

目 录

基础知识

概述

开始使用

工作原理

教程

Model

支持Model

Model语法

效果器

函数

基于角色的访问控制

域内RBAC

Casbin RBAC v.s. RBAC96

ABAC

优先级模型

超级管理员

存储

Model存储

Policy存储

政策子集加载

扩充功能

适配器

观察者

调度器

角色管理器

中间件

GraphQL Middlewares

Cloud Native

API

API Overview

管理 API

RBAC API

RBAC with Domains API

RoleManager API

Data Permissions

高级用法

多线程

基准测试

性能优化

Authorization of Kubernetes

管理

管理员门户

Casbin 服务

日志 & 错误处理

前端使用

编辑器

在线编辑器

IDE 插件





基础知识





概述

概述

Casbin是一个强大的、高效的开源访问控制框架，其权限管理机制支持多种访问控制模型。

Casbin支持以下编程语言：

			
Casbin	jCasbin	node-Casbin	PHP-Casbin
production-ready	production-ready	production-ready	production-ready

			
PyCasbin	Casbin.NET	Casbin-CPP	Casbin-RS
production-ready	production-ready	beta-test	production-ready

Casbin 是什么？

Casbin 可以：

1. 支持自定义请求的格式，默认的请求格式为{subject, object, action}。
2. 具有访问控制模型model和策略policy两个核心概念。
3. 支持RBAC中的多层角色继承，不止主体可以有角色，资源也可以具有角色。
4. 支持内置的超级用户 例如：`root`或`administrator`。超级用户可以执行任何操作而无需显式的权限声明。
5. 支持多种内置的操作符，如 `keyMatch`，方便对路径式的资源进行管理，如 `/foo/bar` 可以映射到 `/foo*`
Casbin 不能：
 6. 身份认证 `authentication`（即验证用户的用户名、密码），`casbin`只负责访问控制。应该有其他专门的组件负责身份认证，然后由`casbin`进行访问控制，二者是相互配合的关系。
 7. 管理用户列表或角色列表。`Casbin` 认为由项目自身来管理用户、角色列表更为合适，用户通常有他们的密码，但是 `Casbin` 的设计思想并不是把它作为一个存储密码的容器。而是存储RBAC方案中用户和角色之间的映射关系。

开始使用

开始使用

安装

Go

```
go get github.com/casbin/casbin/v2
```

Java

```
<dependency>  
  <groupId>org.casbin</groupId>  
  <artifactId>jcasbin</artifactId>  
  <version>1.6.0</version>  
</dependency>
```

Node.js

```
# NPM  
npm install casbin --save  
  
# Yarn  
yarn add casbin
```

PHP

composer.json 中需要这个软件包。这将下载软件包:

```
composer require casbin/casbin
```

Python

```
pip install casbin
```

.NET

```
dotnet add package Casbin.NET
```

Rust

```
cargo install cargo-edit  
cargo add casbin  
  
// 如果你使用异步标准作为异步执行者  
cargo add async-std  
  
// 如果你使用tokio作为异步执行者  
cargo add tokio // 请确保您激活了其`macros`特性
```

新建一个Casbin enforcer

Casbin使用配置文件来设置访问控制模型

它有两个配置文件, `model.conf` 和 `policy.csv`。其中, `model.conf` 存储了我们的访问模型, 而 `policy.csv` 存储的是我们具体的用户权限配置。Casbin的使用非常精炼。基本上, 我们只需要一种主要的结构: `enforcer` 当构造这个结构的时候, `model.conf` 和 `policy.csv` 将会被加载。

用另一种说法就是, 当新建Casbin enforcer的时候 你必须提供一个 [Model](#) 和一个 [Adapter](#)。

Casbin拥有一个 [FileAdapter](#), 想知道更多请查阅“更多Adapter”中的[Adapter](#)

- 使用Model文件和默认 [FileAdapter](#):

Go

```
import "github.com/casbin/casbin/v2"

e, err := casbin.NewEnforcer("path/to/model.conf", "path/to/policy.csv")
```

Java

```
import org.casbin.jcasbin.main.Enforcer;

Enforcer e = new Enforcer("path/to/model.conf", "path/to/policy.csv");
```

Node.js

```
import { newEnforcer } from 'casbin';

const e = await newEnforcer('path/to/model.conf', 'path/to/policy.csv');
```

PHP

```
require_once './vendor/autoload.php';

use Casbin\Enforcer;

$e = new Enforcer("path/to/model.conf", "path/to/policy.csv");
```

Python

```
import casbin

e = casbin.Enforcer("path/to/model.conf", "path/to/policy.csv")
```

.NET

```
using NetCasbin;

var e = new Enforcer("path/to/model.conf", "path/to/policy.csv");
```

Delphi

```

var
  casbin: ICasbin;
begin
  casbin := TCasbin.Create('path/to/model.conf', 'path/to/policy.csv');
  ...
end

```

Rust

```

use casbin::prelude::*;

// 如果你使用async_std作为异步执行器
#[cfg(feature = "runtime-async-std")]
#[async_std::main]
async fn main() -> Result<> {
  let mut e = Enforcer::new("path/to/model.conf", "path/to/policy.csv").await?;
  Ok(())
}

// 如果你使用tokio作为异步执行器
#[cfg(feature = "runtime-tokio")]
#[tokio::main]
async fn main() -> Result<> {
  let mut e = Enforcer::new("path/to/model.conf", "path/to/policy.csv").await?;
  Ok(())
}

```

- 与其他Adapter一起使用Model text

Go

```

import (
  "log"

  "github.com/casbin/casbin/v2"
  "github.com/casbin/casbin/v2/model"
  xormadapter "github.com/casbin/xorm-adapter/v2"
  _ "github.com/go-sql-driver/mysql"
)

// 使用MySQL数据库初始化一个Xorm适配器
a, err := xormadapter.NewAdapter("mysql", "mysql_username:mysql_password@tcp(127.0.0.1:3306)/casbin")
if err != nil {
  log.Fatalf("error: adapter: %s", err)
}

m, err := model.NewModelFromString(
  [request_definition]
  r = sub, obj, act

  [policy_definition]
  p = sub, obj, act

  [policy_effect]
  e = some(where (p.eft == allow))

  [matchers]
  m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
)

```

```
if err != nil {
    log.Fatalf("error: model: %s", err)
}

e, err := casbin.NewEnforcer(m, a)
if err != nil {
    log.Fatalf("error: enforcer: %s", err)
}
```

检查权限

在访问发生之前，在代码中添加 **enforcement hook**

Go

```
sub := "alice" // 想要访问资源的用户
obj := "data1" // 将要被访问的资源
act := "read" // 用户对资源的操作

ok, err := e.Enforce(sub, obj, act)

if err != nil {
    // 处理err
}

if ok == true {
    // 允许alice读取data1
} else {
    // 拒绝请求，抛出异常
}

// 您可以使用BatchEnforce()来批量执行一些请求
// 这个方法返回布尔切片，此切片的索引对应于二维数组的行索引。
// 例如results[0]是{"alice", "data1", "read"}的结果
results, err := e.BatchEnforce([][]interface{}{"alice", "data1", "read"}, {"bob", "data2", "write"}, {"jack", "data3", "read"})
```

Java

```
String sub = "alice"; // 想要访问资源的用户
String obj = "data1"; // 将要被访问的资源
String act = "read"; // 用户对资源进行的操作

if (e.enforce(sub, obj, act) == true) {
    // 允许alice读取data1
} else {
    // 拒绝请求，抛出异常
}
```

Node.js

```
const sub = 'alice'; // 想要访问资源的用户
const obj = 'data1'; // 将要被访问的资源
const act = 'read'; // 用户对资源进行的操作

if ((await e.enforce(sub, obj, act)) === true) {
    // 允许alice读取data1
}
```



```
} else {  
    // 拒绝请求，抛出异常  
}
```

PHP

```
$sub = "alice"; // 想要访问资源的用户  
$obj = "data1"; // 将要被访问的资源  
$act = "read"; // 用户对资源进行的操作  
  
if ($e->enforce($sub, $obj, $act) === true) {  
    // 允许alice读取data1  
} else {  
    // 拒绝请求，抛出异常  
}
```

Python

```
sub = "alice" # 想要访问资源的用户  
obj = "data1" # 将要被访问的资源  
act = "read" # 用户对资源进行的操作  
  
if e.enforce(sub, obj, act):  
    # 允许alice读取data1  
    pass  
else:  
    # 拒绝请求，抛出异常  
    pass
```

.NET

```
var sub = "alice"; # 想要访问资源的用户  
var obj = "data1"; # 将要被访问的资源  
var act = "read"; # 用户对资源进行的操作  
  
if (await e.EnforceAsync(sub, obj, act))  
{  
    // 允许alice读取data1  
}  
else  
{  
    // 拒绝请求，抛出异常  
}
```

Delphi

```
if casbin.enforce(['alice', data1, read']) then  
    // Alice很开心它能够读取data1了  
else  
    // Alice很伤心
```

Rust

```
let sub = "alice"; // 想要访问资源的用户  
let obj = "data1"; // 将会被访问的资源  
let act = "read"; // 用户对资源的操作
```

```
if e.enforce((sub, obj, act)).await? {  
    // 允许alice读取data1  
} else {  
    // 发生错误  
}
```

Casbin 还提供了在运行时管理用户权限的API。例如，您可以通过以下方式查询分配给某个用户的所有角色：

Go

```
roles := e.GetRolesForUser("alice")
```

Java

```
Roles roles = e.getRolesForUser("alice");
```

Node.js

```
const roles = await e.getRolesForUser('alice');
```

PHP

```
$roles = $e->getRolesForUser("alice");
```

Python

```
roles = e.get_roles_for_user("alice")
```

.NET

```
var roles = e.GetRolesForUser("alice");
```

Delphi

```
roles = e.rolesForEntity("alice")
```

Rust

```
let roles = e.get_roles_for_user("alice");
```

请参阅 [Management API and RBAC API](#) 以获取更多使用方式。

请查看测试用例以获取更多使用方式。

工作原理

工作原理

在 Casbin 中, 访问控制模型被抽象为基于 PERM (Policy, Effect, Request, Matcher) 的一个文件。因此, 切换或升级项目的授权机制与修改配置一样简单。您可以通过组合可用的模型来定制您自己的访问控制模型。例如, 您可以在一个 model 中结合 RBAC 角色和 ABAC 属性, 并共享一组 policy 规则。

PERM 模式由四个基础 (政策、效果、请求、匹配) 组成, 描述了资源与用户之间的关系。

请求

定义请求参数。基本请求是一个元组对象, 至少需要主题(访问实体)、对象(访问资源) 和动作(访问方式)

例如, 一个请求可能长这样: `r={sub, obj, act}`

它实际上定义了我们应该提供访问控制匹配功能的参数名称和顺序。

策略

定义访问策略模式。事实上, 它是在政策规则文件中定义字段的名称和顺序。

例如: `p={sub, obj, act}` 或 `p={sub, obj, act, eft}`

注: 如果未定义 eft (policy result), 则策略文件中的结果字段将不会被读取, 和匹配的策略结果将默认被允许。

匹配器

匹配请求和政策的规则。

例如: `m = r.sub == p.sub && r.act == p.act && r.obj == p.obj` 这个简单和常见的匹配规则意味着如果请求的参数 (访问实体, 访问资源和访问方式) 匹配, 如果可以在策略中找到资源和方法, 那么策略结果 (p.eft) 便会返回。策略的结果将保存在 p.eft 中。

效果

它可以被理解成一种模型, 在这种模型中, 对匹配结果再次作出逻辑组合判断。

例如: `e = some (where (p.eft == allow))`

这句话意味着, 如果匹配的策略结果有一些是允许的, 那么最终结果为真。

让我们看看另一个示例: `e = some (where (p.eft == allow)) && !some (where (p.eft == deny))` 此示例组合的逻辑含义是: 如果有符合允许结果的策略且没有符合拒绝结果的策略, 结果是为真。换言之, 当匹配策略均为允许 (没有任何否认) 是为真 (更简单的是, 既允许又同时否认, 拒绝就具有优先地位)。

Casbin 中最基本、最简单的 model 是 ACL。ACL 中的 model 配置为:

```
# Request definition
[request_definition]
r = sub, obj, act

# Policy definition
[policy_definition]
p = sub, obj, act
```

```
# Policy effect
[policy_effect]
e = some(where (p.eft == allow))

# Matchers
[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```

ACL模型的一个示例策略类似:

```
p, alice, data1, read
p, bob, data2, write
```

它意味着:

- **alice**可以读取**data1**
- **bob**可以编写**data2**

我们还支持多行模式, 通过在结尾处追加“\”:

```
# 匹配器
[matchers]
m = r.sub == p.sub && r.obj == p.obj \
  && r.act == p.act
```

此外, 对于 ABAC, 您可以在 **Casbin golang** 版本中尝试下面的in (**jCasbin** 和 **Node-Casbin** 尚不支持) 操作:

```
# Matchers
[matchers]
m = r.obj == p.obj && r.act == p.act || r.obj in ('data2', 'data3')
```

但是你应该确保数组的长度大于 1, 否则的话将会导致 **panic**。

对于更多操作, 你可以查看[govaluate](#)。

教程

教程

在阅读本教程之前，请注意有的Casbin模型教程适合不同语言执行的Casbin，然而有的其他的教程是对于特定的语言我们的论文

- [PML: 一种基于Interpreter的Web服务访问控制策略语言](#)

这篇论文深入介绍了Casbin的设计细节。如果您使用Casbin/PML，请引用以下BibTex作为参考文献：

