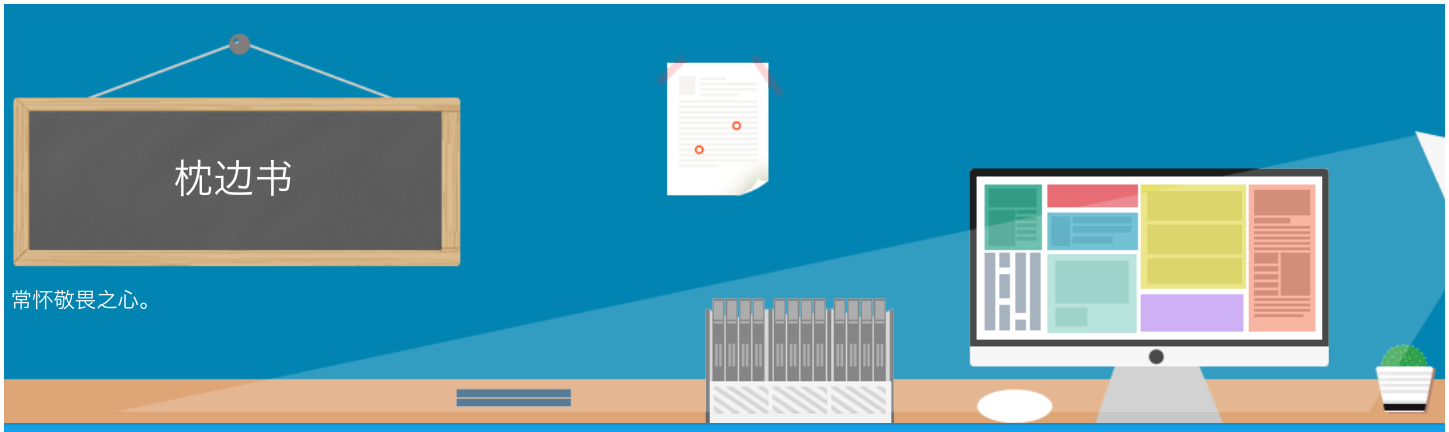




# PHP 调用 Go 服务的正确方式 - Unix Domain Sockets

## 问题

可能是由于经验太少，工作中经常会遇到问题，探究和解决问题的过程总想记录一下，所以我写博客经常是问题驱动，首先介绍一下今天要解决的问题：

**服务耦合**

我们在开发过程中可能会遇到这样的情况：

- 进程依赖于某服务，所以把服务耦合在进程代码中；
- 服务初始化耗时长，拖慢了进程启动时间；
- 服务运行要占用大量内存，多进程时内存损耗严重。

如我上篇文章 小时到分钟 - 一步步优化巨量关键词的匹配 中介绍的文本匹配服务，它是消息处理流程中的一环，被多个消息处理进程依赖，每次初始化进程要 `6秒` 左右时间构造 Trie 树，而且服务读取关键词大文件、使用树组构造 Trie 树，会占用大量(目前设置为 `256M` )内存。

我已经把进程写成了守护进程的形式，让它们长时间执行，虽然不用更多地考虑初始化时间了，但占用内存量巨大的问题没有办法。如果关键词量再大一些，一台机器上面跑十来个消息处理进程后就干不了其他了。

而且，如果有需求让我把文本匹配服务封装为接口给外部调用呢？我们知道，web 服务时，每一个请求处理进程的生存周期是从受理请求到响应结束，如果每次请求都用大量内存和时间来初始化服务，那接口响应时间和服务器压力可想而知。

**服务抽取**

这样，服务形式必须要改变，我们希望这个文本匹配这个服务能做到：

- 随调随走，不依赖，不再与“消息处理服务”耦合在一起；
- 一次初始化，进程运行期间持续提供服务；
- 同步响应，高效而准确，最好能不用各种锁来保持资源占有；

解决办法也很简单，就是把这个文本匹配的服务抽取出来，单独作为一个守护进程来运行，像一个特殊的服务器，多个“消息处理服务”在有需要时能调用此服务进程。

现在，我们需要考虑文本匹配服务进程如何与外界通信，接受匹配请求，响应匹配结果。绕来绕去，问题还是回到了 `进程间通信`。

## Unix Domain Sockets

**进程间通信**

> 进程间通信（IPC，Inter-Process Communication），指至少两个进程或线程间传送数据或信号的一些技术或方法。进程是计算机系统分配资源的最小单位(严格说来是线程)。每个进程都有自己的一部分独立的系统资源，彼此是隔离的。为了能使不同的进程互相访问资源并进行协调工作，才有了进程间通信。

