个简单而强大的Go语言ORM库. 通过它可以使数据库操作非常简便。它同样具有强大的功能：

1. 支持Struct和数据库表之间的灵活映射，并支持自动同步
2. 事务支持，支持嵌套事务（支持类JAVA Spring的事务传播机制）
3. 同时支持原始SQL语句和ORM操作的混合执行
4. 支持级联加载Struct
5. 支持类ibatis方式配置SQL语句
6. 支持动态SQL功能
7. 支持数据库查询结果直接返回Json字符串和xml字符串
8. 支持SqlMap配置文件和SqlTemplate模板密文存储和解析
9. 支持缓存
10. 支持主从数据库(Master/Slave)数据库读写分离
11. 支持记录版本（即乐观锁）
12. 支持查询结果集导出csv、tsv、xml、json、xlsx、yaml、html功能

使用XORM库可以轻松的描述数据库数据存储结构和对象之间的关系，同时简单的实现各种所需的数据操作

### 3.2.3 UI组件库

UI组件库是由社区贡献者们提供的各种各样的特色功能插件，包括emoji文字、Github的图标、渲染文字、jQuery的时间器等等，能够满足特定的使用者群体的功能。

## 3.3 用例图

在实际环境中，使用者需要自行部署服务器，其用例图如下图2：

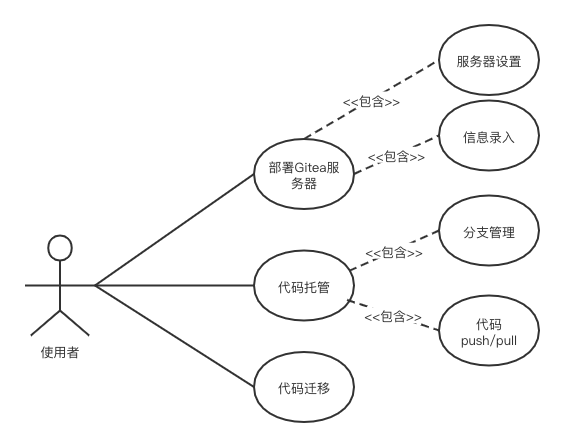


图 2 使用者用例图

## 3.4 功能模块描述

对于用例图中的功能的涉及模块的描述：

## 3.5 其他事务模块描述

### 3.5.1 日志模块

日志模块由XORM模块承担。

日志是一个接口，通过设置日志，可以显示SQL，警告以及错误等，默认的显示级别为INFO。可以设置日志的显示方式，这里总结为三种：

1. 控制台打印SQL语句

2. 控制台打印调试信息

3. 输出调试信息到文件中

1. engine.ShowSQL(**true**) // 会在控制台打印出生成的SQL语句；
2. engine.Logger().SetLevel(core.LOG\_DEBUG) // 会在控制台打印调试及以上的信息；
3. NewSimpleLogger(w io.Writer) // 将信息不仅打印到控制台，而且保存为文件，接收一个io.Writer接口来将数据写入到对应的设施中

### 3.5.2 Toolbox工具箱

Toolbox进一步集成了大量的小型功能，比如进行服务器运行状况的健康检查，配置文件服务以及统计服务。

该中间件可以单独使用命令行导入，也可以在开发环境进行导入

1. import (
2. "gopkg.in/macaron.v1"
3. "github.com/go-macaron/toolbox"
4. )
6. func main() {
7. m := macaron.Classic()
8. m.Use(toolbox.Toolboxer(m))
9. m.Run()
10. }
11. m.Use(toolbox.Toolboxer(m, toolbox.Options{
12. URLPrefix:      "/debug",       // 工具箱的URL前缀
13. HealthCheckURL: "/healthcheck",     // 健康检查请求的URL前缀
14. HealthCheckers: []HealthChecker{...},   // 健康检查的入口
15. HealthCheckFuncs: []\*toolbox.HealthCheckFuncDesc{...}, // 健康检查程序
16. DisableDebug:**false**,             // 当为true时关闭所有调试功能
17. PprofURLPrefix: "/debug/pprof/",    // pprof的URL前缀
18. ProfileURLPrefix:   "/debug/profile/",  // 配置文件的URL前缀
19. ProfilePath:        "profile",          // 路径配置
20. }))

### 3.5.3 缓存

xorm内置了一致性缓存支持，不过默认并没有开启。要开启缓存，需要在engine创建完后进行配置。

1. cher := xorm.NewLRUCacher(xorm.NewMemoryStore(), 1000)
2. engine.SetDefaultCacher(cacher)

上述代码采用了LRU算法的一个缓存，缓存方式是存放到内存中，缓存struct的记录数为1000条，缓存针对的范围是所有具有主键的表，没有主键的表中的数据将不会被缓存。

### 3.5.4 模块切换器

模块切换器为Macaron提供了主机切换功能。即在同一个程序中运行多个Macaron实例。

1. func main() {
2. m1 := macaron.Classic()
3. m2 := macaron.Classic()
5. hs := switcher.NewHostSwitcher()
7. hs.Set("gowalker.org", m1)
8. hs.Set("gogs.io", m2)
10. hs.Run()
11. }

如果这两个实例并不是平行关系而是父子关系，模块切换器一样可以达到效果。

1. m := macaron.Classic()
3. hs := macaron.NewHostSwitcher()
5. hs.Set("\*.example.com", m)
6. hs.Run()

### 3.5.5 bindata

使用bindata可以将模版和公共目录转换为单独的包。

1. // 将该目录转换为单独的包
2. m.Use(macaron.Static("public",
3. macaron.StaticOptions{
4. FileSystem: bindata.Static(bindata.Options{
5. Asset:      **public**.Asset,
6. AssetDir:   **public**.AssetDir,
7. AssetNames: **public**.AssetNames,
8. Prefix:     "",
9. }),
10. },
11. ))

## 3.6 异常处理

Git模块的推拉等操作中出现的错误，会上抛到gitea.log.error中进行错误记录和处理。同样的，还有lexer模块中的一些异常也是。实质上，gitea.log.error最终调用的还是fmt.error或者error.news。

Webhook模块的交付任务中，如果最终无法交付，则会抛到fmt.error中进行处理。同样的，password模块、记录测试模块以及bindata模块中出现的诸如文档类型不匹配、文档指针空的情况都会上抛到fmt.error进行处理。

# 4 运行环境说明

## 4.1 部署环境

系统：Linux、MacOS、Windows，以及任何可以运行Go的环境

开发环境：Go

## 4.2 接口

### 4.2.1 硬件接口

### 4.2.2 软件接口

Go的类库

### 4.2.3 通信接口

HTTPS协议、SSH协议

### 4.2.4 用户接口

命令行工具：通过gitea命令行工具进行控制

# 5. 扩展模块设计与实现方案

## 5.1 需求分析

## 5.2 设计思路

## 5.3 实现方案