```
@article{luo2019pml,  
  title={PML: An Interpreter-Based Access Control Policy Language for Web Services},  
  author={Luo, Yang and Shen, Qingni and Wu, Zhonghai},  
  journal={arXiv preprint arXiv:1903.09756},  
  year={2019}  
}
```

- [一种基于元模型的访问控制策略描述语言](#)

这里是另外一种更长版本的论文，发表在《软件学报》上 不同格式的引文 (Refworks, EndNote 等) 可在以下网址找到： [\(另一个版本\) 基于元模型的访问控制策略规格语言\(中文\)](#)

视频

- [一个安全保险库 - 实现与 Casbin 的中间件的授权 - JuniorDevSG](#)
- [基于Casbin的微型服务架构分享用户权限\(俄文\)](#)
- [Nest.js - Casbin RESTful RBAC授权中间件](#)
- [Gin 教程 第10章: 30分钟内学习 Casbin 基础模型](#)
- [Gin 教程第11章: 编码, API 和Casbin中的自定义功能](#)
- [Gin+Casbin权限实战速学\(中文\)](#)
- [jCasbin 基础: 一个简单的RBAC示例\(中文\)](#)
- [基于Casbin的Golang RBAC模型\(中文\)](#)
- [学习Gin + Casbin\(1\): 通路& 概述\(中文\)](#)
- [ThinkPHP 5.1 + Casbin: 导言\(中文\)](#)
- [ThinkPHP 5.1 + Casbin: RBAC授权 \(中文\)](#)
- [ThinkPHP 5.1 + Casbin: RESTfull & 中间件\(中文\)](#)

- [PHP-Casbin 快速上手\(中文\)](#)

PERM元模型(策略、效果、请求、匹配器)

- [在Casbin中使用PERM建模授权](#)
- [使用 Casbin 设计一个灵活的权限系统](#)
- [Authorize with Access Control Lists](#)
- [Access control with PERM and Casbin \(in Persian\)](#)
- [RBAC? ABAC? .. PERM! New Way of Authorization for Cloud-Based Web Services and Apps \(in Russian\)](#)
- [Practice & Examples of Flexible Authorization Using Casbin & PERM \(in Russian\)](#)
- [Permission management with Casbin \(in Chinese\)](#)
- [Analysis of Casbin \(in Chinese\)](#)
- [Design of System Permissions \(in Chinese\)](#)
- [Casbin: A Permission Engine \(in Chinese\)](#)
- [Implementing ABAC with Casbin \(in Chinese\)](#)
- [Source code analysis of Casbin \(in Chinese\)](#)
- [Permission evaluation with Casbin \(in Chinese\)](#)
- [Casbin: Library of the day for Go \(in Chinese\)](#)

Go

HTTP & RESTful

在Go中使用 [Casbin](#) 实现基础的基于角色的 [HTTP](#) 授权 (或 [中文翻译](#))

Watcher

[RBAC Distributed Synchronization via Casbin Watcher \(in Chinese\)](#)

Beego

- [Using Casbin with Beego: 1. Get started and test \(in Chinese\)](#)
- [Using Casbin with Beego: 2. Policy storage \(in Chinese\)](#)
- [Using Casbin with Beego: 3. Policy query \(in Chinese\)](#)

- [Using Casbin with Beego: 4. Policy update \(in Chinese\)](#)
- [Using Casbin with Beego: 5. Policy update \(continued\) \(in Chinese\)](#)

Gin

[Tutorial: Integrate Gin with Cabsin](#)

[Policy enforcements on K8s with Pipeline](#)

[Authentication and authorization in Gin application with JWT and Casbin](#)

[Backend API with Go: 1. Authentication based on JWT \(in Chinese\)](#)

[Backend API with Go: 2. Authorization based on Casbin \(in Chinese\)](#)

[Using Go's authorization library Casbin with Gin and GORM \(in Japanese\)](#)

Echo

[Web authorization with Casbin](#)

Iris

[Iris + Casbin: Practice for permission management \(in Chinese\)](#)

[Role-based access control for HTTP based on Casbin \(in Chinese\)](#)

[Learning iris + Casbin from scratch](#)

VMware Harbor

[Casbin: Golang access control framework \(in Chinese\)](#)

[Access control in Harbor \(in Chinese\)](#)

Argo CD

[Organizational RBAC in Argo CD with Casbin](#)

GShark

[GShark: Scan for sensitive information in Github easily and effectively \(in Chinese\)](#)

Java

SprintBoot

- [jCasbin: a more light-weight permission management solution \(in Chinese\)](#)

[Integrating jCasbin with JFinal \(in Chinese\)](#)

Node.js

Express

[How to Add Role-Based-Access-Control to Your Serverless HTTP API on AWS](#)

Koa

[Authorisation with Casbin and Koa Part 1](#)

[Authorisation with Casbin and Koa Part 2](#)

Nest

- [How to Create Role based Authorization Middleware with Casbin and Nest.js](#)
- [nest.js: Casbin RESTful RBAC authorization middleware \(Video\)](#)
- [A Demo App of Attribute-based Access Control in Node.js Based on Casbin](#)
- [Multi tenant SaaS starter kit with cqrs graphql microservice architecture](#)

Fastify

[Access Control in Node.js with Fastify and Casbin](#)

PHP

[Casbin, Powerful and Efficient ACL for Your Projects](#)

Laravel

[Laravel authorization: authorization library supporting ACL, RBAC, ABAC and other models](#)

.NET

[Using Casbin for authorization in dotnet](#)

Rust

- [Basic Role-Based HTTP Authorization in Rust with Casbin](#)
- [How to use casbin authorization in your rust web-app \[Part - 1\]](#)
- [How to use casbin authorization in your rust web-app \[Part - 2\]](#)

Model

Model

支持Model

支持的Models

1. **ACL (Access Control List, 访问控制列表)**
2. 具有 **超级用户** 的 ACL
3. 没有用户的 ACL: 对于没有身份验证或用户登录的系统尤其有用。
4. 没有资源的 ACL: 某些场景可能只针对资源的类型, 而不是单个资源, 诸如 **write-article**, **read-log**等权限。它不控制对特定文章或日志的访问。
5. **RBAC (基于角色的访问控制)**
6. 支持资源角色的RBAC: 用户和资源可以同时具有角色 (或组)。
7. 支持域/租户的RBAC: 用户可以为不同的域/租户设置不同的角色集。
8. **ABAC (基于属性的访问控制):** 支持利用resource.Owner这种语法糖获取元素的属性。
9. **RESTful:** 支持路径, 如 **/res/***, **/res/: id** 和 HTTP 方法, 如 GET, POST, PUT, DELETE。
10. 拒绝优先: 支持允许和拒绝授权, 拒绝优先于允许。
11. 优先级: 策略规则按照先后次序确定优先级, 类似于防火墙规则。

例子

访问控制模型	Model 文件	Policy 文件
ACL	basic_model.conf	basic_policy.csv
具有超级用户的ACL	basic_with_root_model.conf	basic_policy.csv
没有用户的ACL	basic_without_users_model.conf	basic_without_us
没有资源的ACL	basic_without_resources_model.conf	basic_without_res
RBAC	rbac_model.conf	rbac_policy.csv
支持资源角色的RBAC	rbac_with_resource_roles_model.conf	rbac_with_resour
支持域/租户的RBAC	rbac_with_domains_model.conf	rbac_with_domain
ABAC	abac_model.conf	无
RESTful	keymatch_model.conf	keymatch_policy.
拒绝优先	rbac_with_not_deny_model.conf	rbac_with_deny_r

同高与决绝 切高控制模型	rbac_with_deny_model.conf Model 文件	rbac_with_deny_r Policy 文件
优先级	priority_model.conf	priority_policy.csv
明确优先级	priority_model_explicit	priority_policy_ex

Model语法

Model语法

- Model CONF 至少应包含四个部分: [request_definition], [policy_definition], [policy_effect], [matchers]。
- 如果 model 使用 RBAC, 还需要添加[role_definition]部分。
- Model CONF 文件可以包含注释。注释以 # 开头, # 会注释该行剩余部分。

Request定义

[request_definition] 部分用于request的定义, 它明确了 e.Enforce(...) 函数中参数的含义。

[request_definition]

```
r = sub, obj, act
```

sub, obj, act 表示经典三元组: 访问实体 (Subject), 访问资源 (Object) 和访问方法 (Action)。但是, 你可以自定义你自己的请求表单, 如果不需要指定特定资源, 则可以这样定义 sub、act, 或者如果有两个访问实体, 则为 sub、sub2、obj、act。

Policy定义

[policy_definition] 部分是对policy的定义, 以下文的 model 配置为例:

[policy_definition]

```
p = sub, obj, act
p2 = sub, act
```

这些是我们对policy规则的具体描述

```
p, alice, data1, read
p2, bob, write-all-objects
```

policy部分的每一行称之为一个策略规则, 每条策略规则通常以形如p, p2的policy type开头。如果存在多个policy定义, 那么我们会根据前文提到的policy type与具体的某条定义匹配。上面的policy的绑定关系将会在matcher中使用, 罗列如下:

```
(alice, data1, read) -> (p.sub, p.obj, p.act)
(bob, write-all-objects) -> (p2.sub, p2.act)
```

NOTE 当前只支持形如 p的单个policy定义. p2 类型的尚未支持。通常情况下, 用户无需使用多个 policy 定义, 如果您有其他情形的policy定义诉求, 请在 <https://github.com/casbin/casbin/issues/new> 提出issue告知我们。

TIP Policy rule中的元素总会被视作string。如果您对此有任何疑问, 请查看讨论: <https://github.com/casbin/casbin/issues/113>

Policy effect定义

[policy_effect] 部分是对policy生效范围的定义, 原语定义了当多个policy rule同时匹配访问请求request时,该如何对多个决策结果进行集成以实现统一决策。以下示例展示了一个只有一条规则生效, 其余都被拒绝的情况:

```
[policy_effect]
e = some(where (p.eft == allow))
```

该Effect原语表示如果存在任意一个决策结果为allow的匹配规则，则最终决策结果为allow，即allow-override。其中p.eft表示策略规则的决策结果，可以为allow或者deny，当不指定规则的决策结果时，取默认值allow。通常情况下，policy的p.eft默认为allow，因此前面例子中都使用了这个默认值。

这是另一个policy effect的例子：

```
[policy_effect]
e = !some(where (p.eft == deny))
```

该Effect原语表示不存在任何决策结果为deny的匹配规则，则最终决策结果为allow，即deny-override。some量词判断是否存在一条策略规则满足匹配器。any量词则判断是否所有的策略规则都满足匹配器（此处未使用）。policy effect还可以利用逻辑运算符进行连接：

```
[policy_effect]
e = some(where (p.eft == allow)) && !some(where (p.eft == deny))
```

该Effect原语表示当至少存在一个决策结果为allow的匹配规则，且不存在决策结果为deny的匹配规则时，则最终决策结果为allow。这时allow授权和deny授权同时存在，但是deny优先。

NOTE 尽管我们设计了政策效果的语法。目前的执行只是使用硬编码的政策效果，因为我们认为这种灵活性没有多大必要。目前为止你必须使用内置的 policy effects，不能自定义。

支持的 policy effects 如下：

Policy effect	意义	示例
some(where (p.eft == allow))	allow-override	ACL, RBAC, etc.
!some(where (p.eft == deny))	deny-override	Deny-override
some(where (p.eft == allow)) && !some(where (p.eft == deny))	allow-and-deny	Allow-and-deny
priority(p.eft)==deny	priority	Priority

匹配器

matchers定义了策略匹配器。匹配器是一组表达式。它定义了如何根据请求来匹配策略规则

```
[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```

上面的这个匹配器是最简单的，它表示请求的三元组：主题、对象、行为都应该匹配策略规则中的表达式。

在匹配器中，你可以使用算术运算符如 +, -, *, /, 也可以使用逻辑运算符如: &&, ||, !。

NOTE Although it seems like there will be multiple matchers such as m1, m2 like other primitives, currently, we only support one matcher m. 通常情况下,您可以在一个matcher中使用上文提到的逻辑运算符来实现复杂的逻辑判断,因此,我们认为现在没有必要支持多个匹配器。如果您对此有疑问,请告知我们(<https://github.com/casbin/casbin/issues>)。

Special Grammar

You could also use `in`, the only operator with a text name. This operator checks the right-hand side array to see if it contains a value that is equal to the left-side value. Equality is determined by the use of the `==` operator, and this library doesn't check types between the values. Any two values, when cast to `interface{}`, and can still be checked for equality with `==` will act as expected. Note that you can use a parameter for the array, but it must be an `[]interface{}`.

Also refer to [rbac_model_matcher_using_in_op](#), [keyget2_model](#) and [keyget_model](#)

Example:

```
[request_definition]
r = sub, obj
...
[matchers]
m = r.sub.Name in (r.obj.Admins)
```

```
e.Enforce(Sub{Name: "alice"}, Obj{Name: "a book", Admins: []interface{}{"alice", "bob"}})
```

Expression evaluator

The matcher evaluation in Casbin is implemented by expression evaluators in each language. Casbin integrates their powers to provide the unified PERM language. Besides all the model syntax provided here, those expression evaluators may provide extra functionality, which may be not supported by another language or implementation. Use it at your own risk.

The expression evaluators used by each Casbin implementation are:

实现	语言	表达式运算器
Casbin	Golang	https://github.com/Knetic/govaluate
jCasbin	Java	https://github.com/killme2008/aviator
Node-Casbin	Node.js	https://github.com/donmccurdy/expression-eval
PHP-Casbin	PHP	https://github.com/symfony/expression-language
PyCasbin	Python	https://github.com/danthedekie/simpleeval
Casbin.NET	C#	https://github.com/davideicardi/DynamicExpresso
Casbin4D	Delphi	https://github.com/casbin4d/Casbin4D/tree/master/Sou
casbin-rs	Rust	https://github.com/jonathandturner/rhai

NOTE

If you encounter performance issue about Casbin, it's probably caused by the low efficiency of the expression evaluator. You can both send issue to Casbin or the expression evaluator directly for advice to speed up. See [Benchmarks](#) section for details.

效果器

效果器

Effect是一个policy rule的结果 Effector 是用于Casbin effector的API

MergeEffects()

MergeEffects将 enforcer 收集的所有匹配结果合并为一项决定。

例如:

Go

```
Effect, explainIndex, err = e.MergeEffects(expr, effects, matches, policyIndex, policyLength)
```

在本示例中:

- Effect 是此函数合并的最后决定(初始参数为 indeterminate)。
- explainIndex 是 eft 的索引, eft 的值可为 Allow 或者 Deny.(初始值是 -1)
- err 用于检查effect是否受到支持。
- expr 是被存储为 string 的policy_effects
- effects 是 Effect 的数组, 其中值可以为Allow, Indeterminate 或者 Deny
- matchers 是显示结果是否符合策略的数组。
- policyIndex 是模型中的策略索引。
- policyLength 是策略的长度。
- 上面的代码说明了我们如何将参数传递到 MergeEffects 函数, 并且该函数将根据 expr 处理效果和匹配。

要部署一个Effector, 我们可以这样做:

Go

```
var e Effector  
Effect, explainIndex, err = e.MergeEffects(expr, effects, matches, policyIndex, policyLength)
```

MergeEffects 表明如果 expr 可以匹配结果, 也就是说 p_oft 是 `allow, 我们就可以合并所有效果。如果没有符合拒绝的规则, 我们就允许这样做。

:::note

如果 expr 不能匹配 “priority(p_oft) || deny” 以及 policyIndex 短于 policyLength-1`, 它将短路中间的一些效果。 :::

函数

函数

Matchers中的函数

你甚至可以在Matcher中指定函数，使它更强大。你可以使用内置函数，或者指定你自己的函数。所有的内置函数都需要这样的格式(除了 keyGet 和 keyGet2):

```
bool function_name(string arg1, string arg2)
```

它返回arg1是否匹配arg2。

keyGet and keyGet2 将返回匹配通配符的字符串，如果没有匹配返回 ""。

支持的内置函数如下：

函数	参数1	参数2	示例
keyMatch	一个URL 路径，例如 /alice_data/resource1	一个URL 路径或 * 模式下， 例如 /alice_data/*	keymatch
keyGet	一个URL 路径，例如 /alice_data/resource1	一个URL 路径或 * 模式下， 例如 /alice_data/*	keyget_m
keyMatch2	一个URL 路径，例如 /alice_data/resource1	一个URL 路径或 : 模式下， 例如 /alice_data/:resource	keymatch
keyGet2	一个URL 路径，例如 /alice_data/resource1	一个URL 路径或 : 模式下， 例如 /alice_data/:resource	keyget_m
keyMatch3	一个URL 路径，例如 /alice_data/resource1	一个URL 路径或 {} 模式下， 例如 /alice_data/{resource}	https://gitL196

keyMatch4	一个URL 路径，例如 /alice_data/resource1	一个URL 路径或 {} 模式下， 例如 /alice_data/{id}/book/{id}	https://gitL222
regexMatch	任意字符串	正则表达式模式	keymatch
ipMatch	一个 IP 地址，例如 192.168.2.123	一个 IP 地址或一个 CIDR ， 例如192.168.2.0/24	ipmatch_r
	类似路径的	一个全局模式。例如	https://git

globMatch 函数	参数e_data/resource1	参数e_data/*	https://github.com/casbin/casbin/blob/master/
-----------------	--------------------	------------	---

详情见: https://github.com/casbin/casbin/blob/master/util/buildtin_operators_test.go。

怎样增加自定义函数

首先准备您的函数。它接受一些参数，然后返回一个布尔类型:

```
func KeyMatch(key1 string, key2 string) bool {
    i := strings.Index(key2, "*")
    if i == -1 {
        return key1 == key2
    }

    if len(key1) > i {
        return key1[:i] == key2[:i]
    }
    return key1 == key2[:i]
}
```

