链滴

Wide 开发指南

作者：88250

原文链接：https://ld246.com/article/1538876422995

来源网站：链滴

文档概要

该文档介绍了 Wide 的设计原理与核心实现。如果您要读 Wide 的源码或者要进行修改、二次开发等请先仔细阅读该文档。

架构

本章将介绍 Wide 的架构概要，说明 Wide 是如何完成一个 IDE 的核心功能。

构建 & 运行

- 一个浏览器 tab 对应一个 Wide 会话
- 通过 WebSocket 进行程序执行输出推送

1. 客户端浏览器发送 Build 请求
2. 服务器使用 os/exec 执行 go build 命令
2.1. 生成可执行文件
3. 客户端浏览器发送 Run 请求
4. 服务器使用 os/exec 执行文件
4.1. 生成进程
4.2. 运行结果输出到 WebSocket 通道
5. 客户端浏览器监听 ws.onmessage 到消息后做展现

代码辅助

- 自动完成
- 查找使用

1. 浏览器客户端发送代码辅助请求
2. Handler 根据请求对应的 HTTP 会话获取用户工作空间
3. 执行 gocode/ide_stub(gotools) 命令
3.1 设置环境变量（${GOPATH} 为用户工作空间路径）
3.2 gocode 命令需要设置参数 lib-path

关键设计

本章将介绍 Wide 中的关键设计，以及一些实现细节。

工作空间

Wide 是一个多用户的集成开发环境，每个用户的工作空间是完全隔离的：

- 文件路径
- 环境变量

例如在执行“构建 & 运行”时，Wide 会设置命令 go build 执行的 $GOPATH 为用户的工作空间，样编译过程就是在该用户的工作空间中进行，保证其隔离性。

会话

在 UI 上存在两种 tab：浏览器 tab；编辑器 tab。

用户可以在一个浏览器 tab 中打开多个编辑器 tabs，但输出窗口只有一个。这意味着一个浏览器 tab 只能有一个正在运行的用户代码的程序进程。

需要同时运行多个程序进程的场景可以通过打开多个浏览器 tabs 达成。

- 一个浏览器 tab 对应一个 Wide 会话：只要打开/刷新一个 tab，就会新建一个 Wide 会话
- Wide 会话的 id 在源码中统一命名为 sid
- 在需要使用 HTTP 会话的地方统一明确表明是在使用 HTTP 会话，而不是在使用 Wide 会话

WebSocket

WebSocket 用于后端推送数据给前端：

- 输出通道 OutputWS
- 通知通道 NotificationWS
- Shell 通道 ShellWS
- 会话通道SessionWS

通道和会话一对一关联，也就是说如果用户新开浏览器 tab，则新建通道和其关联。

其中_会话通道_用于会话管理，后面会单独介绍。

事件与通知

每个会话都有一个对应的事件队列，当接收到事件时取出该事件并转为通知，再通过通知窗口通道推给前端。

还存在一个全局事件队列，入队的事件将分发到每个用户的事件队列中，以便进行通知广播。

通知中的消息需要国际化处理。在语言配置文件中以 notification_ 为前缀，事件代码（event code 为后缀的项即通知消息。

代码辅助

- 自动完成，通过 gocode 工具实现
- 跳转到声明，通过 gotools 工具实现
- 查找使用，通过 gotools 工具实现

代码辅助功能满足工作空间隔离。

会话管理

本章我们将说明 Wide 会话的生命周期以及和它相关的服务器端状态管理。

服务器端状态

一般 Web 应用的服务器端都是“无状态”的设计：一次 HTTP 请求-响应完成一个操作，下一个操和这次的操作没有任何直接的关联。

比如“用户注册”一般都是在一个请求的生命周期内完成，即使不能完成，也需要服务器端持久化状，“下一步”时查询该状态并继续进行操作。

这样的“无状态”设计这主要是因为：

- 通讯协议受限：Web 应用（网站）都是基于 HTTP 协议，而 HTTP 是无状态协议
- 状态难以抽象：“有状态”设计比较复杂，状态的具体表现在不同应用（甚至是不同业务）中是完不一样的，很难抽取出共性
- 性能考量：“有状态“应用会占用大量系统资源，对于在线用户量大的网站是不可接受的
- “无状态80%”：大多数需求都可以通过“无状态”设计来实现，而且能够工作得很好

Wide 会话

Wide 不是一个 Web 网站，它对服务器端的状态管理有特殊的需求。

生命周期

在前面我们提到过 Wide 会话和浏览器 tab 的关系，这定义了 Wide 会话的生命周期和浏览器 tab 一致的：

- 新建 tab 时（刷新也认为是新建），服务器端创建会话
- tab 处于打开时，服务器端维持会话
- 关闭 tab 时，服务器端销毁会话

会话状态

一个 Wide 会话包含如下信息：

- 关联的 HTTP 会话
- 关联的 事件队列
- 关联的 运行进程集
- 当前会话状态：Active/Closed
- 创建时间
- 最近一次使用时间：收到事件、运行进程都会更新这个时间

销毁一个会话时，需要将其相关的资源回收，避免泄漏。

实现

服务器端使用 WebSocket 建立同 tab 的通讯通道，通道关闭时销毁服务器端会话。

会话还原

当再次打开 IDE 时我们需要还原最近一次的“会话内容”：

- 最近一次打开的文件（编辑器 tabs）
- 当前编辑器
- 最近一次文件树展开状态

准实时保存

- 前端每 30 秒发送一次会话内容保存请求
- 后端每 1 分钟持久化一次所有会话内容到 {username}.json

当一个用户存在有多个会话（打开多个浏览器 tabs）时，还是以该策略进行保存（顺时覆盖）。

编码规范

本章描述了 Wide 项目中编码时的一些细节事项。

关于 Golang 公共的编码规范请参考《Golang 编码规范》。

注释

必须遵守：

- m1：优先使用 **英文**进行注释。目前后端都是使用英文注释，前端部分是使用中文注释
- m2：优先使用行注释 //

尽量遵守（处女座特质）：

- s1：在中文和英文、数字中间插入一个空格
- s2：使用英文标点符号
- s3：注释符号后空格再写文本，例如 // xxxx 而不是 //xxxx
- s4：需要预格式化（ <pre>）的使用 1 个空格进行缩进（参考 Go 编码规范）

示例：

```
// HTTP Handler 包装.
// 完成共性处理：
//  1. panic recover
//  2. 请求计时
```

```
func handlerWrapper(f func(w http.ResponseWriter, r *http.Request)) func(w http.ResponseWr
ter, r *http.Request) {
    handler := panicRecover(f)
    handler = stopwatch(handler)

    return handler
}
```

Handler 包装

在 main.go 中加入新的 HTTP Handler 时需要使用 handlerWrapper 函数进行包装，以便做一些共处理：

- panic recover
- 请求处理计时