



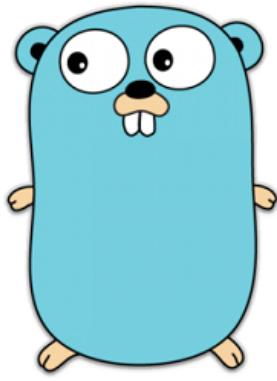
概述

Casbin 是一个强大和高效的开放源码访问控制库，它支持各种 [访问控制模型](#) 以强制全面执行授权。

Enforcing a set of rules is as simple as listing subjects, objects, and the desired allowed action (or any other format as per your needs) in a *policy* file. This is synonymous across all flows in which Casbin is used. The developer/administrator has complete control over the layout, execution, and conditions for authorization, which are set via the *model* file. Casbin provides an *Enforcer* for validating an incoming request based on the policy and model files given to the Enforcer.

Languages Supported by Casbin

Casbin provides support for various programming languages, ready to be integrated within any project and workflow:

 Go	 Java
Casbin	jCasbin
Production-ready	Production-ready



PyCasbin	Casbin.NET
Production-ready	Production-ready

Feature Set for Different Languages

We are always working our best to make Casbin have the same set of features for all languages. However, the reality is not that beautiful.

特性	Go	Java	Node.js	PHP	Python	C#	Delphi	Rust	C++	Lua	Dart	Elixir
具体实施	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBAC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ABAC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scaling ABAC (eval())	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✓
适配器	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
管理接口	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBAC API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Batch API	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗

特性	Go	Java	Node.js	PHP	Python	C#	Delphi	Rust	C++	Lua	Dart	Elixir
Filtered Adapter	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗
Watcher	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗
Role Manager	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓	✗
Multi-Threading	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗	✗	✗
matcher 中的'in'语法	✓	✓	✓	✓	✓	✗	✓	✗	✗	✗	✓	✓

Note - ✓ for Watcher or Role Manager only means having the interface in the core library. It is not indicative of whether there is a watcher or role manager implementation available.

Casbin 是什么？

Casbin is an authorization library that can be used in flows where we want a certain `object` or entity to be accessed by a specific user or `subject`. The type of access, i.e. `action`, can be `read`, `write`, `delete`, or any other action as set by the developer. This is how Casbin is most widely used, and it's called the "standard" or classic `{ subject, object, action }` flow.

Casbin能够处理除标准流量以外的许多复杂的许可使用者。 There can be the addition of [roles \(RBAC\)](#), [attributes \(ABAC\)](#), etc.

What Casbin Does

1. Enforce the policy in the classic `{ subject, object, action }` form or a customized form as you defined. 支持允许和拒绝授权。
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. 支持内置超级用户，如 `root` 或 `administrator`。 超级用户可以在没有明确权限的情况下做任何事情。
5. Provide multiple built-in operators to support rule matching. For example, `keyMatch` can map a resource key `/foo/bar` to the pattern `/foo*`.

What Casbin Does NOT Do

1. Authentication (aka verifying `username` and `password` when a user logs in)
2. 管理用户列表或角色列表。

It's more convenient for projects to manage their lists of users, roles, or passwords. 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

安装

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [C++](#) [Rust](#)
[Delphi](#) [Lua](#)

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

对于Maven:

```
<!-- https://mvnrepository.com/artifact/org.casbin/jcasbin -->
```

```
<dependency>
    <groupId>org.casbin</groupId>
    <artifactId>jcasbin</artifactId>
    <version>1.x.y</version>
</dependency>
```

```
# NPM
npm install casbin --save
```

```
# Yarn
yarn add casbin
```

Require this package in the `composer.json` of your project to download the package:

```
composer require casbin/casbin

pip install casbin

dotnet add package Casbin.NET

# Download source
git clone https://github.com/casbin/casbin-cpp.git

# Generate project files
cd casbin-cpp && mkdir build && cd build && cmake ..
-DCMAKE_BUILD_TYPE=Release

# Build and install casbin
cmake --build . --config Release --target casbin install -j 10

cargo install cargo-edit
cargo add casbin

// If you use async-std as async executor
cargo add async-std

// If you use tokio as async executor, make sure you activate
its `macros` feature
cargo add tokio
```

Casbins4D 以包的形式提供（目前为 Delphi 10.3 Rio），您可以在 IDE 中安装它。然而，没有可视化组件，意味着你可以在包外独立使用这些 unit。只需在你的项目中导入 unit 即可（如果你不介意它们的数量）。

```
luarocks install casbin
```

If you receive an error message: "Your user does not have write permissions in

/usr/local/lib/luarocks.rocks", you may want to run the command as a privileged user or use your local tree with `--local`. To fix the error, you can add `--local` behind your command like this:

```
luarocks install casbin --local
```

新建一个Casbin enforcer

Casbin uses configuration files to define the access control model.

There are two configuration files: `model.conf` and `policy.csv`. `model.conf` stores the access model, while `policy.csv` stores the specific user permission configuration. The usage of Casbin is very straightforward. We only need to create one main structure: `enforcer`. 当构造这个结构的时候, `model.conf` 和 `policy.csv` 将会被加载。

In other words, to create a Casbin enforcer, you need to provide a [Model](#) and an [Adapter](#).

Casbin provides a [FileAdapter](#) that you can use. See [Adapter](#) for more information.

- Example of using the Model file and the default [FileAdapter](#):

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [C++](#) [Delphi](#)

[Rust](#) [Lua](#)

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

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

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

import { newEnforcer } from 'casbin';

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

require_once './vendor/autoload.php';

use Casbin\Enforcer;

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

import casbin

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

using NetCasbin;

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

#include <iostream>
#include <casbin/casbin.h>

int main() {
    // 创建一个执行者
    casbin::Enforcer e("path/to/model.conf", "path/to/
policy.csv");
```

```

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

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(())
}

local Enforcer = require("casbin")
local e = Enforcer:new("path/to/model.conf", "path/to/
policy.csv") -- The Casbin Enforcer

```

- 与其他Adapter一起使用Model text

Go Python

```
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)/")
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)
}
```

```
import casbin
import casbin_sqlalchemy_adapter

# 将SQLAlchemy Casbin适配器与SQLite DB一起使用
adapter = casbin_sqlalchemy_adapter.Adapter('sqlite:///test.db')

# 创建配置模型策略
with open("rbac_example_model.conf", "w") as f:
    f.write("""
[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
""")

# 从适配器和配置策略创建执行器
e = casbin.Enforcer('rbac_example_model.conf', adapter)
```

检查权限

在访问发生之前，在代码中添加强制挂钩：

Go Java Node.js PHP Python .NET C++ Delphi

Rust Lua

```
sub := "alice" // 想要访问资源的用户。
obj := "data1" // 将被访问的资源。
act := "read" // 用户对资源执行的操作。

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"}})
```

```
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 {
```

```
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
}

$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
}

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

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
```

```
casbin::Enforcer e("../assets/model.conf", "../assets/
policy.csv");

if (e.Enforce({"alice", "/alice_data/Hello", "GET})) {
    std::cout << "Enforce OK" << std::endl;
} else {
    std::cout << "Enforce NOT Good" << std::endl;
}

if (e.Enforce({"alice", "/alice_data/Hello", "POST})) {
    std::cout << "Enforce OK" << std::endl;
} else {
    std::cout << "Enforce NOT Good" << std::endl;
}

if casbin.enforce(['alice,data1,read']) then
    // Alice很高兴它能够读取data1了
else
    // Alice很伤心

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
}

if e:enforce("alice", "data1", "read") then
    -- 允许alice读取data1
else
    -- 拒绝请求，抛出异常
```

Casbin还提供了在运行时进行权限管理的API。例如，你可以获得分配给一个用户的所有角色，如下所示：

Go Java Node.js PHP Python .NET Delphi Rust

Lua

```
roles, err := e.GetRolesForUser("alice")  
  
List<String> roles = e.getRolesForUser("alice");  
  
const roles = await e.getRolesForUser('alice');  
  