进程间通信的方式有很多，网上对此介绍的也很多，下面根据文章的需求来分析一下这些方式：

- 管道：管道是Unix最初的IPC形式，但它只能用于具有共同祖先进程的各个进程，无法用于在没有亲缘关系的进程。如果使用它，需要在“消息处理服务”中启动“文本匹配服务”，跟原来差别不大。
- 命名管道：也被称为有名管道，它在Unix称为 `FIFO`，它通过一个文件来进行进程间数据交互，但服务于多个进程时，需要添加锁来保证原子性，从而避免写入和读取不对应。
- 信号和信号量：用于进程/线程事件级的通信，但它们能交流的信息太少。
- 消息队列和共享内存：都是通过一个公共内存介质来进行通信，我之前也写过一篇关于PHP进程间使用消息队列和共享内存通信的文章： 从并发处理谈PHP进程间通信（二）System V IPC，但它们在通信上都是异步的，处理多个进程时无法分辨请求和对应的响应信息。
- socket：通过Unix封装好的网络API来进行通信，像数据库、服务器都是通过这种方式实现，它们也能提供本地服务。不过网络socket固然能使用，但是要面临着数据包装和网络调用开销，也不是完美的选择。

### 简单介绍

当然还是有完美的方式的，这就是今天的主角 - `Unix Domain Sockets`，它可以理解为一种特殊的 Socket，但它不需要经过网络协议栈，不需要打包拆包、计算校验和、维护序号和应答等，只是将应用层数据从一个进程拷贝到另一个进程，所以在系统内通信效率更高。而且免去了网络问题，它也更能保证消息的完整性，既不会丢失也不会顺序错乱。

作为特殊的 Socket，它的创建、调用方式和网络 Socket 一样，一次完整的交互，服务端都要经过 `create、bind、listen、accept、read、write`，客户端要通过 `create、connect、write、read`。与普通 Socket 不同的是它绑定一个系统内的文件，而不是 IP 和端口。

创建代码这里不再多介绍了，之前的一篇文章 用C写一个web服务器（一） 基础功能 的 `功能实现` 小节里详细介绍了 socket 通信的具体步骤，C 系的语言都是相似的，很容易理解。

### 适用场景

Unix Domain Sockets 真的是进程间通信的一个重型武器，用它可以快速实现进程间的数据、信息交互，而且不需要锁等繁杂操作，也不用考虑效率，可谓是简单高效。

当然，“重型武器” 的在各种场景下也有适合不适合。Unix Domain Sockets适用于以下场景：

- 服务长时间存在。 Unix Domain Sockets 的服务端是个服务器一样的存在，在守护进程中，它阻塞并等待客户端连接的特性可以被充分利用。
- 一服务器多客户端。它能通过 Socket 的文件描述符来区分不同的客户端，避免资源之间的锁操作。
- 同一系统内。它只能在同一系统内进行进程数据复制，跨系统请使用传统 Sockets。

---

## 代码实现

接下来要 show code 了，不过学 PHP 的都知道，PHP 不太适合处理 CPU 密集形的任务，我刚好学了点 Go，一时手痒，就用 Go 实现了下 Trie 树，所以才牵扯到 PHP 和 Go 之间的通信，有了今天的文章。当然介绍的方法，并不只适合 PHP 与 Go 通信，其他语言也可以，至少 C系语言中是通用的。

完整代码见 IPC-GitHub-枕边书，里面还附带了一份随手写的 PHP 版本的 Unix Domain Sockets server 端。

### Go 实现的 Trie 树

Trie树不再是今天的主题，这里介绍一下数据结构和需要注意的点。

```
// trie树结点定义
type Node struct {
    depth    int
    children map[int32]Node // 用map实现key-value型的 字符-节点 对应
}
```

需要注意：

- 使用 slice 的 `append()` 函数保存递增的匹配结果时，有可能由于 slice 容量不够而重新分配地址，所以要传入 slice 的地址来保存递增后的匹配结果结果，`*result = append(*result, word)`，最后再将递增之后的 slice 地址传回。
- 由于 Go 中的编码统一使用的 `utf-8`，不用像 PHP 一样判断字符的边界，所以在进行关键词拆散和消息拆散时，直接使用 `int32()` 方法将关键词和消息都转换为成员为 `int32` 类型的 slice，匹配过程中就使用 `int32` 类型的数字来代表这个中文字符，匹配完成后再使用 `fmt.Printf("%c", int32)` 将其转换为中文。