然后用 `interface{}` 类型的接口包装它:

```
func KeyMatchFunc(args ...interface{}) (interface{}, error) {
    name1 := args[0].(string)
    name2 := args[1].(string)

    return (bool)(KeyMatch(name1, name2)), nil
}
```

最后，在Casbin的执法者(enforcer)中注册这个函数:

```
e.AddFunction("my_func", KeyMatchFunc)
```

现在，您可以在您的模型CONF中像这样使用这个函数:

```
[matchers]
m = r.sub == p.sub && my_func(r.obj, p.obj) && r.act == p.act
```

基于角色的访问控制

基于角色的访问控制

角色定义

[role_definition] 是RBAC角色继承关系的定义。Casbin 支持 RBAC 系统的多个实例, 例如, 用户可以具有角色及其继承关系, 资源也可以具有角色及其继承关系。这两个 RBAC 系统不会互相干扰。

此部分是可选的。如果在模型中不使用 RBAC 角色, 则省略此部分。

[role_definition]

```
g = _, _  
g2 = _, _
```

上述角色定义表明, g 是一个 RBAC系统, g2 是另一个 RBAC 系统。_,_表示角色继承关系的前项和后项, 即前项继承后项角色的权限。一般来讲, 如果您需要进行角色和用户的绑定, 直接使用g 即可。当您需要表示角色(或者组)与用户和资源的绑定关系时, 可以使用g 和 g2 这样的表现形式。请参见 [rbacmodel]

(http://https://github.com/casbin/casbin/blob/master/examples/rbac_model.conf “rbac_model”) 和 [rbac_model_with_resource_roles](http://https://github.com/casbin/casbin/blob/master/examples/rbac_model_with_resource_roles.conf) 的示例。

在Casbin里, 我们以policy表示中实际的用户角色映射关系(或是资源-角色映射关系), 例如:

```
p, data2_admin, data2, read  
g, alice, data2_admin
```

这意味着 alice 是角色 data2_admin的一个成员。alice 在这里可以是用户、资源或角色。Cabin 只是将其识别为一个字符串。

接下来在matcher中, 应该像下面的例子一样检查角色信息:

[matchers]

```
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

这意味着在请求中应该在policy中包含sub角色。

NOTE

1. Casbin 只存储用户角色的映射关系。
2. Cabin 没有验证用户是否是有效的用户, 或者角色是一个有效的角色。这应该通过认证来解决。
3. RBAC 系统中的用户名称和角色名称不应相同。因为Casbin将用户名和角色识别为字符串, 所以当前语境下Casbin无法得出这个字面量到底指代用户 alice 还是角色 alice。这时, 使用明确的 role_alice, 问题便可迎刃而解。
4. 假设A具有角色 B, B 具有角色 C, 并且 A 有角色 C。这种传递性在当前版本会造成死循环。

角色层次

Casbin 的 RBAC 支持 RBAC1 的角色层次结构功能, 如果 alice具有role1, role1具有role2, 则 alice 也将拥有 role2 并继承其权限。

下面是一个称为层次结构级别的概念。因此, 此示例的层次结构级别为2。对于Casbin中的内置角色管理器, 可以指定最大层次结构级别。默认值为10。这意味着终端用户 alice 只能继承10个级别的角色。

```
// NewRoleManager is the constructor for creating an instance of the
// default RoleManager implementation.
func NewRoleManager(maxHierarchyLevel int) rbac.RoleManager {
    rm := RoleManager{}
    rm.allRoles = &sync.Map{}
    rm.maxHierarchyLevel = maxHierarchyLevel
    rm.hasPattern = false

    return &rm
}
```

如何区分用户和角色？

在RBAC中，Casbin不对用户和角色进行区分。它们都被视为字符串。如果你只使用单层的RBAC模型（角色不会成为另一个角色的成员）。可以使用 `e.GetAllSubjects()` 获取所有用户，`e.GetAllRoles()` 获取所有角色。它们会为规则 `g, u, r` 分别列出所有的 `u` 和 `r`。

但如果你在使用多层RBAC（带有角色继承），并且你的应用没有记录下一个名字（字符串）对应的是用户还是角色，或者你将用户和角色用相同的名字命名。那么你可以给角色加上像 `role::admin` 的前缀再传递到Casbin中。由此可以通过查看前缀来区分用户和角色。

如何查询隐性角色或权限？

当用户通过RBAC层次结构继承角色或权限，而不是直接在策略规则中分配它们时，我们将这种类型的分配称为 `implicit`。To query such implicit relations, you need to use these 2 APIs: `GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser()` instead of `GetRolesForUser()` and `GetPermissionsForUser()`. 有关详情，请参阅 [this GitHub issue](#)。

在 RBAC 中使用模式匹配

有时，您希望一些具有特定模式的subjects, object 或者 domains/tenants能够被自动授予角色。RBAC中的模式匹配函数可以帮助做到这一点。模式匹配函数与前一个函数共享相同的参数和返回值：`matcher function`。

模式匹配函数支持g的每一个参数

我们知道，在matcher里面RBAC通常被表示为 `g(r.sub, p.sub)` 接下来我们将使用如下策略：

```
p, alice, book_group, read
g, /book/1, book_group
g, /book/2, book_group
```

因此alice 可以阅读所有书籍，包括book 1和book 2。但是当有数千本书时，如果我们仅仅使用g策略规则将每本书一个一个地添加到书籍角色（或组），那将会是非常繁琐的。

不过，凭借着模式匹配函数，你可以把整个策略只用一行写下！

```
g, /book/:id, book_group
```

Casbin会自动将/book/1和/book/2匹配为模式/book/:id。您需要做的仅仅是向enforcer注册该方法，例如像这样：

```
r := e.GetRoleManager()
r.(*defaultRoleManager.RoleManager).AddMatchingFunc("KeyMatch2", util.KeyMatch2)
```

当在domains/tenants里面使用模式匹配函数的时候，你需要把这个函数向enfoecer以及model进行注册

将 `keyMatch2` 向enforcer进行注册：

```
r := e.GetRoleManager()  
r.(*defaultrolemanager.RoleManager).AddDomainMatchingFunc("KeyMatch2", util.KeyMatch2)
```

将 `keyMatch2` 向 `model` 进行注册:

```
m = g(r.sub, p.sub, r.dom) && keyMatch2(r.dom, p.dom) && r.obj == p.obj && r.act == p.act
```

If you don't understand what `g(r.sub, p.sub, r.dom)` means, please read [this](#). In short, `g(r.sub, p.sub, r.dom)` will check whether the user `r.sub` has a role `p.sub` in the domain `r.dom`. So this is how the matcher work. You can see the full example [here](#).

除了上面的模式匹配语法外，我们还可以使用纯域模式。

例如，如果我们想要，`sub` 在这两个不同的域拥有访问权限，`domain1` 以及 `domain2`，我们可以使用纯域模式匹配:

```
p, admin, domain1, data1, read  
p, admin, domain1, data1, write  
p, admin, domain2, data2, read  
p, admin, domain2, data2, write  
  
g, alice, admin, *  
g, bob, admin, domain2
```

在这个示例中，我们想要 `alice` 可以读写 `data` 在域1和域2中，模式匹配 `*` 在 `g` 使得 `alice` 拥有两个域的权限

通过使用模式匹配，特别是在一些比较复杂的环境中有很多的域以及我们需要考虑很多物体的时候，我们可以更简洁高效地实现 `policy_definition`。

角色管理器

详见 [Role Managers](#)

域内RBAC

域内RBAC

域租户的角色定义

在Casbin中的RBAC角色可以是全局或是基于特定于域的。特定域的角色意味着当用户处于不同的域/租户群体时，用户所表现的角色也不尽相同。这对于像云服务这样的大型系统非常有用，因为用户通常分属于不同的租户群体。

域/租户的角色定义应该类似于：

```
[role_definition]
g = _, _, _
```

第三个 _ 表示域/租户的名称，此部分不应更改。然后，政策可以是：

```
p, admin, tenant1, data1, read
p, admin, tenant2, data2, read

g, alice, admin, tenant1
g, alice, user, tenant2
```

该实例表示tenant1的域内角色admin 可以读取data1，alice在tenant1域中具有admin角色，但在tenant2域中具有user角色，所以alice可以有读取data1的权限。同理，因为alice不是tenant2的admin，所以她访问不了data2。

接下来在matcher中，应该像下面的例子一样检查角色信息：

```
[matchers]
m = g(r.sub, p.sub, r.dom) && r.dom == p.dom && r.obj == p.obj && r.act == p.act
```

更多示例参见: [rbac_with_domains_model.conf](#)。

Casbin RBAC v.s. RBAC96

Casbin RBAC v.s. RBAC96

Casbin RBAC和RBAC96

1. 在这个文档中，我们会比较Casbin RBAC与 [RBAC96](#)的区别。

Casbin RBAC支持RBAC96的几乎所有特点，并在此基础上增加了新的特点。

RBAC版本	支持级别	说明
RBAC0	完全支持	RBAC0是RBAC96的基本版本。它澄清了使用者、角色和权限之间的关系。
RBAC1	完全支持	Casbin 的 RBAC 支持 RBAC1 的角色层次结构功能，如果 alice具有role1, role1具有role2, 则 alice 也将拥有 role2 并继承其权限。
RBAC2	支持相互专用处理 (示例), 但量化限制不是	RBAC2在RBAC0的基础上添加了约束 因此，RBAC2可以处理政策中相互排斥的问题。
RBAC3	支持相互专用处理 (示例), 但量化限制不是	RBAC3是RBAC1和RBAC2的组合。 RBAC3支持RBAC1和RBAC2中的角色等级和制约因素

Casbin RBAC和RBAC96之间的差异

1. 在Casbin，用户和角色之间的区分不明确。

在Casbin中，用户和角色都被视为字符串。如果您写了类似于以下的策略文件：

```
p, admin, book, read
p, alice, book, read
g, amber, admin
```

并调用方法 GetAllSubjects() 就像这样(e 是一个Casbin Enforcer的实例)：

```
e.GetAllSubjects()
```

然后您将得到下面的返回值：

```
[admin alice bob]
```

因为在Casbin，主体包括用户和角色。

然而，如果你调用方法 `GetAllRoles()` 就像这样：

```
e.GetAllRoles()
```

然后您将得到下面的返回值：

```
[admin]
```

现在你知道卡斯宾的用户和角色之间有区别，但并不像RBAC96那样尖锐。当然，您可以为您的政策添加一些前缀，如 `user::alice, role::admin` 以澄清他们之间的关系。

2. Casbin RBAC比RBAC96提供了更多的权限

RBAC96中只定义了7个权限：读、写、添加、执行、信贷、借记、查询。

然而，在Casbin中，我们将权限视为字符串。这种方式，您可以创建一些更适合您的权限。

3. Casbin RBAC支持域

在 Casbin 中，您可以通过域名进行授权。此功能使您的访问控制模型更加灵活。

ABAC

ABAC

什么是ABAC模式？

ABAC是 基于属性的访问控制，可以使用主体、客体或动作的属性，而不是字符串本身来控制访问。您之前可能就已经听过 XACML，是一个复杂的 ABAC 访问控制语言。与XACML相比，Casbin的ABAC非常简单：在ABAC中，可以使用struct(或基于编程语言的类实例) 而不是字符串来表示模型元素。

例如，ABAC的官方实例如下：

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == r.obj.Owner
```

我们在 `matcher` 中使用 `r.obj.Owner` 代替 `r.obj`。在 `Enforce()` 函数中传递的 `r.obj` 函数是结构或类实例，而不是字符串。Casbin将使用映像来检索 `obj`结构或类中的成员变量。

这里是 `r.obj construction` 或 `class` 的定义：

```
type testResource struct {
    Name string
    Owner string
}
```

如何使用ABAC？

简单地说，要使用ABAC，您需要做两件事：

1. 在模型匹配器中指定属性。
2. 将元素的结构或类实例作为Casbin的`Enforce()` 的参数传入。

WARNING 目前，仅有形如`r.sub`，`r.obj`，`r.act` 等请求元素支持ABAC。您不能在`policy`元素上使用它，比如`p.sub`，因为在Casbin的`policy`中没有定义结构或者类。

您可以在匹配器中使用多个ABAC属性，例如：`m = r.sub.Domain == r.obj.Domain`。

...

适配复杂且大量的ABAC规则

上述ABAC实施实例的核心非常简单。但授权系统通常需要非常复杂和大量的ABAC规则。为了满足这一需要上述实现将在很大程度上增加模型的详细程度。因此，通过在策略中添加规则代替模型中添加规则是明智的。这是通过引入一个 `eval()` 功能结构完成的。下面是管理此类ABAC模型的示例实例。

这是用于定义ABAC模型的 `CONF` 文件的定义。

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub_rule, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = eval(p.sub_rule) && r.obj == p.obj && r.act == p.act
```

在这里， `p.sub_rule` 是由策略中使用的必要属性组成的结构类型或类类型(用户定义类型)。

这是针对Enforcement模型使用的策略 现在您就可以使用作为参数传递到 `eval()`函数的对象实例来定义某些ABAC约束条件。

```
p, r.sub.Age > 18, /data1, read
p, r.sub.Age < 60, /data2, write
```

优先级模型

优先级模型

Casbin支持参考优先级加载策略。

####通过隐式优先级加载策略
这非常简单，顺序决定了策略的优先级，策略出现的越早优先级就越高。

model.conf:

```
[policy_effect]  
e = priority(p, eft) || deny
```

通过显式优先级加载策略

另见: casbin#550

策略的第一个要素一定是优先级，并且优先级值较小的将具有较高的优先地位。如果优先级有非数字字符，它将被排在最后，而不是导致报错。现在，明确的优先级仅支持 添加策略 & 添加策略，如果 升级策略 被调用，那么您不应该改变优先级属性。

model.conf:

```
[request_definition]  
r = sub, obj, act  
  
[policy_definition]  
p = priority, sub, obj, act, eft  
  
[role_definition]  
g = _, _  
  
[policy_effect]  
e = priority(p, eft) || deny  
  
[matchers]  
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

policy.csv

```
p, 10, data1_deny_group, data1, read, deny  
p, 10, data1_deny_group, data1, write, deny  
p, 10, data2_allow_group, data2, read, allow  
p, 10, data2_allow_group, data2, write, allow  
  
p, 1, alice, data1, write, allow  
p, 1, alice, data1, read, allow  
p, 1, bob, data2, read, deny  
  
g, bob, data2_allow_group  
g, alice, data1_deny_group
```

请求:

优先级模型

```
alice, data1, write --> true // `p, 1, alice, data1, write, allow` 拥有最高级的优先权
bob, data2, read --> false
bob, data2, write --> true // 对于bob是 `data2_allow_group` 的角色可以写入data2，而且没有具有更高优先级的否认策略
```

超级管理员

超级管理员

超级管理员是整个系统的管理员。我们可以在RBAC，ABAC以及带域的RBAC等模型中使用它 具体例子如下：

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act || r.sub == "root"
```

它说明了对于定义的 `request_definition`, `policy_definition`, `policy_effect` 和 `matcher`, Casbin 判断如果请求可以匹配的上策略，或者更重要的，如果 `sub` 是超级用户的话。一旦判决正确，就允许授权，而且用户有权做一切。就像Linux系统的root一样，用户被授权为 `root`, 我们就有访问所有文件和设置的权限。因此，如果我们想要 `sub` 能够完全访问整个系统。我们可以让它成为超级管理员，然后 `sub` 才有权做一切。

存储

存储

Model存储

Model存储

与 `policy` 不同，`model` 只能加载，不能保存。因为我们认为 `model` 不是动态组件，不应该在运行时进行修改，所以我们没有实现一个 API 来将 `model` 保存到存储中。

但是，好消息是，我们提供了三种等效的方法来静态或动态地加载模型：

从 `.CONF` 文件中加载 `model`

当你向 Casbin 团队寻求帮助时，他们会给你这个 Casbin 最常用的方法，此方法对于初学者来说很容易理解并且便于分享。

`.CONF`文件的内容 [examples/rbac_model.conf](#):

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

接着你可以加载模型文件如下：