$roles = $e->getRolesForUser("alice");  
  
roles = e.get_roles_for_user("alice")  
  
var roles = e.GetRolesForUser("alice");  
  
roles = e.rolesForEntity("alice")  
  
let roles = e.get_roles_for_user("alice");  
  
local roles = e:GetRolesForUser("alice")
```

更多使用方法见 [Management API](#)和 [RBAC API](#)。

请查看测试用例以获取更多使用方式。

How It Works

在 Casbin 中, 访问控制模型被抽象为基于 PERM (Policy, Effect, Request, Matcher) 的一个配置文件。Switching or upgrading the authorization mechanism for a project is as simple as modifying a configuration. 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, 并共享一组策略规则。

The PERM model is composed of four foundations: Policy, Effect, Request, and Matchers. These foundations describe the relationship between resources and users.

请求

Defines the request parameters. A basic request is a tuple object, requiring at least a subject (accessed entity), object (accessed resource), and action (access method).

例如, 一个请求可能长这样: `r={sub, obj, act}`

This definition specifies the parameter names and ordering required by the access control matching function.

策略

Defines the model for the access strategy. It specifies the name and order of the fields in the Policy rule document.

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

Note: If eft (policy result) is not defined, the result field in the policy file will not be read, and the matching policy result will be allowed by default.

匹配器

Defines the matching rules for Request and Policy.

For example: `m = r.sub == p.sub && r.act == p.act && r.obj == p.obj`

This simple and common matching rule means that if the requested parameters (entities, resources, and methods) are equal to those found in the policy, then the policy result (`p.eft`) is returned. 策略的结果将保存在 `p.eft` 中。

效果

Performs a logical combination judgment on the matching results of Matchers.

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

This statement means that if the matching strategy result `p.eft` has the result of (some) allow, then the final result is true.

Let's look at another example:

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

The logical meaning of this example combination is: if there is a strategy that matches the result of allow and no strategy that matches the result of deny, the result is true. In other words, it is true when the matching strategies are all allow. If there is any deny, both are false (more simply, when allow and deny exist at the same time, deny takes precedence).

Casbin最基本和最简单的模式是ACL。 The model CONF for ACL is as follows:

```

# 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

```

An example policy for the ACL model is:

```

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

```

This means:

- alice可以读取data1
- bob可以编写data2

We also support multi-line mode by appending '\' in the end:

```

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

```

Furthermore, if you are using ABAC, you can try the 'in' operator as shown in the following example for the Casbin golang edition (jCasbin and Node-Casbin are not supported yet):

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

But you **SHOULD** make sure that the length of the array is **MORE** than 1, otherwise it will cause a panic.

For more operators, you may take a look at [govaluate](#).

使用指南

阅读前, 请注意某些教程适用于Casbin的模型, 适用于所有Casbin的不同语言的实现。其他的一些教程是关于特定语言的。

我们的论文

- PML: An Interpreter-Based Access Control Policy Language for Web Services
(PML: 基于解释器的Web服务访问控制策略语言)

这篇论文深入剖析了Casbin的设计细节。如果您在论文中使用Casbin/PML作为参考, 请引用以下BibTex:

```
@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政策规格语言\(中文\)](#)

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)
- ThinkPHP 5.1 + Casbin:如何使用自定义匹配函数(中文)
- Webman实战教程：如何使用casbin权限控制 (中文)

PERM元模型(策略、效果、请求、匹配器)

- 使用不同的访问控制模型配置来了解Casbin
- 利用Casbin的PERM模型进行访问控制
- 使用 Casbin 设计一个灵活的权限系统
- 授权访问控制列表
- 使用PERM和Casbin的访问控制(波斯语)
- RBAC? ABAC? .. PERM! New Way of Authorization for Cloud-Based Web Services and Apps (in Russian) (基于云的Web服务和应用程序授权的新方式 (俄语))

- 练习 & 使用 Casbin & PERM 的灵活授权实例 (俄语)
- Casbin权限管理 (中文)
- Casbin分析 (中文)
- 系统权限设计 (中文)
- Casbin: 一个权限引擎(中文)
- 使用 Casbin 实现ABAC (中文)
- Casbin 源代码分析(中文)
- Casbin 的权限评估(中文)
- Casbin: Go今日的库(中文)

Go Java Node.js PHP .NET Rust Lua

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)

监视器

- RBAC 通过Casbin Watcher分发同步(中文)

Beego

- 使用 Casbin 与 Beego: 1. 开始测试(中文)
- 使用 Casbin 与 Beego: 2. 策略储存(中文)
- 使用 Casbin 与 Beego: 3. 策略查询 (中文)
- 使用 Casbin 与 Beego: 4. 更新策略(中文)
- 使用 Casbin 与 Beego: 5. 更新策略(续)(中文)

Gin

- 使用 Casbin 的 Golang 项目授权
- 教程: 将 Gin 与 Casbin 集成

- 带Pipeline的 K8s 上的策略执行
- 使用 JWT 和 Casbin 在Gin 应用程序中进行身份验证和授权
- 后端 API 与 Go: 1. 基于 JWT 的身份验证 (中文)
- 后端 API 与 Go: 2. 基于 Casbin 的授权(中文)
- 在Gin和GORM使用Go的授权库Casbin (日语)

Echo

- Casbin 网络授权

Iris

- Iris + Casbin 权限控制实战
- 从头学习 iris + Casbin

VMware Harbor

- Casbin: Golang访问控制框架 (中文)
- Harbor 访问控制 (中文)

Argo CD

- Argo CD 中使用 Casbin 建立 RBAC 权限体系

GShark

- GShark: 轻松有效地扫描Github中的敏感信息

SpringBoot

- jCasbin: 更轻量级的权限管理解决方案 (中文)
- JCasbin与JFinal的集成 (中文)

Express

- 如何将基于角色访问控制添加到您的AWS上的服务器

Koa

- Casbin and Koa授权 Part 1
- Casbin and Koa授权 Part 2

Nest

- 如何使用 Casbin 和 Nest.js 创建基于角色的认证中间件
- nest.js: Casbin RESTful RBAC授权插件（视频）
- 基于 Casbin 的 Node.js 基于属性的访问控制演示应用程序
- 多租户SaaS 启动工具包，带有cqrs graphql microservice 架构

Fastify

- 使用 Fastify 和 Casbin 在 Node.js 中的访问控制
- Casbin, 您的项目中强大和高效的 ACL
- 在 .Net 中使用 Casbin 授权
- 在 Rust 中用 Casbin 实现基础的基于角色的 HTTP 授权
- 如何在您的rust web应用程序中使用casbin 授权 [Part - 1]
- 如何在您的rust web应用程序中使用casbin 授权 [Part - 2]

APISIX

- 使用 Casbin 在 APISX 中授权



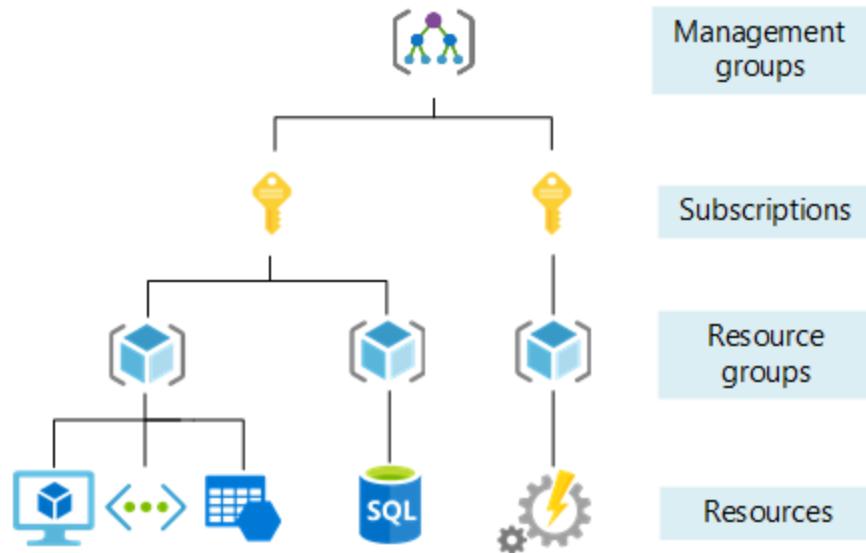
Understanding How Casbin Matching Works in Detail

In this post, I will explain the design and implementation of RBAC using the [Casbin](#) library. For a SaaS platform dealing with multiple resource hierarchies and roles that inherit permissions from higher levels, Casbin provides a performant alternative to consider.