### Go Server

Go 中创建一个 socket 并使用的步骤非常简单，只是 Go 没有异常，判断 error 会比较恶心一点，不知道有没有大神有更好的写法。下面为了精简，把 error 全置空了。

```
// 创建一个Unix domain soceket
socket, _ := net.Listen("unix", "/tmp/keyword_match.sock")
// 关闭时删除绑定的文件
defer syscall.Unlink("/tmp/keyword_match.sock")
// 无限循环监听和受理客户端请求
for {
```

```
            client, _ := socket.Accept()

            buf := make([]byte, 1024)
            data_len, _ := client.Read(buf)
            data := buf[0:data_len]
            msg := string(data)

            matched := trie.Match(tree, msg)
            response := []byte("[]") // 给响应一个默认值
            if len(matched) > 0 {
                json_str, _ := json.Marshal(matched)
                response = []byte(string(json_str))
            }
            _, _ = client.Write(response)
        }
```

### PHP Client

下面是 PHP 实现的客户端：

```
$msg = "msg";
// 创建 连接 发送消息 接收响应 关闭连接
$socket = socket_create(AF_UNIX, SOCK_STREAM, 0);
socket_connect($socket, '/tmp/keyword_match.sock');
socket_send($socket, $msg, strlen($msg), 0);
$response = socket_read($socket, 1024);
socket_close($socket);

// 有值则为匹配成功
if (strlen($response) > 3) {
    var_dump($response);
}
```

## 小结

### 效率

这里总结一下这套设计的效率表现：

纯粹用 Go 进行文本关键词匹配，一千条数据运行一秒多，差不多是 PHP 效率的两倍。不过说好的 8倍效率呢？果然测评都是骗人的。当然，也可能是我写法有问题或者 Trie 树不在 Go 的发挥范围之内。然后是 PHP 使用 Unix Domain Socket 调用 Go 服务的耗时，可能是进程间复制数据耗时或 PHP 拖了后腿，3秒多一点，跟纯 PHP 脚本差不多。

### 杂谈

用 PHP 的都知道，PHP 因为解释型语言的特性和其高度的封装，导致其虽然在开发上速度很快，可是执行与其他语言相比略差。对此，业界的 FB 有 HHVM，PHP7 有 opcache 新特性，据说还要在 PHP8 添加 JIT，用以弥补其先天硬伤。

不过，对于开发者，特别是跟我一样对于效率有执著追求的人来说，在了解使用 PHP 的新特性之外，自己再掌握一门较高执行效率、开发效率略低的语言，用来写一些高计算量，逻辑单一的代码，与 PHP 互补或许会更好一点。

于是，在考虑良久，也见识了各种 Go 的支持者和反对者之间的撕逼后，我觉得还是要相信一下谷歌爸爸，毕竟也没什么其他我觉得可选的语言了。PS：请不要针对这一段发表意见，谢谢：）

另外C呢，虽然暂时开发中用不到，可是毕竟是当代N多语言的起源，偶尔写写数据结构、算法什么的以免生锈。而且学了些C，从 PHP 到 Go，切换起来还略有些得心应手的感觉~

关于本文有什么问题可以在下面留言交流，如果您觉得本文对您有帮助，可以点击下面的 **推荐** 支持一下我。博客一直在更新，欢迎 **关注** 。

分类： 后端

**标签：** PHP， Go， Unix Domain Sockets， IPC， 进程间通信



枕边书
关注 - 18
粉丝 - 274
荣誉：推荐博客
+加关注



« 上一篇：小时到分钟 - 一步步优化巨量关键词的匹配
» 下一篇：设计模式，Let's “Go”！ (上)

1. 小时到分钟 - 一步
词的匹配(86)
2. Linux - 请允许我
6)
3. linux的“自动化”(
4. 网页实时聊天之js
x长轮询(10)
5. Redis “瘦身”指南
6. Gotorch - 多机定
(10)
7. 网页实时聊天之P
et(10)
8. 搭建自己的PHP
9. 扩充你的工具箱
(9)
10. 初探PHP多进程