```
e := casbin.NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

从代码加载 `model`

模型可以从代码中动态初始化，不需要使用 `.CONF`。下面是RBAC模型的一个例子：

```
import (
    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin/v2/model"
    "github.com/casbin/casbin/v2/persist/file-adapter"
)

// Initialize the model from Go code.
m := model.NewModel()
m.AddDef("r", "r", "sub, obj, act")
m.AddDef("p", "p", "sub, obj, act")
m.AddDef("g", "g", "_, _")
m.AddDef("e", "e", "some(where (p.eft == allow))")
m.AddDef("m", "m", "g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act")

// Load the policy rules from the .CSV file adapter.
// 使用自己的 adapter 替换。
a := fileadapter.NewAdapter("examples/rbac_policy.csv")
```

```
// Create the enforcer.  
e := casbin.NewEnforcer(m, a)
```

从字符串加载的 **model**

或者您可以从多行字符串加载整个模型文本。这种方法的优点是您不需要维护模型文件。

```
import (  
    "github.com/casbin/casbin/v2"  
    "github.com/casbin/casbin/v2/model"  
)  
  
// Initialize the model from a string.  
text :=  
`  
[request_definition]  
r = sub, obj, act  
  
[policy_definition]  
p = sub, obj, act  
  
[role_definition]  
g = _, _  
  
[policy_effect]  
e = some(where (p.eft == allow))  
  
[matchers]  
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act  
`  
m, _ := model.NewModelFromString(text)  
  
// Load the policy rules from the .CSV file adapter.  
// Replace it with your adapter to avoid files.  
a := model.NewModelFromString("examples/rbac_policy.csv")  
  
// Create the enforcer.  
e := casbin.NewEnforcer(m, a)
```


Policy存储

Policy存储

在Casbin中，策略存储作为adapter实现。请参照：</docs/en/adapters>。

政策子集加载

政策子集加载

一些adapter支持过滤策略管理。这意味着Casbin加载的策略是基于给定过滤器的存储策略的子集。当解析整个策略成为性能瓶颈时，这将会允许在大型多租户环境中有效地执行策略。

要使用支持的adapter处理过滤后的策略，只需调用 `LoadFilteredPolicy` 方法。过滤器参数的有效格式取决于所用的适配器。为了防止意外数据丢失，当策略已经加载，`SavePolicy` 方法会被禁用。

例如，下面的代码片段使用内置的过滤文件adapter和带有域的RBAC模型。在本例中，过滤器将策略限制为单个域。除“`domain1`”以外的任何域策略行被忽略：

```
import "github.com/casbin/casbin"

enforcer := casbin.NewEnforcer()

adapter := fileadapter.NewFilteredAdapter("examples/rbac_with_domains_policy.csv")
enforcer.InitWithAdapter("examples/rbac_with_domains_model.conf", adapter)

filter := &fileadapter.Filter{
  P: []string{"", "domain1"},
  G: []string{"", "", "domain1"},
}
enforcer.LoadFilteredPolicy(filter)

// 被加载的策略现在只包含属于domain1的入口
```

还有另一种方法支持子集加载功能：`LoadIncrementalFilteredPolicy`。`LoadIncrementalFilteredPolicy` 类似于 `LoadFilteredPolicy`，但是它没有清楚之前加载的策略，只是附加

扩充功能

扩充功能

适配器

适配器

在Casbin中，策略存储作为adapter(Casbin的中间件)实现。Casbin用户可以使用adapter从存储中加载策略规则 (aka LoadPolicy()) 或者将策略规则保存到其中 (aka SavePolicy())。为了保持代码轻量级，我们没有把adapter代码放在主库中。

目前支持的适配器列表

Casbin的适配器完整列表如下。我们欢迎任何第三方对adapter进行新的贡献，如果有请通知我们，我们将把它放在这个列表中:)

Go

适配器	类型	作者	自动保存
File Adapter (内置)	File	Casbin	<input type="checkbox"/>
Filtered File Adapter (内置)	File	@faceless-saint	<input type="checkbox"/>
SQL Adapter	SQL	@Blank-Xu	<input type="checkbox"/>
Xorm Adapter	ORM	Casbin	<input type="checkbox"/>
Gorm Adapter	ORM	Casbin	<input type="checkbox"/>

Ent Adapter	ORM	Casbin	<input type="checkbox"/>
Beego ORM Adapter	ORM	Casbin	<input type="checkbox"/>
SQLX Adapter	ORM	@memwey	<input type="checkbox"/>

SQLX Adapter 适配器	ORM 类型	@memwey 作者	<input type="checkbox"/> 自动保存
Sqlx Adapter	SQL	@Blank-Xu	<input type="checkbox"/>
GF ORM Adapter	ORM	@vance-liu	<input type="checkbox"/>
GoFrame ORM Adapter	ORM	@kotlin2018	<input type="checkbox"/>
Filtered PostgreSQL Adapter	SQL	Casbin	<input type="checkbox"/>
PostgreSQL Adapter	SQL	@cychiuae	<input type="checkbox"/>
RQLite Adapter	SQL	EDOMO Systems	<input type="checkbox"/>
MongoDB Adapter	NoSQL	Casbin	<input type="checkbox"/>
RethinkDB Adapter	NoSQL	@adityapandey9	<input type="checkbox"/>
Cassandra Adapter	NoSQL	Casbin	<input type="checkbox"/>
DynamoDB Adapter	NoSQL	HOOQ	<input type="checkbox"/>
Dynacasbin	NoSQL	NewbMiao	<input type="checkbox"/>
ArangoDB Adapter	NoSQL	@adamwasila	<input type="checkbox"/>
https://github.com/Soluto/casbin-minio-adapter	Cloud	Soluto	<input type="checkbox"/>
Azure Cosmos DB Adapter	Cloud	@spacycoder	<input type="checkbox"/>

GCP Firestore Adapter	Cloud	@reedom	<input type="checkbox"/>
GCP Cloud Storage Adapter	Cloud	qurami	<input type="checkbox"/>
Consul Adapter	KV store	@ankitm123	<input type="checkbox"/>

适配器	KV 类型 store	Casbin 作者	自动保存
Redis Adapter	KV store	@casbin	☑
Etcd Adapter	KV store	@sebastianliu	☐
BoltDB Adapter	KV stor	@speza	☐
Bolt Adapter	KV store	@wirepair	☐
BadgerDB Adapter	KV store	@inits	☐
Protobuf Adapter	Stream	Casbin	☐
JSON Adapter	String	Casbin	☐
String Adapter	String	@qiangmzsx	☐
HTTP File Adapter	HTTP	@h4ckedneko	用于http.FileS

NOTE

1. 如果使用显式或隐式adapter调用casbin.NewEnforcer(), 策略将自动加载。
2. 可以调用e.LoadPolicy() 来从存储中重新加载策略规则。
3. 如果adapter不支持Auto-Save特性, 则在添加或删除策略时不能将策略规则自动保存回存储器。你必须手动调用SavePolicy() 来保存所有的策略规则

例子

这里我们提供几个例子:

文件适配器 (内置)

以下代码展示了如何通过内置文件适配器初始化一个enforcer:

Go

```
import "github.com/casbin/casbin"

e := casbin.NewEnforcer("examples/basic_model.conf", "examples/basic_policy.csv")
```

PHP

```
use Casbin\Enforcer;

$e = new Enforcer('examples/basic_model.conf', 'examples/basic_policy.csv');
```

Rust

```
use casbin::prelude::*;

let mut e = Enforcer::new("examples/basic_model.conf", "examples/basic_policy.csv").await?;
```

它等同于如下代码:

Go

```
import (
    "github.com/casbin/casbin"
    "github.com/casbin/casbin/file-adapter"
)

a := fileadapter.NewAdapter("examples/basic_policy.csv")
e := casbin.NewEnforcer("examples/basic_model.conf", a)
```

PHP

```
use Casbin\Enforcer;
use Casbin\Persist\Adapters\FileAdapter;

$a = new FileAdapter('examples/basic_policy.csv');
$e = new Enforcer('examples/basic_model.conf', $a);
```

Rust

```
use casbin::prelude::*;

let a = FileAdapter::new("examples/basic_policy.csv");
let e = Enforcer::new("examples/basic_model.conf", a).await?;
```

MySQL 适配器

下面展示了如何从MySQL数据库初始化一个enforcer。此处样例中的MySQL数据库运行在127.0.0.1:3306上，用户为root，密码为空。

Go

```
import (
    "github.com/casbin/casbin"
    "github.com/casbin/mysql-adapter"
)

a := mysqladapter.NewAdapter("mysql", "root:@tcp(127.0.0.1:3306)/")
e := casbin.NewEnforcer("examples/basic_model.conf", a)
```

Rust

```
// https://github.com/casbin-rs/diesel-adapter
// 请确保您激活了 `mysql` 特性

use casbin::prelude::*;
use diesel_adapter::{ConnOptions, DieselAdapter};
```

```

let mut conn_opts = ConnOptions::default();
conn_opts
    .set_hostname("127.0.0.1")
    .set_port(3306)
    .set_host("127.0.0.1:3306") // overwrite hostname, port config
    .set_database("casbin")
    .set_auth("casbin_rs", "casbin_rs");

let a = DieselAdapter::new(conn_opts)?;
let mut e = Enforcer::new("examples/basic_model.conf", a).await?;

```

PHP

```

// https://github.com/php-casbin/dbal-adapter

use Casbin\Enforcer;
use CasbinAdapter\DBAL\Adapter as DatabaseAdapter;

$config = [
    // Either 'driver' with one of the following values:
    // pdo_mysql, pdo_sqlite, pdo_pgsql, pdo_oci (unstable), pdo_sqlsrv, pdo_sqlsrv,
    // mysqli, sqlanywhere, sqlsrv, ibm_db2 (unstable), drizzle_pdo_mysql
    'driver' => 'pdo_mysql',
    'host' => '127.0.0.1',
    'dbname' => 'test',
    'user' => 'root',
    'password' => '',
    'port' => '3306',
];

$a = DatabaseAdapter::newAdapter($config);
$e = new Enforcer('examples/basic_model.conf', $a);

```

使用自建的adapter

您可以按照如下所示的方式使用您自定义的适配器：

```

import (
    "github.com/casbin/casbin"
    "github.com/your-username/your-repo"
)

a := yourpackage.NewAdapter(params)
e := casbin.NewEnforcer("examples/basic_model.conf", a)

```

在运行时进行加载或保存配置信息

您可以在初始化后重新加载模型与策略，或保存新的策略：

```

// 从模型CONF文件重新加载模型。
e.LoadModel()

// 从文件或数据库中重新加载策略。
e.LoadPolicy()

// 保存当前策略（通常使用 Casbin API更改后）返回文件或数据库。
e.SavePolicy()

```


自动保存

自动保存机制 (Auto-Save) 是适配器的特性之一。如果当前适配器支持自动保存特性，该适配器可以向存储中添加一个策略规则，或者从存储中移除一个策略规则。这与SavePolicy()不同，因为后者将删除存储中的所有策略规则，并将所有策略规则从Casbin enforcer保存到存储中。因此，当策略规则的数量较多时，使用SavePolicy()可能会出现性能问题。

当适配器支持自动保存机制时，您可以通过Enforcer.EnableAutoSave() 函数来开启或关闭该机制。默认情况下启用该选项（如果适配器支持自动保存的话）。

NOTE

1. Auto-Save 特性是可选的。Adapter可以选择是否实现它。
2. Auto-Save 只在Casbin enforcer使用的adapter支持它时才有效。
3. 请参阅上面适配器列表中的自动保存 列，以查看自动保存是否有适配器支持。

下面是一个关于如何使用Auto-Save的例子：

```

import (
    "github.com/casbin/casbin"
    "github.com/casbin/xorm-adapter"
    _ "github.com/go-sql-driver/mysql"
)

```

// enforcer会默认开启AutoSave机制.

```

a := xormadapter.NewAdapter("mysql", "mysql_username:mysqlpassword@tcp(127.0.0.1:3306)/")
e := casbin.NewEnforcer("examples/basicmodel.conf", a)

```

// 禁用AutoSave机制

```
e.EnableAutoSave(false)
```

// 因为禁用了AutoSave，当前策略的改变只在内存中生效

// 这些策略在持久层中仍是不变的

```

e.AddPolicy(...)
e.RemovePolicy(...)

```

// 开启AutoSave机制

```
e.EnableAutoSave(true)
```

// 因为开启了AutoSave机制，现在内存中的改变会同步回写到持久层中

```

e.AddPolicy(...)
e.RemovePolicy(...)

```

有关更多示例，请参见：https://github.com/casbin/xorm-adapter/blob/master/adapter_test.go

如何编写 Adapter

Adapter应实现Adapter 中定义的接口，其中必须实现的为LoadPolicy(model model.Model) error和SavePolicy(model model.Model) error。

其他三个函数是可选的。如果adapter支持Auto-Save特性，则应该实现它们。

方法	**类型**	**描述**
LoadPolicy()	强制的	从存储中加载所有策略规则
SavePolicy()	强制的	将所有策略规则保存到存储中
AddPolicy()	可选的	向存储中添加策略规则
RemovePolicy()	可选的	从存储中删除策略规则
RemoveFilteredPolicy()	可选的	从存储中删除匹配筛选器的策略规则

NOTE

如果一个适配器不支持Auto-Save，它应该为以下三个可选函数提供一个空实现。 以下是Golang中的一个样例：

```
// AddPolicy 向存储器添加了一条策略规则。
func (a *Adapter) AddPolicy(sec string, ptype string, rule []string) error {
return errors.New("not implemented")
}

// RemovePolicy 从存储器中移除一条策略规则。
func (a *Adapter) RemovePolicy(sec string, ptype string, rule []string) error {
return errors.New("not implemented")
}

// RemoveFilteredPolicy 从存储器中移除可匹配过滤器的策略规则。
func (a *Adapter) RemoveFilteredPolicy(sec string, ptype string, fieldIndex int, fieldValues ...string) error {
return errors.New("not implemented")
}
```

Casbin enforcer在调用这三个可选实现的接口时，会忽略返回的not implemented 错误。

谁负责创建数据库？

作为约定，adapter应该能够自动创建一个名为casbin的数据库（如果它不存在的话），并将其用于策略存储。具体请参考Xorm adapter的实现：<https://github.com/casbin/xorm-adapter>

观察者

观察者

我们支持使用分布式消息系统，例如 [etcd](#) 来保持多个Casbin执行器实例之间的一致性。因此，我们的用户可以同时使用多个Casbin 执行器来处理大量的权限检查请求。

与策略存储 [adapters](#)类似，我们没有把[watcher](#)的代码放在主库中。任何对新消息系统的支持都应该作为[watcher](#)程序来实现。完整的Casbin [watchers](#)列表如下所示。欢迎任何第三方对 [watcher](#) 进行新的贡献，如果有请告知我们，我将把它放在这个列表中:)

Go

** 监视器**	类型	作者	说明
Etdc Watcher	KV store	Casbin	etcd 的监视器
Redis Watcher	KV store	Casbin	Watcher for Redis
Redis Watcher	KV store	@billcobbler	Watcher for Redis
Kafka Watcher	Messaging system	@wgarunap	Watcher for Apache Kafka
NATS Watcher	Messaging system	Solute	Watcher for NATS
ZooKeeper Watcher	Messaging system	Greprs	Watcher for Apache ZooKeeper
NATS, RabbitMQ, GCP Pub/Sub, AWS SNS & SQS, Kafka, InMemory	Messaging System	@rusenask	Watcher based on Go Cloud Dev Kit that works with leading cloud providers and self-hosted infrastructure
RocketMQ Watcher	Messaging system	@fmyxyz	Watcher for Apache RocketMQ

WatcherEx

In order to support incremental synchronization between multiple instances, we provide the [WatcherEx](#) interface. We hope it can notify other instances when the policy changes, but there is currently no implementation of [WatcherEx](#). We recommend that you use [dispatcher](#) to achieve this.