Introduction to RBAC

RBAC is a method of restricting access to resources based on the roles that individuals hold. To better understand how hierarchical RBAC works, let's take a look at Azure's RBAC system in the next section and then attempt to implement a similar system.

Understanding Azure's Hierarchical RBAC



There is a role called **Owner** for all resources in Azure. Suppose if I have the **Owner** role assigned to me at the subscription level, that means I am the **Owner** of all the resource groups and resources under that subscription. If I have **Owner** at the resource group level, then I am the **Owner** of all the resources under that resource group.

This image shows that I have **Owner** access at the subscription level.

Microsoft Azure

Home > Subscriptions > pay-as-you-go

pay-as-you-go | Access control (IAM)

Subscription

Search

Add Download role assignments Edit columns Refresh Remove Feedback

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Security Events Cost Management Cost analysis Cost alerts Budgets Advisor recommendations Billing Invoices External services Payment methods Partner information Settings Programmatic deployment Resource groups Resources Preview features

Number of role assignments for this subscription: 31 / 4000

Type: All Role: All Scope: All scopes Group by: Role

Showing a filtered set of results. Total number of role assignments: 31

Name	Type	Role	Scope	Condition
aravindkumar	User	Owner	This resource	None

When I check the IAM of a Resource Group under this Subscription, you can see that I have inherited Owner access from the subscription.

Microsoft Azure

Home > Resource groups > test-resource-group

test-resource-group | Access control (IAM)

Resource group

Search Add Download role assignments Edit columns Refresh Remove Feedback

Overview Activity log Access control (IAM) Tags Resource visualizer Events

Settings Deployments Security Policies Properties Locks

Cost Management Cost analysis Cost alerts (preview) Budgets Advisor recommendations

Monitoring Insights (preview) Alerts Metrics Diagnostic settings

Number of role assignments for this subscription: 31 / 4000

Type: All Role: All Scope: All scopes Group by: Role

Showing a filtered set of results. Total number of role assignments: 31

Name	Type	Role	Scope	Condition
aravindkumar	User	Owner	Subscription (inherited)	None

So, this is how Azure's RBAC is hierarchical. Most enterprise software uses hierarchical RBAC because of the hierarchical nature of the resource levels. In this

tutorial, we'll try to implement a similar system using Casbin.

How Does Casbin Work?

Before diving into the implementation, it is important to understand what Casbin is and how it functions at a high level. This understanding is necessary because each Role-Based Access Control (RBAC) system may vary based on specific requirements. By grasping the workings of Casbin, we can effectively fine-tune the model.

What is ACL?

ACL stands for Access Control List. It is a method in which users are mapped to actions and actions to resources.

The model definition

Let's consider a simple example of an ACL model.

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

[policy_definition]
p = sub, act, obj

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

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

1. The `request_definition` is the query template of the system. For example, a request `alice, write, data1` can be interpreted as "Can subject Alice perform the action 'write' on object 'data1'?".
2. The `policy_definition` is the assignment template of the system. For example, by creating a policy `alice, write, data1`, you are assigning permission to subject Alice to perform the action 'write' on object 'data1'.
3. The `policy_effect` defines the effect of the policy.
4. In the `matchers` section, the request is matched with the policy using the conditions `r.sub == p.sub && r.obj == p.obj && r.act == p.act`.

Now let's test the model on the Casbin editor

Open the [editor](#) and paste the above model in the Model editor.

Paste the following in the Policy editor:

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

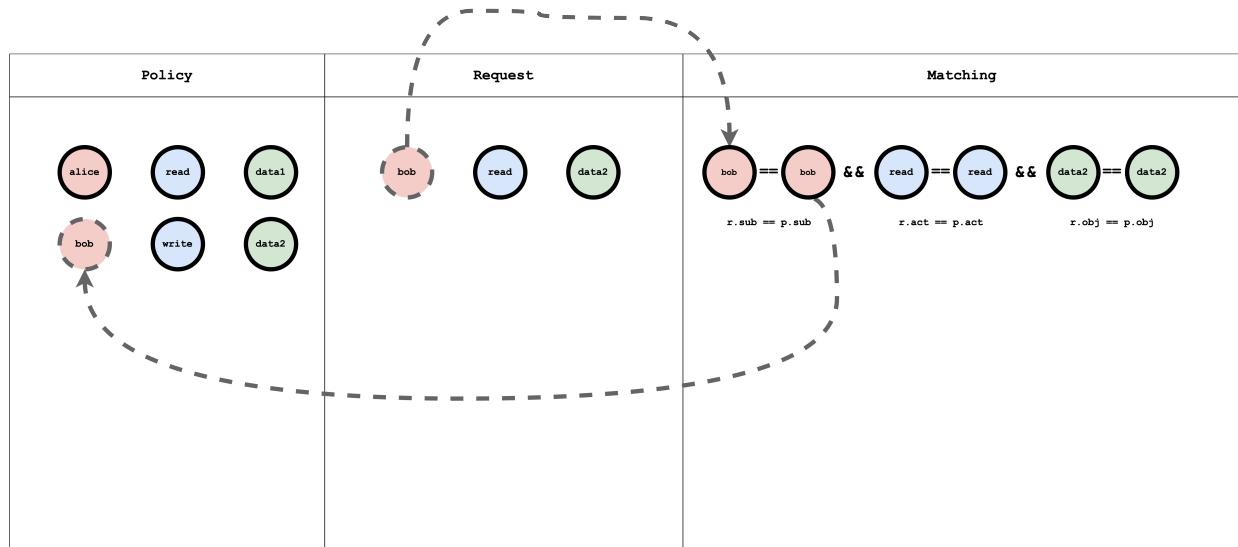
and the following in the Request editor:

```
alice, read, data1
```

The result will be:

```
true
```

Visual representation of the ACL model, policy, and request matching



What is RBAC?

RBAC stands for Role-Based Access Control. In RBAC, a user is assigned a role for a resource, and a role can contain arbitrary actions. The request then checks if the user has the permission to perform the action on the resource.

The model definition

Let's consider a simple example RBAC model:

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

1. The role_definition is a graph relation builder that uses a Graph to compare the request object with the policy object.

Now let's test the model on Casbin editor

Open the [editor](#) and paste the above model in the Model editor.

