Go CAS认证实现 gin+cas认证 单点登录

背景

在用golang对接cas认证时，发现网上的资料都是使用Golang的 cas 客户端的包：gopkg.in/cas.v2实现。经测试，在接入 cas 服务器后，不断跳转，重定向而不继续执行认证成功后的操作

为了解决该问题，基于cas认证原理，自己实现了一个cas认证客户端

定义响应结构体

首先，我们需要定义 CAS 认证成功后的响应结构体：

```
// model/cas.go
type CasServiceResponse struct {
XMLName xml.Name `xml:"serviceResponse"`
Data    struct {
SFRZH       string `xml:"user"`
Attributes struct {
Uid       string `xml:"uid"`
UserName string `xml:"userName"`
} `xml:"attributes"`
} `xml:"authenticationSuccess"`
}
```

编写 CAS 认证逻辑

接下来，我们编写 CAS 认证的核心逻辑：

```
// utils/cas.go
package utils

import (
"encoding/xml"
"errors"
"fmt"
"github.com/gin-gonic/gin"
"go.uber.org/zap"
"io"
"net/http"
"roomlive-go/global"
"roomlive-go/model/cas"
"roomlive-go/model/user"
"strings"
)

func IsAuthentication(w http.ResponseWriter, r *http.Request,
casServerUrl string) (bool, *cas.CasServiceResponse) {
if !hasTicket(r) {
redirectToCasServer(w, r, casServerUrl)
return false, nil
}
localUrl := getLocalUrl(r)
ok, err, res := validateTicket(localUrl, casServerUrl)
global.SYSLOG.Debug("cas validateTicket", zap.Bool("ok", ok),
zap.Error(err), zap.Any("res", res))
if !ok {
redirectToCasServer(w, r, casServerUrl)
return false, nil
}
global.SYSLOG.Info("user authenticated", zap.String("sfrzh",
res.Data.SFRZH))
return true, res
}
func redirectToCasServer(w http.ResponseWriter, r *http.Request,
casServerUrl string) {
casServerUrl = casServerUrl + "/login?service=" + getLocalUrl(r)
http.Redirect(w, r, casServerUrl, http.StatusFound)
}

/*
验证访问路径中的ticket是否有效
*/
func validateTicket(localUrl, casServerUrl string) (bool, error,
*cas.CasServiceResponse) {
```

```
casServerUrl = casServerUrl + "/serviceValidate?service=" + localUrl
res, err := http.Get(casServerUrl)
if err != nil {
return false, err, nil
}
defer res.Body.Close()
data, err := io.ReadAll(res.Body)
if err != nil {
return false, err, nil
}
casRes, err := ParseCasUserInfo(data)
if err != nil {
return false, err, nil
}
if casRes.Data.SFRZH == "" {
return false, errors.New("authentication failed"), nil
}
//dataStr := string(data)
//if !strings.Contains(dataStr, "cas:authenticationSuccess") {
//return false, err, nil
//}
return true, nil, casRes
}

/*
从请求中获取访问路径
*/
func getLocalUrl(r *http.Request) string {
scheme := "http://"
if r.TLS != nil {
scheme = "https://"
}
url := strings.Join([]string{scheme, r.Host, r.RequestURI}, "")
fmt.Printf("url: %v\n", url)
slice := strings.Split(url, "?")
if len(slice) > 1 {
localUrl := slice[0]
urlParamStr := ensureOneTicketParam(slice[1])
url = localUrl + "?" + urlParamStr
}
return url
}

/*
处理并确保路径中只有一个ticket参数
*/
func ensureOneTicketParam(urlParams string) string {
if len(urlParams) == 0 || !strings.Contains(urlParams, "ticket") {
```

```
return urlParams
}
sep := ”&”
params := strings.Split(urlParams, sep)
newParams := ””
ticket := ””
for _, value := range params {
if strings.Contains(value, ”ticket”) {
ticket = value
continue
}
if len(newParams) == 0 {
newParams = value
} else {
newParams = newParams + sep + value
}
}
newParams = newParams + sep + ticket
return newParams
}

/*
获取ticket
*/
func getTicket(r *http.Request) string {
return r.FormValue(”ticket”)
}

/*
判断是否有ticket
*/
func hasTicket(r *http.Request) bool {
t := getTicket(r)
return len(t) != 0
}

func ParseCasUserInfo(data []byte) (*cas.CasServiceResponse, error) {
var casResponse cas.CasServiceResponse
if err := xml.Unmarshal(data, &casResponse); err != nil {
return nil, err
}
return &casResponse, nil
}

func GetUser(c *gin.Context) (*user.User, error) {
if res, exists := c.Get(”casResponse”); !exists {
return nil, errors.New(”cas authentication failed”)
} else {
```

```
casRes := res.(*cas.CasServiceResponse)
waitUser := &user.User{
UserName: casRes.Data.Attributes.UserName,
SFRZH:    casRes.Data.SFRZH,
}
return waitUser, nil
}
}
```

在 Gin 中间件中应用

将 CAS 认证逻辑应用到 Gin 的中间件中：

```
func CASMiddleware() gin.HandlerFunc {
return func(c *gin.Context) {
isAuth, casResponse := utils.IsAuthentication(c.Writer, c.Request,
utils.CASServer)
if !isAuth {
c.Abort()
return
}
c.Set("casResponse", casResponse)
c.Next()
return
}
}
```

总结
--

这里只根据cas原理实现了一个基本的CAS客户端认证流程，包括了请求检查、重定向处理、票据验证和用户信息解析，并通过Gin中间件集成到了Web应用程序中。