Compared with [Watcher](#) interface, with implementing [WatcherEx](#) what kind of update action can be distinguished, etc [AddPolicy](#), [RemovePolicy](#), etc.

[WatcherEx](#) Apis:

api	description
SetUpdateCallback(func(string)) error	SetUpdateCallback sets the callback that the watcher will call, when the has been changed by other instance callback is Enforcer.LoadPolicy().
Update() error	Update calls the update callback of instances to synchronize their policy usually called after changing the po like Enforcer.SavePolicy(), Enforcer./ Enforcer.RemovePolicy(), etc.
Close()	Close stops and releases the watche callback function will not be called a
UpdateForAddPolicy(params ...string) error	UpdateForAddPolicy calls the update other instances to synchronize their called after Enforcer.AddPolicy()
UpdateForRemoveFilteredPolicy(fieldIndex int, fieldValues ...string) error	UpdateForRemoveFilteredPolicy call update callback of other instances t synchronize their policy. It is called ; Enforcer.RemoveFilteredNamedGrou
UpdateForSavePolicy(model model.Model) error	UpdateForSavePolicy calls the upda of other instances to synchronize th is called after Enforcer.RemoveFilteredNamedGrou

调度器

调度器

调度器提供了同步递增策略变化的方法。它应以手工艺等一致性算法为基础，以确保所有执行者的一致性和一致性。通过调度器用户们可以轻松地建立分布式集群。

调度器的方法分为两部分。第一种是与Casbin相结合的方法。这些方法应该在Casbin内部调用。用户们可以使用由Casbin本身提供的更完整的api。

另一个部分是调度器本身定义的方法，包括调度器初始化方法，和不同算法提供的不同函数，如动态资格、配置变更等。

NOTE 我们希望调度器在运行时确保Casbin执行的一致性。因此，如果初始化时策略不一致，调度器将无法正常工作。用户在使用调度器之前需要确保所有实例的状态一致。

完整的Casbin调度器列表如下所示。我们欢迎来自任何第三方的调度器，请通知我们，以将您的调度器加入列表中:)

Go

调度器调度器	类型	作者	说明
Hashicorp Raft Dispatcher	raft	Casbin	基于 hashicorp/raft 的调度器

分布式执行

DistributedEnforcer 为调度器包装 SyncedEnforcer.

Go

```
e, _ := casbin.NewDistributedEnforcer("examples/basic_model.conf", "examples/basic_policy.csv")
```

角色管理器

角色管理器

角色管理器用于管理Casbin中的RBAC角色层次结构(用户角色映射)。角色管理器可以从Casbin策略规则或外部源(如LDAP、Okta、Auth0、Azure AD等)检索角色数据。我们支持角色管理器的不同实现。为了保持代码轻量级,我们没有把角色管理器代码放在主库中(默认的角色管理器除外)。下面提供了Casbin角色管理器的完整列表。欢迎任何第三方对角色管理器进行新的贡献,如果有请告知我们,我将把它放在这个列表中:)

Go

Role manager	作者	说明
Default Role Manager (built-in)	Casbin	支持存储在Casbin策略中的角色层次结构
Session Role Manager	EDOMO Systems	支持存储在Casbin策略中的角色层次结构,以及基于时间范围的会话
Okta Role Manager	Casbin	Supports role hierarchy stored in Okta
Auth0 Role Manager	Casbin	Supports role hierarchy stored in Auth0's Authorization Extension

For developers: all role managers must implement the [RoleManager](#) interface. [Session Role Manager](#) can be used as a reference implementation.

中间件

中间件

中间件

Web框架

高性能、简约的web框架，通过插件：`echo-authz` (感谢 [@xqbumu](#)) 或者 `casbinrest` 实现

GraphQL Middlewares

GraphQL Middlewares

Casbin follows the officially suggested way to provide authorization for GraphQL endpoints by having a single source of truth for authorization: <https://graphql.org/learn/authorization/> . In another word, Casbin should be placed between GraphQL layer and your business logic.

```
// Casbin authorization logic lives inside postRepository
var postRepository = require('postRepository');

var postType = new GraphQLObjectType({
  name: 'Post',
  fields: {
    body: {
      type: GraphQLString,
      resolve: (post, args, context, { rootValue }) => {
        return postRepository.getBody(context.user, post);
      }
    }
  }
});
```

Supported GraphQL middlewares

A complete list of Casbin GraphQL middlewares is provided as below. Any 3rd-party contribution on a new GraphQL middleware is welcomed, please inform us and we will put it in this list:)

Go

<i>*Middleware *</i>	GraphQL Implementation	Author	Descri
graphql-casbin	graphql	@esmaeilpour	An Implem of using Graphq Casbin togethe
gqlgen_casbin_RBAC_example	gqlgen	@WenyXu	(empty)

Cloud Native

Cloud Native

With the Cloud Native developing rapidly, Casbin community is also actively participating in the working on supporting Cloud Native plugins.

Here are some Cloud Native projects relating to Casbin, welcome to add your projects to the table.

Go

Project	** Author **	Description
traefik	Traefik Labs	Traefik is a modern HTTP reverse proxy and load balancer that makes deploying microservices easy

We are also working on Kubenetes plugins which can implement the management of K8S clusters, please [stay tuned](#). If you are interested in developing the project, welcome to join in [Casbin community](#)!!

API

API

API Overview

API Overview

This overview only shows you how to use Casbin APIs and doesn't explain how Casbin is installed and how it works. You can find those tutorials here: [installation of Casbin](#) and [how Casbin works](#). So, when you start reading this tutorial, we assume that you have fully installed and imported Casbin into your code.

Enforce API

Let's start at the Enforce APIs of Casbin. We will load a RBAC model from model.conf, and load policies from policy.csv. You can learn the Model syntax [here](#), and we won't talk about it in this tutorial. We assume that you can understand the config files given below:

model.conf

```
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

[policy_effect]
e = some(where (p.eft == allow))

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

policy.csv

```
p, admin, data1, read
p, admin, data1, write
p, admin, data2, read
p, admin, data2, write
p, alice, data1, read
p, bob, data2, write
g, amber, admin
g, abc, admin
```

After reading the config files, please read the following code.

```
// load information from files
enforcer, err := casbin.NewEnforcer("./example/model.conf", "./example/policy.csv")
if err != nil {
    log.Fatalf("error, detail: %s", err)
}
ok, err := enforcer.Enforce("alice", "data1", "read")
```

This code loads the access control model and policies from local files. Function `casbin.NewEnforcer()` will return an enforcer. It will recognize its 2 parameters as file paths, and load the files from there. Errors occurred in the

process are stored in err. This code used the default adapter to load model and policies. And of course you can get the same result by using a third-party adapter.

Code `ok, err := enforcer.Enforce("alice", "data1", "read")` is to confirm access permissions. If alice can access the data1 with the operation read, the returned value ok will be true, otherwise it'll be false. In this example, the value of ok is true.

EnforceEx API

Sometimes you may wonder which policy allowed the request, so we prepared the function `EnforceEx()`. You can use it like this:

```
ok, reason, err := enforcer.EnforceEx("amber", "data1", "read")
fmt.Println(ok, reason) // true [admin data1 read]
```

function `EnforceEx()` will return the exact policy string in the return value reason. In this example, amber is a role of admin, so policy p, admin, data1, read made this request true. The out put of this code is in the comment.

Casbin prepared a lot of APIs like this. Those APIs added some extra functions on the basic one. They are:

- `ok, err := enforcer.EnforceWithMatcher(matcher, request)`

With a matcher.

- `ok, reason, err := enforcer.EnforceExWithMatcher(matcher, request)`

A combination of `EnforceWithMatcher()` and `EnforceEx()`.

- `boolArray, err := enforcer.BatchEnforce(requests)`

Do a list job, returns an array.

This is a simple use of Casbin. You can use Casbin to start an authorization server via these APIs. We will show you some other types of APIs in the next paragraphs.

Management API

Get API

These APIs are used to get exact objects in policies. This time we loaded an enforcer like the last example and get something from it.

Please read the following code:

```
enforcer, err := casbin.NewEnforcer("./example/model.conf", "./example/policy.csv")
if err != nil {
    fmt.Printf("Error, details: %s\n", err)
}
allSubjects := enforcer.GetAllSubjects()
fmt.Println(allSubjects)
```

Same as the last example, the first 4 lines loaded some necessary information from local files. We won't talk about that here anymore.

Code `allSubjects := enforcer.GetAllSubjects()` got all the subjects in the policy file and returned them as an array. Then we printed that array.

Normally, the output of the code should be:

```
[admin alice bob]
```

You can also change the function `GetAllSubjects()` to `GetAllNamedSubjects()`, to get the list of subjects that show up in the current named policy.

Similarly, we prepared `GetAll` functions for Objects, Actions, Roles. The only thing you need to do is to change the word `Subject` in the function name to what you want if you want to access those functions.

Besides, we have more getters for policies. The call method and return value are similar to those above.

- `policy = e.GetPolicy()` gets all the authorization rules in the policy.
- `filteredPolicy := e.GetFilteredPolicy(0, "alice")` gets all the authorization rules in the policy, field filters can be specified.
- `namedPolicy := e.GetNamedPolicy("p")` gets all the authorization rules in the named policy.
- `filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "bob")` gets all the authorization rules in the named policy, field filters can be specified.
- `groupingPolicy := e.GetGroupingPolicy()` gets all the role inheritance rules in the policy.
- `filteredGroupingPolicy := e.GetFilteredGroupingPolicy(0, "alice")` gets all the role inheritance rules in the policy, field filters can be specified.
- `namedGroupingPolicy := e.GetNamedGroupingPolicy("g")` gets all the role inheritance rules in the policy.
- `namedGroupingPolicy := e.GetFilteredNamedGroupingPolicy("g", 0, "alice")` gets all the role inheritance rules in the policy.

Add, Delete, Update API

Casbin prepared a lot of APIs for policies. These APIs allow you to add, delete or edit policies dynamically at runtime.

This code shows you how to add, remove and update your policies, and told you how to confirm that a policy exists:

```
// load information from files
enforcer, err := casbin.NewEnforcer("./example/model.conf", "./example/policy.csv")
if err != nil {
    fmt.Printf("Error, details: %s\n", err)
}

// add a policy, then use HasPolicy() to confirm that
enforcer.AddPolicy("added_user", "data1", "read")
hasPolicy := enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // true, we added that policy successfully

// remove a policy, then use HasPolicy() to confirm that
enforcer.RemovePolicy("alice", "data1", "read")
hasPolicy = enforcer.HasPolicy("alice", "data1", "read")
fmt.Println(hasPolicy) // false, we deleted that policy successfully

// update a policy, then use HasPolicy() to confirm that
enforcer.UpdatePolicy([]string{"added_user", "data1", "read"}, []string{"added_user", "data1", "write"})
```

```
hasPolicy = enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // false, the origin policy has lapsed
hasPolicy = enforcer.HasPolicy("added_user", "data1", "write")
fmt.Println(hasPolicy) // true, the new policy is in use
```

Using these four kinds of APIs can edit your policies. Like these, we prepared some kinds of APIs to FilteredPolicy, NamedPolicy, FilteredNamedPolicy, GroupingPolicy, NamedGroupingPolicy, FilteredGroupingPolicy, FilteredNamedGroupingPolicy. To use them, you only need to replace word Policy in the function name to the words above.

Besides, if you change parameters to arrays, you can batch edit you policies.

For example, to functions like this:

```
enforcer.UpdatePolicy([]string{"eve", "data3", "read"}, []string{"eve", "data3", "write"})
```

if we change Policy to Policies, and edit the parameter to:

```
enforcer.UpdatePolicies([][]string{{"eve", "data3", "read"}, {"jack", "data3", "read"}}, [][]string{{"eve", "data3", "write"}, {"jack", "data3", "write"}})
```

then we can batch edit these policies.

Same operations also useful to GroupingPolicy, NamedGroupingPolicy.

RBAC API

Casbin provided come APIs for you to modify the RBAC model and policies. If you are familiar with RBAC, you can use these APIs easily.

Here we only show you how to use RBAC APIs of Casbin and won't talk about RBAC itself. You can get more details [here](#).

We use this code to load model and policies just like before.

```
enforcer, err := casbin.NewEnforcer("./example/model.conf", "./example/policy.csv")
if err != nil {
    fmt.Printf("Error, details: %s\n", err)
}
```

then, use a instance of Enforcer enforcer to access these APIs.

```
roles, err := enforcer.GetRolesForUser("amber")
fmt.Println(roles) // [admin]
users, err := enforcer.GetUsersForRole("admin")
fmt.Println(users) // [amber abc]
```

GetRolesForUser() returned an array that contained all roles contained amber. In this example, amber has only one role admin, so array roles is [admin]. And similarly, you can use GetUsersForRole() to get users belongs to the role. The return value of this function is also an array.

```
enforcer.HasRoleForUser("amber", "admin") // true
```

You can use `HasRoleForUser()` to confirm whether the user belongs to the role. In this example, amber is a member of admin, so the return value of the function is true.

```
fmt.Println(enforcer.Enforce("bob", "data2", "write")) // true
enforcer.DeletePermission("data2", "write")
fmt.Println(enforcer.Enforce("bob", "data2", "write")) // false
```

You can use `DeletePermission()` to delete a permission.

```
fmt.Println(enforcer.Enforce("alice", "data1", "read")) // true
enforcer.DeletePermissionForUser("alice", "data1", "read")
fmt.Println(enforcer.Enforce("alice", "data1", "read")) // false
```

And use `DeletePermissionForUser()` to delete a permission for a user.

Casbin have a lot of APIs like this. Their call methods and return values have the same style as the above APIs. You can find these APIs in [next documents](#).

管理 API

管理 API

提供对Casbin策略管理完全支持的基本API。

Filtered API

Almost all filtered api has the same parameters (fieldIndex int, fieldValues ...string). fieldIndex is the index where matching start, fieldValues denotes the values result should have. Note that empty string in fieldValues could be any word.

Example:

```
p, alice, book, read
p, bob, book, read
p, bob, book, write
p, alice, pen, get
p, bob, pen, get
```

```
e.GetFilteredPolicy(1, "book") // will return: [[alice book read] [bob book read] [bob book write]]
e.GetFilteredPolicy(1, "book", "read") // will return: [[alice book read] [bob book read]]
e.GetFilteredPolicy(0, "alice", "", "read") // will return: [[alice book read]]
e.GetFilteredPolicy(0, "alice") // will return: [[alice book read] [alice pen get]]
```

Reference

global variable e is Enforcer instance.

Go

```
e, err := NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

Enforce()

Enforce decides whether a "subject" can access a "object" with the operation "action", input parameters are usually: (sub, obj, act).

For example:

Go

```
ok, err := e.Enforce(request)
```

EnforceWithMatcher()

EnforceWithMatcher use a custom matcher to decides whether a "subject" can access a "object" with the operation "action", input parameters are usually: (matcher, sub, obj, act), use model matcher by default when matcher is "".