Paste the following in the Policy editor:

```
p, alice, reader, data1  
p, bob, owner, data2
```

```
g, reader, read  
g, owner, read  
g, owner, write
```

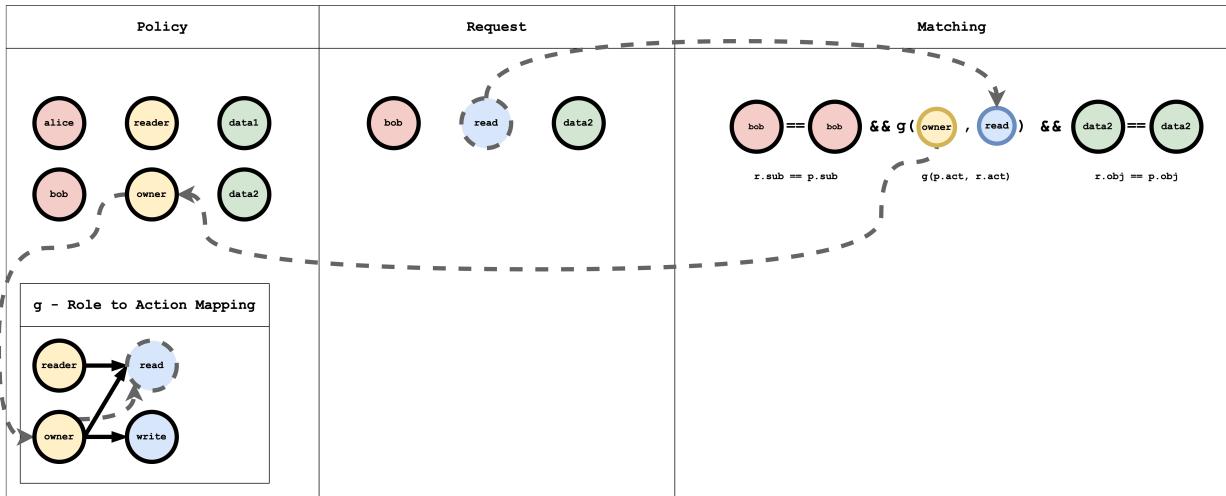
and the following in the Request editor:

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

The result will be:

```
true  
false  
true  
true  
false
```

Visual representation of the RBAC model, policy, and request matching



The g - Role to action mapping table has a Graph mapping the role to action. This Graph can be coded as a list of edges, as shown in the policy which is a common way of representing a Graph:

```
g, reader, read  
g, owner, read  
g, owner, write
```

① 信息

p indicates a normal policy that can be compared using the $==$ operator. g is a Graph-based comparison function. You can define multiple Graph comparators by adding a numerical suffix like g , $g2$, $g3$, ... and so on.

What is Hierarchical RBAC?

In Hierarchical RBAC, there are more than one type of resources and there is an inheritance relationship between the resource types. For example, "subscription" is one type and "resourceGroup" is another type. A sub1 of type Subscription can contain multiple resourceGroups (rg1, rg2) of type ResourceGroup.

Similar to the resource hierarchy, there will be two types of roles and actions: Subscription roles and actions, and ResourceGroup roles and actions. There is an arbitrary relationship between the Subscription role and ResourceGroup role. For example, consider a Subscription Role **sub-owner**. This role is inherited by a ResourceGroup Role **rg-owner**, which means that if I am assigned the **sub-owner** role on Subscription **sub1**, then I automatically also get the **rg-owner** role on **rg1** and **rg2**.

The model definition

Let's take a simple example of the Hierarchical RBAC model:

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

[policy_definition]
p = sub, act, obj

[role_definition]
g = _, _
g2 = _, _

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

1. The role_definition is a graph relation builder which uses a Graph to compare the request object with the policy object.

Now let's test the model on the Casbin editor

Open the [editor](#) and paste the above model in the Model editor.

Paste the following in the Policy editor:

```
p, alice, sub-reader, sub1
p, bob, rg-owner, rg2

// subscription role to subscription action mapping
g, sub-reader, sub-read
g, sub-owner, sub-read
g, sub-owner, sub-write

// resourceGroup role to resourceGroup action mapping
g, rg-reader, rg-read
g, rg-owner, rg-read
g, rg-owner, rg-write

// subscription role to resourceGroup role mapping
g, sub-reader, rg-reader
g, sub-owner, rg-owner

// subscription resource to resourceGroup resource mapping
g2, sub1, rg1
g2, sub2, rg2
```

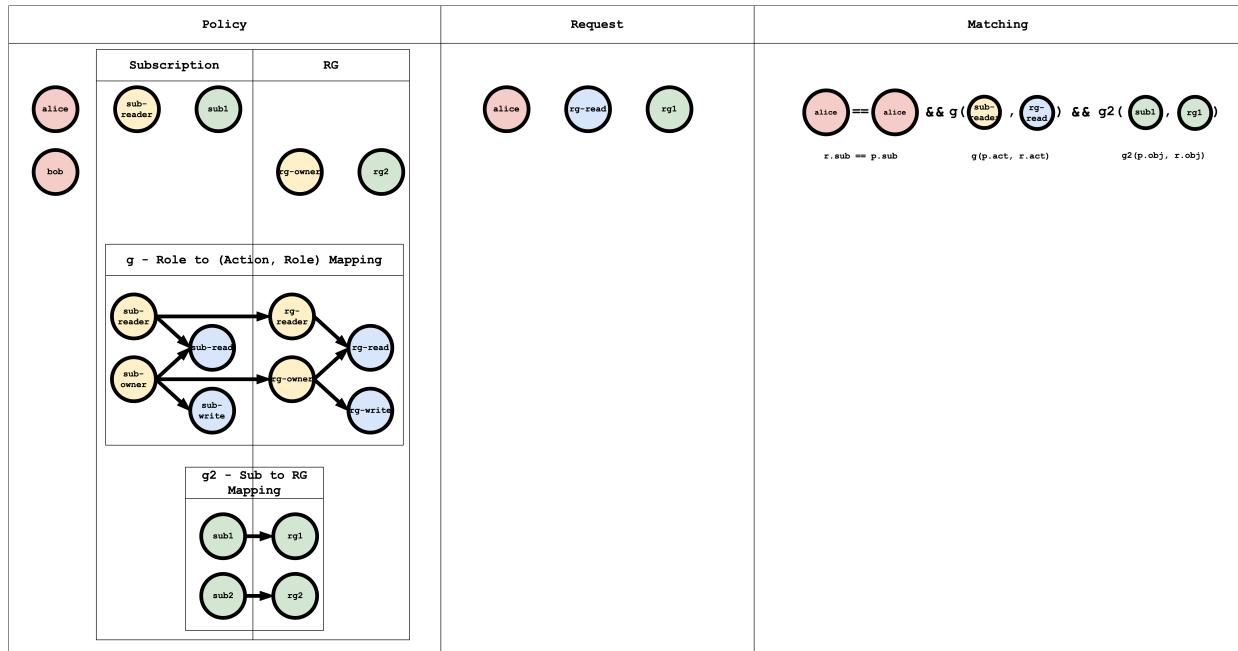
And paste the following in the Request editor:

```
alice, rg-read, rg1
```

The result will be:

true

Visual representation of the RBAC model, policy, and request matching



The **g - Role to (Action, Role) Mapping** table has a graph mapping the role to the action, role mapping. This graph can be coded as a list of edges, as shown in the policy, which is a common way of representing a graph:

```

// subscription role to subscription action mapping
g, sub-reader, sub-read
g, sub-owner, sub-read
g, sub-owner, sub-write

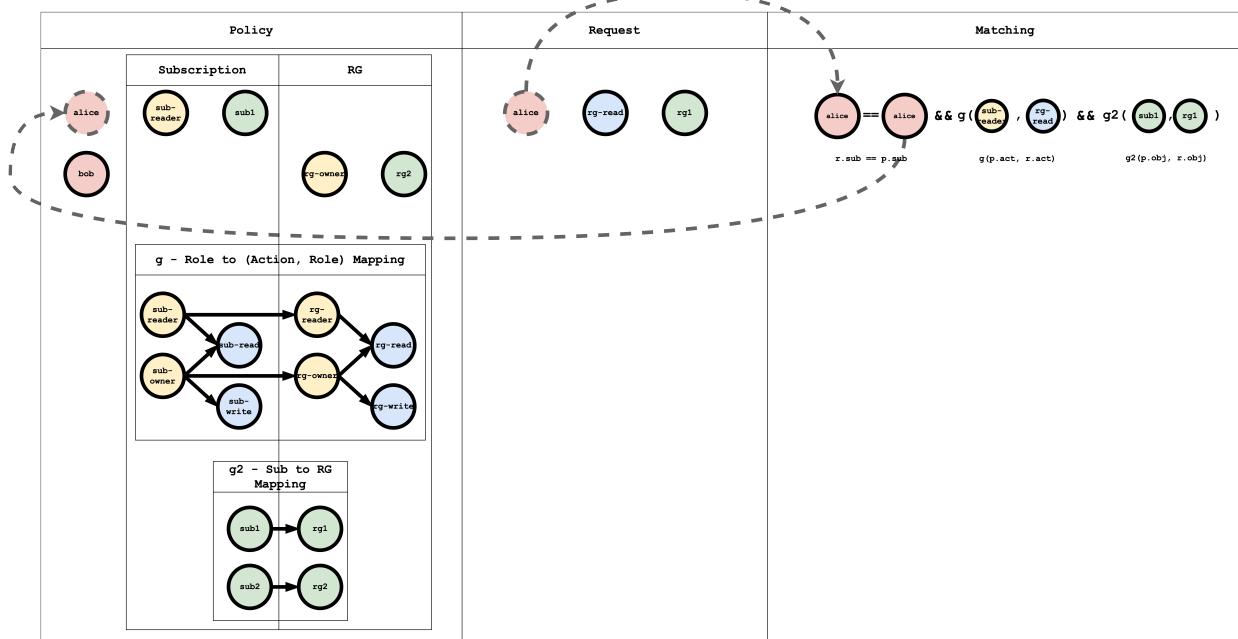
// resourceGroup role to resourceGroup action mapping

```

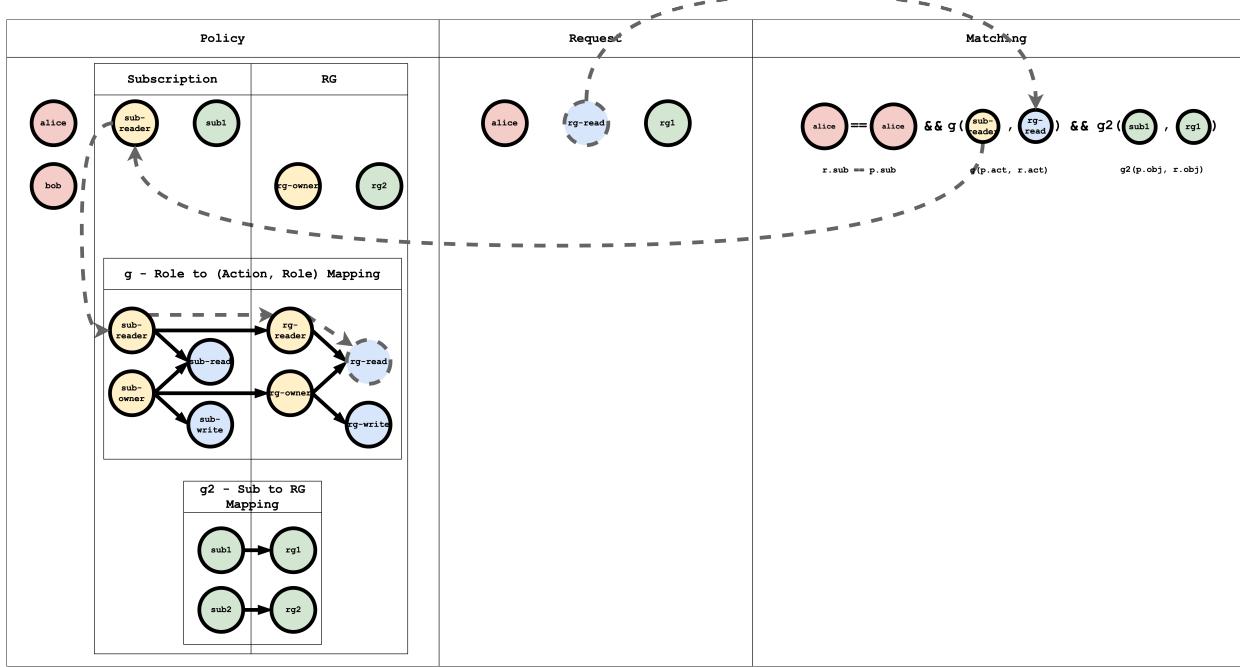
The g2 - Sub to RG Mapping table has a graph mapping subscription to resourceGroup:

```
// subscription resource to resourceGroup resource mapping
g2, sub1, rg1
g2, sub2, rg2
```

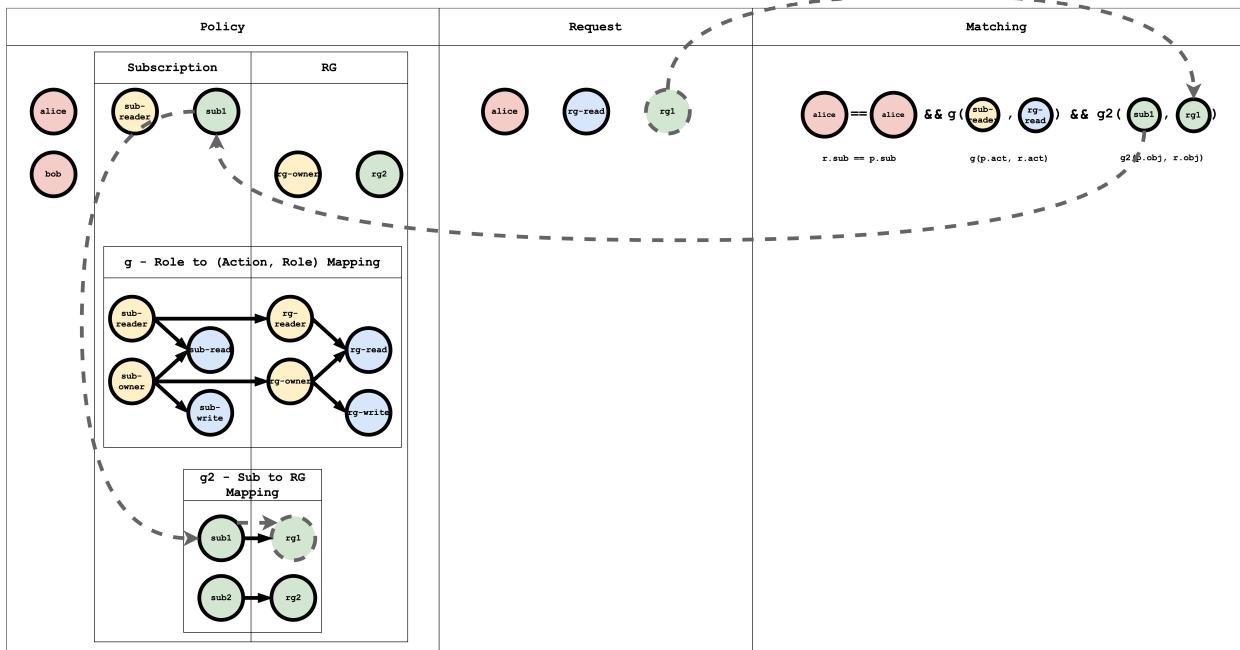
Subject Matching Visual representation



Action Matching Visual representation



Object Matching Visual representation



① 信息

When a request is submitted to Casbin, this matching happens for all the policies. If at least one policy matches, then the result of the request is true. If no policy matches the request, then the result is false.

Conclusion

In this tutorial, we learned about how different authorization models work and how they can be modeled using Casbin. In the second part of this tutorial, we will implement this in a demo Spring Boot Application and secure the APIs using Casbin.

访问控制模型

支持的模型

支持的 Casbin 模型

Model 的语法

Model 的语法

Effector

The Effector interface in Casbin

Functions

Using built-in functions or specifying custom functions

RBAC

Casbin RBAC 的使用

RBAC with Pattern

RBAC with Pattern

域内基于角色的访问控制

Usage of RBAC with domains

RBAC with Conditions

Usage of RBAC with conditions

Casbin RBAC和RBAC96

The Difference Between Casbin RBAC and RBAC96

ABAC

基于 Casbin 的 ABAC

优先级模型

Casbin's Priority Model for managing policies with different priorities

超级管理员

The Super Admin is the administrator of the entire system. It can be used in models such as RBAC, ABAC, and RBAC with domains.