例如:

Go

```
ok, err := e.EnforceWithMatcher(matcher, request)
```

EnforceEx()

EnforceEx explain enforcement by informing matched rules.

例如:

Go

```
ok, reason, err := e.EnforceEx(request)
```

EnforceExWithMatcher()

EnforceExWithMatcher use a custom matcher and explain enforcement by informing matched rules.

例如:

Go

```
ok, reason, err := e.EnforceExWithMatcher(matcher, request)
```

BatchEnforce()

BatchEnforce enforces each request and returns result in a bool array

例如:

Go

```
boolArray, err := e.BatchEnforce(requests)
```

GetAllSubjects()

GetAllSubjects gets the list of subjects that show up in the current policy.

例如:

Go

```
allSubjects := e.GetAllSubjects()
```

GetAllNamedSubjects()

GetAllNamedSubjects gets the list of subjects that show up in the current named policy.

例如:

Go

```
allNamedSubjects := e.GetAllNamedSubjects("p")
```

GetAllObjects()

GetAllObjects gets the list of objects that show up in the current policy.

例如:

Go

```
allObjects := e.GetAllObjects()
```

GetAllNamedObjects()

GetAllNamedObjects gets the list of objects that show up in the current named policy.

例如:

Go

```
allNamedObjects := e.GetAllNamedObjects("p")
```

GetAllActions()

GetAllActions gets the list of actions that show up in the current policy.

例如:

Go

```
allActions := e.GetAllActions()
```

GetAllNamedActions()

GetAllNamedActions gets the list of actions that show up in the current named policy.

例如:

Go

```
allNamedActions := e.GetAllNamedActions("p")
```

GetAllRoles()

GetAllRoles gets the list of roles that show up in the current policy.

例如:

Go

```
allRoles = e.GetAllRoles()
```

GetAllNamedRoles()

GetAllNamedRoles gets the list of roles that show up in the current named policy.

例如:

Go

```
allNamedRoles := e.GetAllNamedRoles("g")
```

GetPolicy()

GetPolicy gets all the authorization rules in the policy.

例如:

Go

```
policy = e.GetPolicy()
```

GetFilteredPolicy()

GetFilteredPolicy gets all the authorization rules in the policy, field filters can be specified.

例如:

Go

```
filteredPolicy := e.GetFilteredPolicy(0, "alice")
```

GetNamedPolicy()

GetNamedPolicy gets all the authorization rules in the named policy.

例如:

Go

```
namedPolicy := e.GetNamedPolicy("p")
```

GetFilteredNamedPolicy()

GetFilteredNamedPolicy gets all the authorization rules in the named policy, field filters can be specified.

例如:

Go

```
filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "bob")
```

GetGroupingPolicy()

GetGroupingPolicy gets all the role inheritance rules in the policy.

例如:

Go

```
groupingPolicy := e.GetGroupingPolicy()
```

GetFilteredGroupingPolicy()

GetFilteredGroupingPolicy gets all the role inheritance rules in the policy, field filters can be specified.

例如:

Go

```
filteredGroupingPolicy := e.GetFilteredGroupingPolicy(0, "alice")
```

GetNamedGroupingPolicy()

GetNamedGroupingPolicy gets all the role inheritance rules in the policy.

例如:

Go

```
namedGroupingPolicy := e.GetNamedGroupingPolicy("g")
```

GetFilteredNamedGroupingPolicy()

GetFilteredNamedGroupingPolicy gets all the role inheritance rules in the policy.

例如:

Go

```
namedGroupingPolicy := e.GetFilteredNamedGroupingPolicy("g", 0, "alice")
```

HasPolicy()

HasPolicy determines whether an authorization rule exists.

例如:

Go

```
hasPolicy := e.HasPolicy("data2_admin", "data2", "read")
```

HasNamedPolicy()

HasNamedPolicy determines whether a named authorization rule exists.

例如:

Go

```
hasNamedPolicy := e.HasNamedPolicy("p", "data2_admin", "data2", "read")
```

AddPolicy()

AddPolicy adds an authorization rule to the current policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

例如:

Go

```
added := e.AddPolicy("eve", "data3", "read")
```

AddPolicies()

AddPolicies adds authorization rules to the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesAdded := e.AddPolicies(rules)
```

AddNamedPolicy()

AddNamedPolicy adds an authorization rule to the current named policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

例如:

Go

```
added := e.AddNamedPolicy("p", "eve", "data3", "read")
```

AddNamedPolicies()

AddNamedPolicies adds authorization rules to the current named policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesAdded := e.AddNamedPolicies("p", rules)
```

RemovePolicy()

RemovePolicy removes an authorization rule from the current policy.

例如:

Go

```
removed := e.RemovePolicy("alice", "data1", "read")
```

RemovePolicies()

RemovePolicies removes authorization rules from the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesRemoved := e.RemovePolicies(rules)
```

RemoveFilteredPolicy()

RemoveFilteredPolicy removes an authorization rule from the current policy, field filters can be specified. RemovePolicy removes an authorization rule from the current policy.

例如:

Go

```
removed := e.RemoveFilteredPolicy(0, "alice", "data1", "read")
```

RemoveNamedPolicy()

RemoveNamedPolicy removes an authorization rule from the current named policy.

例如:

Go

```
removed := e.RemoveNamedPolicy("p", "alice", "data1", "read")
```

RemoveNamedPolicies()

RemoveNamedPolicies removes authorization rules from the current named policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are

consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesRemoved := e.RemoveNamedPolicies("p", rules)
```

RemoveFilteredNamedPolicy()

RemoveFilteredNamedPolicy removes an authorization rule from the current named policy, field filters can be specified.

例如:

Go

```
removed := e.RemoveFilteredNamedPolicy("p", 0, "alice", "data1", "read")
```

HasGroupingPolicy()

HasGroupingPolicy determines whether a role inheritance rule exists.

例如:

Go

```
has := e.HasGroupingPolicy("alice", "data2_admin")
```

HasNamedGroupingPolicy()

HasNamedGroupingPolicy determines whether a named role inheritance rule exists.

例如:

Go

```
has := e.HasNamedGroupingPolicy("g", "alice", "data2_admin")
```

AddGroupingPolicy()

AddGroupingPolicy adds a role inheritance rule to the current policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

例如:

Go

```
added := e.AddGroupingPolicy("group1", "data2_admin")
```

AddGroupingPolicies()

AddGroupingPolicies adds role inheritance rules to the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all authorization the rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesAdded := e.AddGroupingPolicies(rules)
```

AddNamedGroupingPolicy()

AddNamedGroupingPolicy adds a named role inheritance rule to the current policy. If the rule already exists, the function returns false and the rule will not be added. Otherwise the function returns true by adding the new rule.

例如:

Go

```
added := e.AddNamedGroupingPolicy("g", "group1", "data2_admin")
```

AddNamedGroupingPolicies()

AddNamedGroupingPolicies adds named role inheritance rules to the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is added to the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is added to the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesAdded := e.AddNamedGroupingPolicies("g", rules)
```

RemoveGroupingPolicy()

RemoveGroupingPolicy removes a role inheritance rule from the current policy.

例如:

Go

```
removed := e.RemoveGroupingPolicy("alice", "data2_admin")
```

RemoveGroupingPolicies()

RemoveGroupingPolicies removes role inheritance rules from the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

例如:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesRemoved := e.RemoveGroupingPolicies(rules)
```

RemoveFilteredGroupingPolicy()

RemoveFilteredGroupingPolicy removes a role inheritance rule from the current policy, field filters can be specified.

例如:

Go

```
removed := e.RemoveFilteredGroupingPolicy(0, "alice")
```

RemoveNamedGroupingPolicy()

RemoveNamedGroupingPolicy removes a role inheritance rule from the current named policy.

例如:

Go

```
removed := e.RemoveNamedGroupingPolicy("g", "alice")
```

RemoveNamedGroupingPolicies()

RemoveNamedGroupingPolicies removes named role inheritance rules from the current policy. The operation is atomic in nature. Hence, if authorization rules consists of rules which are not consistent with the current policy, the function returns false and no policy rule is removed from the current policy. If all the authorization rules are consistent with the policy rules, the function returns true and each policy rule is removed from the current policy.

For example:

Go

```
rules := [][] string {
    []string {"jack", "data4", "read"},
    []string {"katy", "data4", "write"},
    []string {"leyo", "data4", "read"},
    []string {"ham", "data4", "write"},
}

areRulesRemoved := e.RemoveNamedGroupingPolicies("g", rules)
```

RemoveFilteredNamedGroupingPolicy()

RemoveFilteredNamedGroupingPolicy removes a role inheritance rule from the current named policy, field filters can be specified.

例如:

Go

```
removed := e.RemoveFilteredNamedGroupingPolicy("g", 0, "alice")
```

UpdatePolicy()

UpdatePolicy update a old policy to new policy.

例如:

Go

```
updated, err := e.UpdatePolicy([]string{"eve", "data3", "read"}, []string{"eve", "data3", "write"})
```

UpdatePolicies()

UpdatePolicies updates all old policies to new policies.

例如:

Go

```
updated, err := e.UpdatePolicies([][]string{{"eve", "data3", "read"}, {"jack", "data3", "read"}}, [][]string{{"eve", "data3", "write"}, {"jack", "data3", "write"}})
```

AddFunction()

AddFunction adds a customized function.

例如:

Go

```
func CustomFunction(key1 string, key2 string) bool {
    if key1 == "/alice_data2/myid/using/res_id" && key2 == "/alice_data/:resource" {
        return true
    } else if key1 == "/alice_data2/myid/using/res_id" && key2 == "/alice_data2/:id/using/:resId" {
        return true
    }
}
```

```

    } else {
        return false
    }
}

func CustomFunctionWrapper(args ...interface{}) (interface{}, error) {
    key1 := args[0].(string)
    key2 := args[1].(string)

    return bool(CustomFunction(key1, key2)), nil
}

e.AddFunction("keyMatchCustom", CustomFunctionWrapper)

```

LoadFilteredPolicy()

LoadFilteredPolicy loads filtered policies from file/database.

例如:

Go

```
err := e.LoadFilteredPolicy()
```

LoadIncrementalFilteredPolicy()

LoadIncrementalFilteredPolicy append a filtered policy from file/database.

例如:

Go

```
err := e.LoadIncrementalFilteredPolicy()
```

UpdateGroupingPolicy()

UpdateGroupingPolicy updates oldRule to newRulein g section

例如:

Go

```
succeed, err := e.UpdateGroupingPolicy([]string{"data3_admin", "data4_admin"}, []string{"admin", "data4_admin"})
```

UpdateNamedGroupingPolicy()

UpdateNamedGroupingPolicy updates oldRule named ptype to newRulein g section

For example:

Go

```
succeed, err := e.UpdateGroupingPolicy("g1", []string{"data3_admin", "data4_admin"}, []string{"admin", "data4_admin"})
```

RBAC API

RBAC API

一个更友好的RBAC API。这个API是Management API的子集。RBAC用户可以使用这个API来简化代码。

参考

全局变量 `e` 是实施者实例。

Go

```
e, err := NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

GetRolesForUser()

GetRolesForUser 获取用户具有的角色。

例如:

Go

```
res := e.GetRolesForUser("alice")
```

GetUsersForRole()

GetUsersForRole 获取具有角色的用户。

例如:

Go

```
res := e.GetUsersForRole("data1_admin")
```

HasRoleForUser()

HasRoleForUser 确定用户是否具有角色。

例如:

Go

```
res := e.HasRoleForUser("alice", "data1_admin")
```

AddRoleForUser()

AddRoleForUser 为用户添加角色。如果用户已经拥有该角色（aka不受影响），则返回false。

例如:

Go

```
e.AddRoleForUser("alice", "data2_admin")
```

AddRolesForUser()

AddRolesForUser adds multiple roles for a user. Returns false if the user already has one of these roles (aka not affected).

For example:

Rust

```
let roles = vec!["data1_admin".to_owned(), "data2_admin".to_owned()];
let all_added = e.add_roles_for_user("alice", roles, None).await?; // No domain
```

DeleteRoleForUser()

DeleteRoleForUser deletes a role for a user. Returns false if the user does not have the role (aka not affected).

例如:

Go

```
e.DeleteRoleForUser("alice", "data1_admin")
```

DeleteRolesForUser()

DeleteRolesForUser deletes all roles for a user. Returns false if the user does not have any roles (aka not affected).

例如:

Go

```
e.DeleteRolesForUser("alice")
```

DeleteUser()

DeleteUser deletes a user. Returns false if the user does not exist (aka not affected).

例如:

Go

```
e.DeleteUser("alice")
```

DeleteRole()

DeleteRole deletes a role.

例如:

Go

```
e.DeleteRole("data2_admin")
```

DeletePermission()

DeletePermission deletes a permission. Returns false if the permission does not exist (aka not affected).

例如:

Go

```
e.DeletePermission("read")
```

AddPermissionForUser()

AddPermissionForUser adds a permission for a user or role. Returns false if the user or role already has the permission (aka not affected).

例如:

Go

```
e.AddPermissionForUser("bob", "read")
```

AddPermissionsForUser()

AddPermissionsForUser adds multiple permissions for a user or role. Returns false if the user or role already has one of the permissions (aka not affected).

例如:

Rust

```
let permissions = vec![
    vec!["data1".to_owned(), "read".to_owned()],
    vec!["data2".to_owned(), "write".to_owned()],
];

let all_added = e.add_permissions_for_user("bob", permissions).await?;
```

DeletePermissionForUser()

DeletePermissionForUser deletes a permission for a user or role. Returns false if the user or role does not have the permission (aka not affected).

例如:

Go

```
e.DeletePermissionForUser("bob", "read")
```

DeletePermissionsForUser()

DeletePermissionsForUser deletes permissions for a user or role. Returns false if the user or role does not have any permissions (aka not affected).

例如:

Go

```
e.DeletePermissionsForUser("bob")
```

GetPermissionsForUser()

GetPermissionsForUser gets permissions for a user or role.

For example:

Go

```
e.GetPermissionsForUser("bob")
```

HasPermissionForUser()

HasPermissionForUser determines whether a user has a permission.

例如:

Go

```
e.HasPermissionForUser("alice", []string{"read"})
```

GetImplicitRolesForUser()

GetImplicitRolesForUser gets implicit roles that a user has. Compared to GetRolesForUser(), this function retrieves indirect roles besides direct roles.

For example:

g, alice, role:admin
g, role:admin, role:user

GetRolesForUser("alice") can only get: ["role:admin"].
But GetImplicitRolesForUser("alice") will get: ["role:admin", "role:user"].

For example:

Go

```
e.GetImplicitRolesForUser("alice")
```

GetImplicitUsersForRole()

GetImplicitUsersForRole gets all users inheriting the role. Compared to GetUsersForRole(), this function retrieves indirect users.

For example:

g, alice, role:admin
g, role:admin, role:user

GetUsersForRole("role:user") can only get: ["role:admin"].
But GetImplicitUsersForRole("role:user") will get: ["role:admin", "alice"].

例如:

Go

```
users := e.GetImplicitUsersForRole("role:user")
```

GetImplicitPermissionsForUser()

GetImplicitPermissionsForUser gets implicit permissions for a user or role.
Compared to GetPermissionsForUser(), this function retrieves permissions for inherited roles.

For example:
p, admin, data1, read
p, alice, data2, read
g, alice, admin

GetPermissionsForUser("alice") can only get: [["alice", "data2", "read"]].
But GetImplicitPermissionsForUser("alice") will get: [["admin", "data1", "read"], ["alice", "data2", "read"]].

For example:

Go

```
e.GetImplicitPermissionsForUser("alice")
```

GetDomainsForUser()

GetDomainsForUser gets all domains which a user has.