支持的模型

1. [ACL \(Access Control List, 访问控制列表\)](#)
2. 具有 [超级用户](#) 的 ACL
3. **ACL without users:** This is especially useful for systems that don't have authentication or user logins.
4. **ACL without resources:** In some scenarios, the target is a type of resource instead of an individual resource. Permissions like "write-article" and "read-log" can be used. This doesn't control access to a specific article or log.
5. [RBAC \(基于角色的访问控制\)](#)
6. **RBAC with resource roles:** Both users and resources can have roles (or groups) at the same time.
7. **RBAC with domains/tenants:** Users can have different sets of roles for different domains/tenants.
8. [ABAC \(Attribute-Based Access Control\):](#) Syntax sugar like "resource.Owner" can be used to get the attribute for a resource.
9. [RESTful:](#) Supports paths like "/res/*", "/res/:id", and HTTP methods like "GET", "POST", "PUT", "DELETE".
10. **Deny-override:** Both allow and deny authorizations are supported, where deny overrides allow.
11. **Priority:** The policy rules can be prioritized, similar to firewall rules.

例子

访问控制模型	Model 文件	Policy 文件
ACL	basic_model.conf	basic_policy.csv
具有超	basic_with_root_model.conf	basic_policy.csv

访问控制模型	Model 文件	Policy 文件
级用户的 ACL		
没有用户的 ACL	basic_without_users_model.conf	basic_without_users_policy.csv
没有资源的 ACL	basic_without_resources_model.conf	basic_without_resources_policy.csv
RBAC	rbac_model.conf	rbac_policy.csv
具有资源角色的 RBAC	rbac_with_resource_roles_model.conf	rbac_with_resource_roles_policy.csv
带有域/租户的 RBAC	rbac_with_domains_model.conf	rbac_with_domains_policy.csv
ABAC	abac_model.conf	无
RESTful	keymatch_model.conf	keymatch_policy.csv
拒绝改写	rbac_with_not_deny_model.conf	rbac_with_deny_policy.csv
同意与拒绝	rbac_with_deny_model.conf	rbac_with_deny_policy.csv
优先级	priority_model.conf	priority_policy.csv

访问控制模型	Model 文件	Policy 文件
明确优先级	priority_model_explicit	priority_policy_explicit.csv
主体优先级	subject_priority_model.conf	subject_priority_policyl.csv

Model 的语法

- A model configuration (CONF) should have at least four sections:
[request_definition], [policy_definition], [policy_effect], and
[matchers].
- If a model uses Role-Based Access Control (RBAC), it should also include the
[role_definition] section.
- A model configuration (CONF) can contain comments. Comments start with
the # symbol, and everything after the # symbol will be commented out.

Request 定义

The [request_definition] section defines the arguments in the
e.Enforce(...) function.

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

In this example, sub, obj, and act represent the classic access triple: the subject (accessing entity), the object (accessed resource), and the action (access method). However, you can customize your own request format. For example, you can use sub, act if you don't need to specify a particular resource, or sub,
sub2, obj, act if you have two accessing entities.

Policy Definition

The `[policy_definition]` is the definition for a policy. 它界定了该策略的含义。例如，我们有以下模式：

```
[policy_definition]
p = sub, obj, act
p2 = sub, act
```

And we have the following policy (if in a policy file):

```
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， Each policy rule starts with a `policy type`, such as `p` or `p2`. 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)
```



The elements in a policy rule are always regarded as `strings`. If you have any questions about this, please refer to the discussion at:
<https://github.com/casbin/casbin/issues/113>

Policy Effect

[policy_effect] 部分是对policy生效范围的定义， It determines whether the access request should be approved if multiple policy rules match the request. 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))
```

The above policy effect means that if there's any matched policy rule of `allow`, the final effect is `allow` (also known as allow-override). `p.eft` is the effect for a policy, and it can be either `allow` or `deny`. It is optional, and the default value is `allow`. Since we didn't specify it above, it uses the default value.

Another example for the policy effect is:

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

This means that if there are no matched policy rules of `deny`, the final effect is `allow` (also known as deny-override). `some` means that there exists one matched policy rule. `any` means that all matched policy rules (not used here). The policy effect can even be connected with logical expressions:

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

This means that there must be at least one matched policy rule of `allow`, and

there cannot be any matched policy rule of `deny`. Therefore, in this way, both allow and deny authorizations are supported, and the deny overrides.

① 备注

Although we designed the syntax of the policy effect as above, the current implementations only use hard-coded policy effects. This is because we found that there isn't much need for that level of flexibility. So for now, you must use one of the built-in policy effects instead of customizing your own.

支持的 policy effects 如下：

Policy Effect	意义	示例
<code>some(where (p.eft == allow))</code>	allow-override	ACL, RBAC, etc.
<code>!some(where (p.eft == deny))</code>	deny-override	拒绝改写
<code>some(where (p.eft == allow)) && !some(where (p.eft == deny))</code>	allow-and-deny	同意与拒绝
<code>priority(p.eft) deny</code>	priority	优先级
<code>subjectPriority(p.eft)</code>	priority based on role	主题优先级

匹配器

[matchers] 是策略匹配器的定义。 The matchers are expressions that define how

the policy rules are evaluated against the request.

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

The above matcher is the simplest and means that the subject, object, and action in a request should match the ones in a policy rule.

Arithmetic operators like `+, -, *, /` and logical operators like `&&, ||, !` can be used in matchers.

Order of expressions in matchers

表达式的书写顺序对性能表现有很大影响。Take a look at the following example for more details:

```
const rbac_models = `

[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`
```

The enforcement time may be very long, up to 6 seconds.

```
go test -run ^TestManyRoles$ github.com/casbin/casbin/v2 -v

==== RUN TestManyRoles
    rbac_api_test.go:598: RESPONSE abu
/projects/1      GET : true IN: 438.379µs
    rbac_api_test.go:598: RESPONSE abu      /projects/
2499      GET : true IN: 39.005173ms
    rbac_api_test.go:598: RESPONSE jasmine
/projects/1      GET : true IN: 1.774319ms
    rbac_api_test.go:598: RESPONSE jasmine      /projects/
2499      GET : true IN: 6.164071648s
    rbac_api_test.go:600: More than 100 milliseconds for
jasmine /projects/2499 GET : 6.164071648s
    rbac_api_test.go:598: RESPONSE jasmine      /projects/
2499      GET : true IN: 12.164122ms
--- FAIL: TestManyRoles (6.24s)
FAIL
FAIL      github.com/casbin/casbin/v2      6.244s
FAIL
```

However, if we adjust the order of the expressions in matchers and put more time-consuming expressions like functions behind, the execution time will be very short.

Changing the order of expressions in matchers in the above example to:

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

```
go test -run ^TestManyRoles$ github.com/casbin/casbin/v2 -v
==== RUN TestManyRoles
```

Multiple Section Types

If you need multiple policy definitions or multiple matchers, you can use `p2` or `m2` as examples. In fact, all four sections mentioned above can use multiple types, and the syntax is `r` followed by a number, such as `r2` or `e2`. By default, these four sections should correspond one-to-one. For example, your `r2` section will only use the `m2` matcher to match `p2` policies.

You can pass an `EnforceContext` as the first parameter of the `enforce` method to specify the types. The `EnforceContext` is defined as follows:

[Go](#) [Node.js](#) [Java](#)

```
EnforceContext{"r2", "p2", "e2", "m2"}  
type EnforceContext struct {  
    RType string  
    PType string  
    EType string  
    MType string  
}  
  
const enforceContext = new EnforceContext('r2', 'p2', 'e2',  
    'm2');  
class EnforceContext {  
    constructor(rType, pType, eType, mType) {  
        this.pType = pType;  
        this.eType = eType;  
        this.mType = mType;  
        this.rType = rType;  
    }  
}
```

```
EnforceContext enforceContext = new EnforceContext("2");
public class EnforceContext {
    private String pType;
    private String eType;
    private String mType;
    private String rType;
    public EnforceContext(String suffix) {
        this.pType = "p" + suffix;
        this.eType = "e" + suffix;
        this.mType = "m" + suffix;
        this.rType = "r" + suffix;
    }
}
```

Here is an example usage. Please refer to the [model](#) and [policy](#). The request is as follows:

[Go](#) [Node.js](#) [Java](#)