For example: p, admin, domain1, data1, read p, admin, domain2, data2, read p, admin, domain2, data2, write
g, alice, admin, domain1 g, alice, admin, domain2

GetDomainsForUser("alice") could get ["domain1", "domain2"]

For example:

Go

```
result, err := e.GetDomainsForUser("alice")
```

GetImplicitResourcesForUser()

GetImplicitResourcesForUser returns all policies that should be true for user.

For example:

```
p, alice, data1, read
p, bob, data2, write
p, data2_admin, data2, read
p, data2_admin, data2, write

g, alice, data2_admin
```

GetImplicitResourcesForUser("alice") will return [[alice data1 read] [alice data2 read] [alice data2 write]]

Go

```
resources, err := e.GetImplicitResourcesForUser("alice")
```


RBAC with Domains API

RBAC with Domains API

A more friendly API for RBAC with domains. This API is a subset of Management API. The RBAC users could use this API to simplify the code.

Reference

global variable e is Enforcer instance.

Go

```
e, err := NewEnforcer("examples/rbac_model.conf", "examples/rbac_policy.csv")
```

GetUsersForRoleInDomain()

GetUsersForRoleInDomain 获取具有域内角色的用户。

例如:

Go

```
res := e.GetUsersForRoleInDomain("alice", "domain1")
```

GetRolesForUserInDomain()

GetRolesForUserInDomain gets the roles that a user has inside a domain.

For example:

Go

```
res := e.GetRolesForUserInDomain("admin", "domain1")
```

GetPermissionsForUserInDomain()

GetPermissionsForUserInDomain gets permissions for a user or role inside a domain.

For example:

Go

```
res := e.GetPermissionsForUserInDomain("alice", "domain1")
```

AddRoleForUserInDomain()

AddRoleForUserInDomain adds a role for a user inside a domain. Returns false if the user already has the role (aka not affected).

For example:

Go

```
ok, err := e.AddRoleForUserInDomain("alice", "admin", "domain1")
```

DeleteRoleForUserInDomain()

DeleteRoleForUserInDomain deletes a role for a user inside a domain. Returns false if the user does not have the role (aka not affected).

For example:

Go

```
ok, err := e.DeleteRoleForUserInDomain("alice", "admin", "domain1")
```

DeleteRolesForUserInDomain()

DeleteRolesForUserInDomain deletes all roles for a user inside a domain. Returns false if the user does not have any roles (aka not affected).

For example:

Go

```
ok, err := e.DeleteRolesForUserInDomain("alice", "domain1")
```

GetAllUsersByDomain()

GetAllUsersByDomain would get all users associated with the domain. Returns empty string array if has no domain defined in model.

For example:

Go

```
res := e.GetAllUsersByDomain("domain1")
```

DeleteAllUserByDomain()

DeleteAllUsersByDomain would delete all users associated with the domain. Returns false if has no domain defined in model.

For example:

Go

```
ok, err := e.GetUsersForRoleInDomain("alice", "domain1")
```

DeleteDomains()

DeleteDomains would delete all associated users and roles. It would delete all domains if parameter is not provided.

For example:

Go

```
ok, err := e.DeleteDomains("domain1", "domain2")
```

NOTE If you are handling a domain like name::domain, it may lead to unexpected behavior. In Casbin, :: is a reversed keyword, just like for, if in a programming language, we should never put :: in a domain.

RoleManager API

RoleManager API

RoleManager

RoleManager provides interface to define the operations for managing roles. Adding matching function to rolemanager allows using wildcards in role name and domain.

AddNamedMatchingFunc()

AddNamedMatchingFunc add MatchingFunc by ptype RoleManager. MatchingFunc将在操作角色匹配时工作。

```
e.AddNamedMatchingFunc("g", "", util.KeyMatch)
_, _ = e.AddGroupingPolicies([][]string{{"*", "admin", "domain"}})
_, _ = e.GetRoleManager().HasLink("bob", "admin", "domain") // -> true, nil
```

例如:

Go

```
e, _ := casbin.NewEnforcer("path/to/model", "path/to/policy")
e.AddNamedMatchingFunc("g", "", util.MatchKey)
```

AddNamedDomainMatchingFunc()

AddNamedDomainMatchingFunc 通过 ptype 把MatchingFunc 添加到 RoleManager中。 DomainMatchingFunc 类似于上面列出的 MatchingFunc

例如:

Go

```
e, _ := casbin.NewEnforcer("path/to/model", "path/to/policy")
e.AddNamedDomainMatchingFunc("g", "", util.MatchKey)
```

GetRoleManager()

GetRoleManager 获取现存的 g 的role manager。

例如:

Go

```
rm := e.GetRoleManager()
```

Clear()

Clear清除所有存储的数据并将角色管理器重置到初始状态。

例如:

Go

```
rm.Clear()
```

AddLink()

AddLink 添加了两个角色之间的继承链接。角色: 名称1 和 角色: 名称2 Domain is a prefix to the roles (can be used for other purposes).

例如:

Go

```
rm.AddLink("u1", "g1", "domain1")
```

DeleteLink()

DeleteLink deletes the inheritance link between two roles. role: name1 and role: name2. Domain is a prefix to the roles (can be used for other purposes).

例如:

Go

```
rm := DeleteLink("u1", "g1", "domain1")
```

HasLink()

HasLink determines whether a link exists between two roles. role: name1 inherits role: name2. Domain is a prefix to the roles (can be used for other purposes).

例如:

Go

```
rm.HasLink("u1", "g1", "domain1")
```

GetRoles()

GetRoles gets the roles that a user inherits. Domain is a prefix to the roles (can be used for other purposes).

例如:

Go

```
rm.GetRoles("u1", "domain1")
```

GetUsers()

GetUsers gets the users that inherits a role. Domain is a prefix to the users (can be used for other purposes).

例如:

Go

```
rm.GetUsers("g1")
```

PrintRoles()

PrintRoles prints all the roles to log.

例如:

Go

```
rm.PrintRoles()
```

SetLogger()

SetLogger设置角色管理器的日志。

例如:

Go

```
logger := log.DefaultLogger{}
logger.EnableLog(true)
rm.SetLogger(&logger)
_ = rm.PrintRoles()
```

GetDomains()

GetDomains gets domains that a user has

For example:

Go

```
result, err := rm.GetDomains(name)
```

Data Permissions

Data Permissions

We have two solutions for data permissions (filtering). Using implicit assignment APIs. Or just use BatchEnforce() API.

1. Query implicit roles or permissions

When a user inherits a role or permission via RBAC hierarchy instead of directly assigning them in a policy rule, we call such type of assignment as implicit. To query such implicit relations, you need to use these 2 APIs: GetImplicitRolesForUser() and GetImplicitPermissionsForUser instead of GetRolesForUser() and GetPermissionsForUser. For more details, please see [this GitHub issue](#).

2. Use BatchEnforce()

BatchEnforce enforces each request and returns result in a bool array

For example:

Go

```
boolArray, err := e.BatchEnforce(requests)
```

高级用法

多线程

多线程

如果您以多线程的方式使用Casbin，您可以使用Casbin enforcer的同步包：https://github.com/casbin/casbin/blob/master/enforcer_synced.go.

它还支持 AutoLoad特性，这意味着如果最新的策略规则发生了更改，Casbin enforcer将自动从DB加载这些规则。调用 StartAutoLoadPolicy()启动定期自动加载策略，调用StopAutoLoadPolicy()停止策略。

基准测试

基准测试

策略执行的负载在model_b_test.go中进行基准测试。测试是：

```
英特尔 酷睿 i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 核, 8 处理器
```

go test -bench=的基准测试结果。-benchmem 如下 (op = an Enforce() call, ms = millisecond, KB = kilo bytes):

测试用例	规则大小	时间开销 (ms/op)	内存开销 (KB)
ACL	2 规则 (2用户)	0.015493	5.649
RBAC	5条规则 (2用户, 1个角色)	0.021738	7.522
RBAC (小型)	1100条规则 (1000用户, 100个角色)	0.164309	80.620
RBAC (中型)	11000条规则 (10000用户, 1000个角色)	2.258262	765.152
RBAC (大型)	110000条规则 (100000用户, 10000个角色)	23.916776	7606
具有资源角色的RBAC	6条规则 (2用户, 2个角色)	0.021146	7.906
带有域/租户的RBAC	6 条规则 (2个用户, 1个角色, 2个域)	0.032696	10.755
ABAC	0 规则 (0用户)	0.007510	2.328
RESTful	5 规则 (3用户)	0.045398	91.774
拒绝改写	6条规则 (2用户, 1个角色)	0.023281	8.370
优先级	9条规则 (2用户, 2个角色)	0.016389	5.313

性能优化

性能优化

当应用于数以百万计的用户或权限的生产环境时，您可能会在Casbin的强制执行中遇到性能降级，通常有两个原因：

高访问量

每秒到来的请求数量非常庞大，例如：单个Casbin实例每秒就能收到10000条请求。在这种情况下，仅靠一个Casbin实例通常难以处理完所有请求。现在有两种解决方案：

运用多线程来运行多个Casbin实例，这样以来您就可以充分利用机器中的所有内核。详情请参阅：[多线程](#)

将Casbin实例部署到机器集群(多台机器)。使用Watcher来确保所有Casbin实例运行一致。详情请参阅：[Watcher](#)。

NOTE 您可以同时使用上述方法，例如将Casbin部署到10台计算机的群集中。每台机器同时具有5个线程来满足Casbin强制执行请求。

大量的策略规则

在云或多租户环境中，可能需要数百万条策略规则。每次执行请求甚至是在最初期加载策略规则的速度非常缓慢。这类事件通常可以通过以下几种方式缓解：

- 您的Casbin模型或规则设计得不够好。精致的模型和策略将抽象每个用户/租户的重复逻辑，并将规则的数量减少到一个非常小的级别(< 100)：例如：您可以在所有租户之间分享一些默认规则，让用户稍后自定义他们的规则。自定义规则可以覆盖默认规则。如果您仍然有疑问，请将GitHub issue发送到Casbin报告。
- 通过共享让Casbin的执行者只需要加载一套小套策略规则，例如：enforcer_0 只为tenant_0 到tenant_99提供服务，enforcer_1 只为tenant_100到tenant_199提供服务。只需要加载所有策略规则的一部分，详情请参阅：[策略子集加载](#)。
- 以授予RBAC角色权限，取代直接授予用户权限。Casbin的RBAC是通过角色继承树来实现的(作为缓存)。因此授予类似Alice这样的用户权限，Casbin只使用O(1) 时间查询RBAC树来获取角色用户关系并执行操作。如果您的g 规则不会经常改变，那么RBAC 树将不需要进行更新。详情请参阅这条讨论：
<https://github.com/casbin/casbin/issues/681#issuecomment-763801583>

NOTE 您可以同时尝试以上所有方法。 :::

Authorization of Kubernetes

Authorization of Kubernetes

[K8s-authz](#) is a Kubernetes (k8s) RBAC & ABAC authorization middleware based on Casbin. This middleware uses K8s validation admission webhook to check the policies defined by casbin, for every request related to the pods. These custom admission controllers perform some kind of validation on the request object that was forwarded by api server and based on a logic, sends back a response to api server that contains information on whether to allow or reject the request. These controllers are registered with Kubernetes using the ValidatingAdmissionWebhook.

The K8s API server needs to know when to send the incoming request to our admission controller. For this part, we have defined a validation webhook which would proxy the requests for the pods and perform policy verification on it. The user would be allowed to perform the operations on the pods, only if the casbin enforcer authorizes it. The enforcer checks the roles of the user defined in the policies. This middleware would be deployed on the K8s cluster.

Requirements

Before proceeding, make sure to have the following-

- A running k8s Cluster. You can either run the clusters through Docker by enabling it on the Docker Desktop or you can setup the complete K8s ecosystem locally or on your server. You can follow this detailed [guide](#) to setup the k8s cluster locally on Windows or this [guide](#) if want to setup for Linux.
- Kubectl CLI This is the [guide](#) to setup it on Windows and this [guide](#) for Linux.
- OpenSSL

Usage

- Generate the certificates and keys for every user by using openssl and running make certs or the following script:-

```
./gen_cert.sh
```

- Build the docker image from the [Dockerfile](#) manually by running the following command and then change the build version here and at the deployment [file](#), as per the builds.

```
docker build -t casbin/k8s_authz:0.1 .
```

- Define the casbin policies in the [model.conf](#) and [policy.csv](#). You can refer the docs to get to know more about the working of these policies.
- Before deploying, you can change the ports in [main.go](#) and also in the validation webhook configuration [file](#) depending on your usage.
- Deploy the validation controller and the webhook on k8s cluster by running:-

```
kubectl apply -f deployment.yaml
```

Authorization of Kubernetes

Now the server should be running and ready to validate the requests for the operations on the pods.

In case of any query, you can ask on our gitter [channel](#).

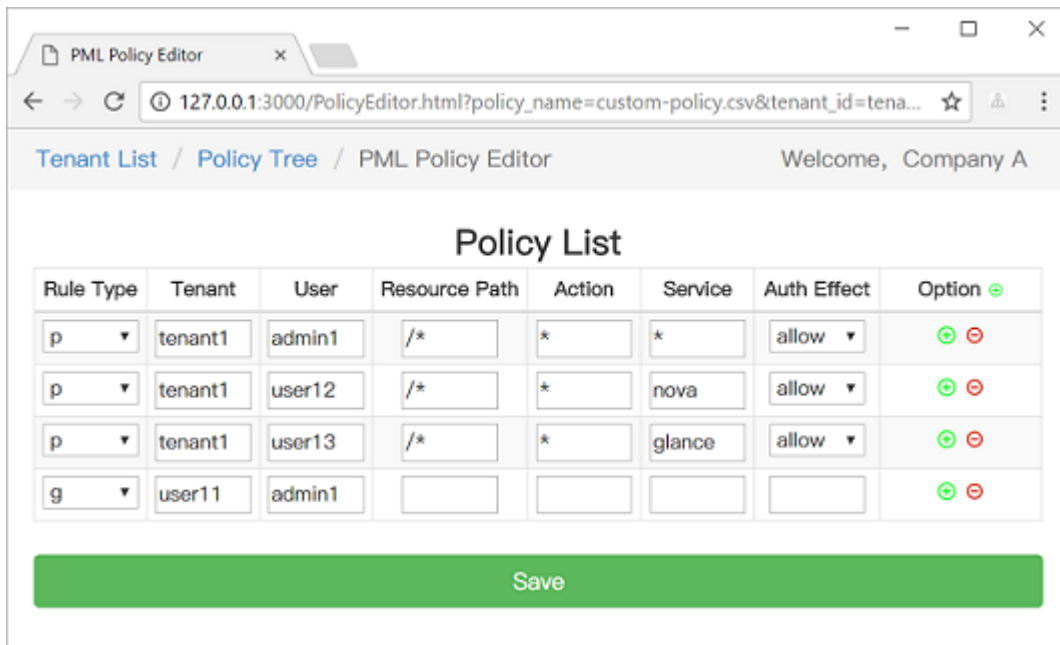
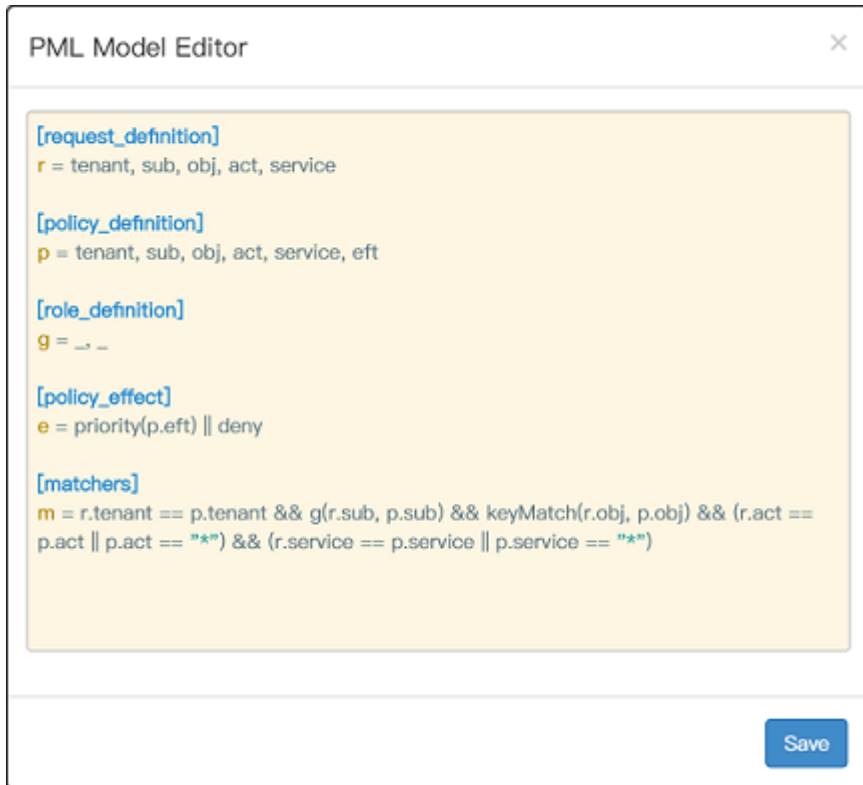
管理