```
// Pass in a suffix as a parameter to NewEnforceContext, such
// as 2 or 3, and it will create r2, p2, etc.
enforceContext := NewEnforceContext("2")
// You can also specify a certain type individually
enforceContext.EType = "e"
// Don't pass in EnforceContext; the default is r, p, e, m
e.Enforce("alice", "data2", "read")           // true
// Pass in EnforceContext
e.Enforce(enforceContext, struct{ Age int }{Age: 70}, "/data1",
"read")           // false
e.Enforce(enforceContext, struct{ Age int }{Age: 30}, "/data1",
"read")           // true
```

```
// Pass in a suffix as a parameter to NewEnforceContext, such
// as 2 or 3, and it will create r2, p2, etc.
const enforceContext = new NewEnforceContext('2');

// You can also specify a certain type individually
enforceContext.eType = "e"

// Don't pass in EnforceContext; the default is r, p, e, m
e.Enforce("alice", "data2", "read")           // true

// Pass in EnforceContext
e.Enforce(enforceContext, {Age: 70}, "/data1", "read")
//false
e.Enforce(enforceContext, {Age: 30}, "/data1", "read")
//true

// Pass in a suffix as a parameter to NewEnforceContext, such
// as 2 or 3, and it will create r2, p2, etc.
EnforceContext enforceContext = new EnforceContext("2");
// You can also specify a certain type individually
enforceContext.setType("e");
// Don't pass in EnforceContext; the default is r, p, e, m
e.enforce("alice", "data2", "read"); // true
// Pass in EnforceContext
// TestEvalRule is located in https://github.com/casbin/jcasbin/
blob/master/src/test/java/org/casbin/jcasbin/main/
AbacAPIUnitTest.java#L56
e.enforce(enforceContext, new
AbacAPIUnitTest.TestEvalRule("alice", 70), "/data1", "read");
// false
e.enforce(enforceContext, new
AbacAPIUnitTest.TestEvalRule("alice", 30), "/data1", "read");
// true
```

Special Grammar

You could also use the "in" operator, which is the only operator with a text name. This operator checks the array on the right-hand side to see if it contains a value that is equal to the value on the left side. Equality is determined by using the == operator, and this library does not check the types between the values. As long as two values can be cast to interface{} and can still be checked for equality with ==, they 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](#).

示例:

```
[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

Casbin的匹配器运算是由不同语言的表达式运算器实现的。 Casbin整合了他们的能力以提供统一的PERM语言。 In addition to the model syntax provided here, these expression evaluators may offer extra functionality that might not be supported by

another language or implementation. Please be cautious when using this functionality.

The expression evaluators used by each Casbin implementation are as follows:

实现	语言	Expression Evaluator
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/danthedeckie/simpleeval
Casbin.NET	C#	https://github.com/davideicardi/DynamicExpresso
Casbin4D	Delphi	https://github.com/casbin4d/Casbin4D/tree/master/SourceCode/Common/Third%20Party/TExpressionParser
casbin-rs	Rust	https://github.com/jonathandturner/rhai
casbin-cpp	C++	https://github.com/ArashPartow/exprtk

ⓘ 备注

If you encounter a performance issue with Casbin, it is likely caused by the

low efficiency of the expression evaluator. You can address the issue to Casbin or the expression evaluator directly for advice on speeding up the performance. For more details, please refer to the [Benchmarks](#) section.

Effector

The `Effect` represents the result of a policy rule, and the `Effector` is the interface for handling effects in Casbin.

MergeEffects()

The `MergeEffects()` function is used to merge all matching results collected by the enforcer into a single decision.

例如:

Go

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

在本示例中:

- `Effect` is the final decision that is merged by this function (initialized as `Indeterminate`).
- `explainIndex` is the index of `eft` (`Allow` or `Deny`), and it is initialized as `-1`.
- `err` 用于检查 `effect` 是否受到支持。
- `expr` is the string representation of the policy effects.
- `effects` is an array of effects, which can be `Allow`, `Indeterminate`, or

`Deny`.

- `matches` is an array that indicates whether the result matches the policy.
- `policyIndex` is the index of the policy in the model.
- `policyLength` 是策略的长度。

The code above illustrates how to pass the parameters to the `MergeEffects()` function, and the function will process the effects and matches based on the `expr`.

To use the `Effector`, follow these steps:

Go

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

The basic idea of `MergeEffects()` is that if the `expr` can match the results, indicating that the `p_eft` is `allow`, then all effects can be merged. If no deny rules are matched, then the decision is `allow`.

① 备注

If the `expr` does not match the condition "`priority(p_eft) || deny`", and the `policyIndex` is shorter than `policyLength-1`, it will short-circuit some effects in the middle.

Functions

Matchers中的函数

你甚至可以在Matcher中指定函数，使它更强大。 You can use built-in functions or specify your own function. The built-in key-matching functions take the following format:

```
bool function_name(string url, string pattern)
```

They return a boolean indicating whether the `url` matches the `pattern`.

支持的内置函数如下：

函数	url	模式	示例
keyMatch	一个URL 路径， 例如 <code>/alice_data/ resource1</code>	一个URL 路径或 <code>*</code> 模式下，例如 <code>/alice_data/*</code>	keymatch_model.conf/keymatch_policy.csv
keyMatch2	一个URL 路径， 例如 <code>/alice_data/ resource1</code>	一个URL 路径或 <code>:</code> 模式下，例如 <code>/alice_data/:resource</code>	keymatch2_model.conf/keymatch2_policy.csv
keyMatch3	一个URL 路径， 例如 <code>/alice_data/ resource1</code>	一个URL 路径或 <code>{}</code> 模式下，例如 <code>/alice_data/{resource}</code>	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L171-L196
keyMatch4	一个URL 路径， 例如 <code>/alice_data/ resource1</code>	一个URL 路径或 <code>{}</code> 模式下，例如 <code>/alice_data//{id}/book/{id}</code>	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L208-L222
keyMatch5	a URL path like <code>/alice_data/ 123/?status=1</code>	a URL path, a <code>{}</code> or <code>*</code> pattern like <code>/alice_data/{id}/*</code>	https://github.com/casbin/casbin/blob/1cde2646d10ad1190c0d784c3a1c0e1ace1b5bc9/util/builtin_operators_test.go#L485-L526
regexMatch	任意字符串	正则表达式模式	keymatch_model.conf/keymatch_policy.csv
ipMatch	一个 IP 地址，例 如 <code>192.168.2.123</code>	一个 IP 地址或一个 CIDR，例如 <code>192.168.2.0/24</code>	ipmatch_model.conf/ipmatch_policy.csv

函数	url	模式	示例
globMatch	类似路径的 /alice_data/ resource1	一个全局模式, 例如 /alice_data/*	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L426-L466

For key-getting functions, they usually take three parameters (except `keyGet`):

```
bool function_name(string url, string pattern, string key_name)
```

如果密钥 `key_name` 与模式匹配, 他们将返回它的值。如果没有匹配, 则返回 ""。

For example, `KeyGet2("/resource1/action", "/:res/action", "res")` will return "resource1", and `KeyGet3("/resource1_admin/action", "/{res}_admin/*", "res")` will return "resource1". As for `KeyGet`, which takes two parameters, `KeyGet("/resource1/action", "/")` will return "resource1/action".