管理

管理员门户

管理员门户

我们提供了一个用于模型管理和政策管理的门户网站 [Casdoor](#)



还有第三方门户管理项目，使用Casbin作为授权引擎。您可以基于这些项目建立您自己的 Cabin 服务。

Go

项目	作者	前端	后端	说明
----	----	----	----	----

项目 项目	作者 作者	前端 前端	后端 后端	说明 说明
----------	----------	----------	----------	----------

Casdoor	Casbin	React + Ant Design	Beego	基于 Beego + XORM + React
go-admin- team/go- admin	@go-admin- team	Vue + Element UI	Gin	go-admin基于Gin + Casbin + GORM
gin-vue- admin	@piexlmax	Vue + Element UI	Gin	基于Gin + GORM + Vue
gin-admin	@LyricTian	React + Ant Design	Gin	基于 Gin + GORM + Casbin + Ant Design React 的RBAC框架
go-admin	@hequan2017	无	Gin	Go RESTful API网关基于Gin + GORM + JWT + RBAC (Casbin)
zeus-admin	bullteam	Vue + Element UI	Gin	基于JWT + Casbin的统一权限管理平台
IrisAdminApi	@snowlyg	Vue + Element UI	Iris	基于 Iris + Casbin 的 后端 API
Gfast	@tiger1103	Vue + Element UI	Go Frame	基于 GF (Go Frame) 的管理门户网站
Echo-Admin	@RealLiuSha	Vue 2.x + Element UI	Echo	基于 Echo + Gorm + Cassbin + Uber-FX 的管理门户网站

Casbin 服务

Casbin 服务

如何使用Casbin作为服务？

名称、	描述
Casbin服务	基于gRPC的官方Casbin as a Service，提供了Management API 和 RBAC API。
PaySuper Casbin Server	PaySuper's fork of the above official Casbin-Server but more actively maintained. 它为 Casbin 授权提供 go-micro 接口。
middleware-acl	RESTful access control middleware based on Casbin.

日志 & 错误处理

日志 & 错误处理

日志

Casbin默认使用内置的 `log` 来将日志输出到控制台，如：

```
2017/07/15 19:43:56 [Request: alice, data1, read ---> true]
```

日志记录不是默认启用的，您可以通过调用 `Enforcer.EnableLog()` 或 `NewEnforcer()` 函数中的最后一个参数来切换它。

NOTE 我们已经支持日志模型、强制请求、角色、Golang策略。您可以定义您自己的日志来记录Casbin。如果您正在使用Python，`pycasbin` 会影响默认的 Python 日志机制。`pycasbin` 软件包调用 `logging.getLogger()` 来设置日志。除了初始化父应用程序中的日志记录器外，不需要特殊配置的日志。如果父应用程序内没有输入日志，您将不会看到来自 `pycasbin` 的日志消息。

对不同的执行器使用不同的记录器

每个执行器都可以有自己的记录器来记录信息，并且可以在运行时进行更改。

而且你可以通过 `NewEnforcer()` 的最后一个参数使用一个适当的日志。如果您使用这种方式来初始化您的执行器，由于日志中启用的字段的优先级更高，您不需要使用启用参数。

```
// 设置默认记录器作为执行器e1的记录器。
//此操作也可以被视为在运行时更改e1的记录器。
e1.SetLogger(&Log.DefaultLogger{})

// 设置另一个记录器作为执行器e2的日志记录器。
e2.SetLogger(&YouOwnLogger)

// 初始化执行器e3时设置您的记录器。
e3, _ := casbin.NewEnforcer("examples/rbac_model.conf", a, logger)
```

支持的记录器

我们提供了一些记录器来帮助您记录信息。

记录器	作者	说明
Default logger (内置)	Casbin	默认使用golang日志。
Zap logger	Casbin	使用 <code>zap</code> ，提供json 编码日志，您可以使用自己的 <code>zap-logger</code> 自定义更多信息。

如何编写一个记录器

您的记录器应该实现 `Logger` 接口。

方法	类型	说明
EnableLog()	强制的	控制是否打印消息。
IsEnabled()	强制的	显示当前日志启用的状态。
LogModel()	强制的	与模型相关的日志信息。
LogEnforce()	强制的	与执行器相关的日志信息。
LogRole()	强制的	与角色相关的日志信息。
LogPolicy()	强制的	与策略相关的日志信息。

您可以将您的自定义 logger 传给 `Enforcer.SetLogger()` 函数。

这是一个关于如何自定义Golang记录器的示例：

```
import (
    "fmt"
    "log"
    "strings"
)

// 默认日志是使用golang日志的日志实现的。
type DefaultLogger struct {
    enabled bool
}

func (l *DefaultLogger) EnableLog(enable bool) {
    l.enabled = enable
}

func (l *DefaultLogger) IsEnabled() bool {
    return l.enabled
}

func (l *DefaultLogger) LogModel(model [][]string) {
    if !l.enabled {
        return
    }
    var str strings.Builder
    str.WriteString("Model: ")
    for _, v := range model {
        str.WriteString(fmt.Sprintf("%v\n", v))
    }
    log.Println(str.String())
}

func (l *DefaultLogger) LogEnforce(matcher string, request []interface{}, result bool, explains [][]string) {
    if !l.enabled {
```

```

return
}

var reqStr strings.Builder
reqStr.WriteString("Request: ")
for i, rval := range request {
    if i != len(request)-1 {
        reqStr.WriteString(fmt.Sprintf("%v, ", rval))
    } else {
        reqStr.WriteString(fmt.Sprintf("%v", rval))
    }
}
reqStr.WriteString(fmt.Sprintf(" --> %t\n", result))

reqStr.WriteString("Hit Policy: ")
for i, pval := range explains {
    if i != len(explains)-1 {
        reqStr.WriteString(fmt.Sprintf("%v, ", pval))
    } else {
        reqStr.WriteString(fmt.Sprintf("%v \n", pval))
    }
}

log.Println(reqStr.String())
}

func (l *DefaultLogger) LogPolicy(policy map[string][]string) {
    if !l.enabled {
        return
    }

    var str strings.Builder
    str.WriteString("Policy: ")
    for k, v := range policy {
        str.WriteString(fmt.Sprintf("%s : %v\n", k, v))
    }

    log.Println(str.String())
}

func (l *DefaultLogger) LogRole(roles []string) {
    if !l.enabled {
        return
    }

    log.Println("Roles: ", roles)
}

```

错误处理

当您使用Casbin时可能会由于以下原因发生错误:

1. Model文件 (. conf) 中的语法有误。
2. policy文件 (. csv) 中的语法有误。
3. 来adapter的自定义错误信息 (譬如连接MySQL失败)。
4. Casbin的bug。

您需要注意以下五个主要函数出现的error:

函数	异常时行为
<code>NewEnforcer()</code>	返回error
<code>LoadModel()</code>	返回error
<code>LoadPolicy()</code>	返回error
<code>SavePolicy()</code>	返回error
<code>Enforce()</code>	返回error

`NEWENFORCER()` 通过内部调用 `LOADMODEL()` 和 `LOADPOLICY()` 。所以当您使用 `NEWENFORCER()` 函数时不需要再去调用这两个函数。

启用 & 禁用

通过调用`Enforcer.EnableEnforce()`方法，我们可以禁用`Enforcer`。当它被禁用时，`Enforcer.Enforce()` 将总是返回 `true`。但是其他操作（例如添加或删除`policy`）不受影响。以下为一些示例：

```
e := casbin.NewEnforcer("examples/basic_model.conf", "examples/basic_policy.csv")

// 将返回假。
// 默认情况下执行器是启用的。
e.Enforce("non-authorized-user", "data1", "read")

// 在运行时禁用enforcer
e.EnableEnforce(false)

// 对任何请求都返回true
e.Enforce("non-authorized-user", "data1", "read")

// 再次启用执行器。
e.EnableEnforce(true)

// 将返回假。
e.Enforce("non-authorized-user", "data1", "read")
```

前端使用

前端使用

[Casbin.js](#)是一个能够帮助你在前端应用中管理访问控制权限的Casbin前端版本。

安装

```
npm install casbin.js  
npm install casbin
```

或者

```
yarn add casbin.js
```

快速开始

您可以在您的前端应用程序中使用 **manual** 模式，并随时设置权限。

```
const casbinjs = require("casbin.js");  
// 设置用户权限  
// 他/她可以可以读取data1和data2并且可以写入data1  
const permission = {  
  "read": ["data1", "data2"],  
  "write": ["data1"]  
};  
  
// 在manual模式使用Casbin.js需要您手动设置权限  
const authorizer = new casbinjs.Authorizer("manual");
```

现在我们有了一个授权者 **authorizer**。我们可以通过使用API **authorizer.can()**和**authorizer.cannot()**获得得许可规则。这2个API的返回值是JavaScript Promise (详细信息) 所以我们应该使用 **then()** 返回值的方法，例如：

```
result = authorizer.can("write", "data1");  
result.then((success, failed) => {  
  if (success) {  
    console.log("you can write data1");  
  } else {  
    console.log("you cannot write data1");  
  }  
});  
// 输出：您可以写入data1
```

cannot() 以同样方式使用：

```
result = authorizer.cannot("read", "data2");  
result.then((success, failed) => {  
  if (success) {  
    console.log("you cannot read data2");  
  } else {  
    console.log("you can read data2");  
  }  
});
```

```
});
// 输出：您可以读取data2
```

在上面的代码中，变量 `success` 意味着请求获得结果而不产生错误， 而不意味着权限规则是 `true failed` 也和权限规则无关 只有在请求过程中出现错误时才有意义。

您可以参考我们的[React示例](#)来查看Casbin.js的实际用法。

高级用法

Casbin.js提供了一个完美的解决方案来将您的前端访问控制管理和后端Casbin服务一体化。

在初始化 Casbin.js Authorizer时使用 `auto` 模式并指定你的后端地址， 它会自动同步权限并调整前端状态。

```
const casbinjs = require('casbin.js');

// 设置你的后端Casbin服务url
const authorizer = new casbinjs.Authorizer(
  'auto', // 模式
  {endpoint: 'http://your_endpoint/api/casbin'}
);

// 设置你的访客。
// Casbin.js 会自动与你的后端Casbin服务同步权限。
authorizer.setUser("Tom");

// 评估权限
result = authorizer.can("read", "data");
result.then((success, failed) => {
  if (success) {
    // 一些前端操作
  }
});
```

因此，您需要开放一个接口(例如一个 RestAPI)来创建权限对象并将其返回到前端。 在你的 API 控制器中，调用 `CasbinJsGetUserPermission` 以创建权限对象。 下面是一个 Beego 框架的示例：

您的端点服务器应该返回类似的内容

```
{
  "other": "other",
  "data": "What you get from `CasbinJsGetPermissionForUser`"
}
```

```
// 路由器
beego.Router("api/casbin", &controllers.APIController{}, "GET:GetFrontendPermission")

// 控制器
func (c *APIController) GetFrontendPermission() {
  // 在 GET 请求的参数中获取访客。(其中的键是"casbin_subject")
  visitor := c.Input().Get("casbin_subject")
  // `e` 是一个初始化的Casbin Enforcer实例
  c.Data["perm"] = casbin.CasbinJsGetPermissionForUser(e, visitor)
  // 将数据传到前端
  c.ServeJSON()
}
```

NOTE 目前 CasbinJsGetPermissionForUser api只支持Go Casbin 和 Node-Casbin。如果您希望这个api支持其它语言，请在此提交issue 或者留下评论。

API 列表

- `setPermission(permission: string)`
 - 设置权限对象。始终在 `manual` 模式中使用。
- `setUser(user: string)`
 - 设置访客身份并更新权限。始终在 `auto`模式中使用。
- `can(action: string, object: string)`
 - 检查用户是否能对 `object` 执行 `action`。
- `cannot(action: string, object: string)`
 - 检查用户是否不能对 `object` 执行 `action`。
- `canAll(action: string, objects: Array)`
 - 检查用户是否能在`objects`对所有对象执行`action`。
- `canAny(action: string, objects: Array)`
 - 检查用户是否能对 `objects` 中的任意一个执行 `action`。

为什么选择 Casbin.js

人们可能会想知道Node-Casbin和Casbin.js之间的区别。总的来说，Node-Casbin 是一个用 NodeJS 环境实现的 Casbin 核心，它通常在服务端用作一个访问控制工具包。Casbin.js 是一个能帮助您在客户端使用Casbin为你网页里的用户授权的前端库。

通常，由于以下问题，直接构建一个 Casbin 服务并在网页前端执行授权/执行是不妥当的：

- i. 当有人启动客户端时，执行器会被初始化，随后在后端持久层中拉取所有策略。高并发会为数据库带来巨大的压力并带来极高的网络成本。
- ii. 将所有策略悉数加载进客户端会带来安全风险。
- iii. 区分客户端和服务端以及灵活的开发有困难。

我们希望有一种能够简化 Casbin 前端开发的工具。实际上，Casbin.js 的核心是在客户端操纵当前用户的权限。正如你提到的，Casbin.js 从一个指定的后端获取数据。这个程序会与 Casbin 后端服务同步用户的权限。取得权限数据之后，开发者可以使用 Casbin.js 提供的接口来在前端管理用户行为。

Casbin.js 避免了以上提及的两个问题：Casbin 服务将不再被反复请求数据，并且客户端与服务端之间传递数据量大大减少 我们也避免了将所有的策略都储存在前端的问题。用户仅能操作与之相关的权限，对其它诸如访问控制模型、其他用户的权限的内容将一无所知。此外，Casbin.js 还能有效地在授权管理方面解耦客户端和服务端。

编辑器

编辑器

在线编辑器

在线编辑器

你也可以使用在线编辑器(<https://casbin.org/en/editor/>) 在你的浏览器中编辑你的Casbin模型和策略。它提供了一些比如语法高亮 以及代码补全这样的功能，就像编程语言的IDE一样。

IDE 插件

IDE 插件

我们有这些IDE插件:

JetBrains

- 下载: <https://plugins.jetbrains.com/plugin/14809-casbin>
- 源代码: <https://github.com/will7200/casbin-idea-plugin>

VSCode (正在制作)

- 源代码: <https://github.com/casbin/casbin-vscode-plugin>