函数	url	模式	密钥名称	示例
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_model.conf/keymatch_policy.csv
keyGet2	a URL path like /proj/ resource1	一个URL 路径或 : 模式下, 例如 /alice_data/:resource	模式中指 定的密钥 名称	keyget_model.conf/keymatch_policy.csv
keyGet3	一个URL 路 径, 例如 /proj/ res3_admin/	一个URL 路径或 {} 模式下, 例如 /proj/{resource}_admin/	key name specified in the pattern	https://github.com/casbin/casbin/blob/7bd496f94f5a2739a392d333a9aaa10ae397673/util/builtin_operators_test.go#L209-L247

See details for the above functions at: https://github.com/casbin/casbin/blob/master/util/builtin_operators_test.go

怎样增加自定义函数

First, prepare your function. It takes several parameters and returns a bool:

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

    if len(key1) > i {
```

Then, wrap it with `interface{}` types:

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

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

Finally, register the function to the Casbin enforcer:

```
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
```

RBAC

Role Definition

The `[role_definition]` is used to define the RBAC role inheritance relations. Casbin supports multiple instances of RBAC systems, where users can have roles and their inheritance relations, and resources can have roles and their inheritance relations too. These two RBAC systems won't interfere with each other.

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 = _, _
```

The above role definition shows that `g` is an RBAC system, and `g2` is another RBAC system. `_, _` means there are two parties involved in an inheritance relation. In the most common case, you usually use `g` alone if you only need roles for users. You can also use both `g` and `g2` when you need roles (or groups) for both users and resources. 请参见 [rbac_model](#) 和 [rbac_model_with_resource_roles](#) 的示例。

Casbin stores the actual user-role mapping (or resource-role mapping if you are using roles on resources) in the policy. For example:

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

It means that `alice` inherits/is a member of the role `data2_admin`. Here, `alice` can be a user, a resource, or a role. Cabin 只是将其识别为一个字符串。

Then, in a matcher, you should check the role as shown below:

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

It means that the `sub` in the request should have the role `sub` in the policy.

① 备注

1. Casbin 只存储用户角色的映射关系。
2. Casbin doesn't verify whether a user is a valid user or a role is a valid role. 这应该通过认证来解决。
3. Do not use the same name for a user and a role inside an RBAC system, because Casbin recognizes users and roles as strings, and there's no way for Casbin to know whether you are specifying user `alice` or role `alice`. 这时, 使用明确的 `role_alice`, 问题便可迎刃而解。
4. If `A` has role `B`, and `B` has role `C`, then `A` has role `C`. 这种传递性在当前版本会造成死循环。

① TOKEN NAME CONVENTION

Conventionally, the subject token name in the policy definition is `sub` and placed at the beginning. Now, Golang Casbin supports customized token names and places. If the subject token name is `sub`, the subject token can be placed at an arbitrary place without any extra action needed. If the subject token name is not `sub`, `e.SetFieldIndex()` for

`constant.SubjectIndex` should be called after the enforcer is initialized, regardless of its position.

```
# `subject` here is for sub
[policy_definition]
p = obj, act, subject
```

```
e.SetFieldIndex("p", constant.SubjectIndex, 2) // index
starts from 0
ok, err := e.DeleteUser("alice") // without SetFieldIndex,
it will raise an error
```

Role Hierarchy

Casbin's RBAC supports RBAC1's role hierarchy feature, which means that if `alice` has `role1`, and `role1` has `role2`, then `alice` will also have `role2` and inherit its permissions.

Here, we have a concept called a hierarchy level. So, in this example, the hierarchy level is 2. For the built-in role manager in Casbin, you can specify the maximum hierarchy level. 默认值为10。 This means that an end user like `alice` can only inherit 10 levels of roles.

```
// 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{}
```

How to Distinguish Role from User?

Casbin doesn't distinguish between roles and users in its RBAC. They are both treated as strings. If you only use a single-level RBAC (where a role will never be a member of another role), you can use `e.GetAllSubjects()` to get all users and `e.GetAllRoles()` to get all roles. They will list all `u` and all `r`, respectively, in all `g, u, r` rules.

But if you are using a multi-level RBAC (with role hierarchy) and your application doesn't record whether a name (string) is a user or a role, or you have a user and a role with the same name, you can add a prefix to the role like `role::admin` before passing it to Casbin. This way, you will know if it's a role by checking this prefix.

How to Query Implicit Roles or Permissions?

When a user inherits a role or permission via RBAC hierarchy instead of being directly assigned them in a policy rule, we call this type of assignment "implicit". To query such implicit relations, you need to use these two APIs:

`GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser()` instead of `GetRolesForUser()` and `GetPermissionsForUser()`. 有关详情, 请参阅 [this GitHub issue](#)。

Using Pattern Matching in RBAC

详情请参阅 [RBAC with Pattern](#)

Role Manager

See the [Role Managers](#) section for details.

RBAC with Pattern

快速入门

- Use pattern in `g(_)`.

```
e, _ := NewEnforcer("./example.conf", "./example.csv")
e.AddNamedMatchingFunc("g", "KeyMatch2", util.KeyMatch2)
```

- Use pattern with domain.

```
e.AddNamedDomainMatchingFunc("g", "KeyMatch2",
util.KeyMatch2)
```

- Use all patterns.

Just combine the use of both APIs.

As shown above, after you create the `enforcer` instance, you need to activate pattern matching via the `AddNamedMatchingFunc` and `AddNamedDomainMatchingFunc` APIs, which determine how the pattern matches.

 备注

如果您使用在线编辑器，它会在左下角指定模式匹配函数。

```
g10matche
12      *
  matchingDomainForGFunction:
  'keyMatch'
13      */
14      matchingForGFunction:
  'keyMatch2',
15      matchingDomainForGFunction:
  'keyMatch2'
16  };
17 })();
```

Request

```
1 /book/1
2 /book/1
3
```

在 RBAC 中使用模式匹配

Sometimes, you want certain subjects, objects, or domains/tenants with a specific pattern to be automatically granted a role. 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](#)。

The pattern matching function supports each parameter of `g`.

We know that normally RBAC is expressed as `g(r.sub, p.sub)` in a matcher.
Then we can use a policy like:

```
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`。 But there can be thousands of books, and it's very tedious to add each book to the book role (or group) with one `g` policy rule.

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 will automatically match `/book/1` and `/book/2` into the pattern `/book/:id` for you. 您需要做的仅仅是向enforcer注册该方法，例如像这样：

[Go](#) [Node.js](#)

```
e.AddNamedMatchingFunc("g", "KeyMatch2", util.KeyMatch2)
```

```
await e.addNamedMatchingFunc('g', Util.keyMatch2Func);
```

When using a pattern matching function in domains/tenants, you need to register the function with the enforcer and model.

[Go](#) [Node.js](#)

```
e.AddNamedDomainMatchingFunc("g", "KeyMatch2", util.KeyMatch2)
```

```
await e.addNamedDomainMatchingFunc('g', Util.keyMatch2Func);
```

如果您不理解 `g(r.sub, p.sub, r.dom)` 意味着什么，请阅读 [rbac-with-domains](#)。简而言之，`g(r.sub, p.sub, r.dom)` 将检查用户 `r.sub` 在域内 `r.dom` 是否具有角色 `p.sub` So this is how the matcher works. 您可以在这里查看完整的示例。

除了上面的模式匹配语法外，我们还可以使用纯域模式。

For example, if we want `sub` to have access in different domains, `domain1` and `domain2`, we can use the pure domain pattern:

```
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
```

In this example, we want `alice` to read and write `data` in domain1 and domain2. Pattern matching `*` in `g` makes `alice` have access to two domains.

By using pattern matching, especially in scenarios that are more complicated and have a lot of domains or objects to consider, we can implement the `policy_definition` in a more elegant and effective way.

域内基于角色的访问控制

Role Definition with Domain Tenants

在Casbin中的RBAC角色可以是全局或是基于特定于域的。 Domain-specific roles mean that the roles for a user can be different when the user is in different domains/tenants. This is very useful for large systems like a cloud, as users are usually in different tenants.

The role definition with domains/tenants should look like this:

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

The third `_` represents the name of the domain/tenant, and this part should not be changed. 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
```

This means that the `admin` role in `tenant1` can read `data1`. And `alice` has the `admin` role in `tenant1` and the `user` role in `tenant2`. Therefore, she can read `data1`. 同理，因为`alice`不是`tenant2`的`admin`，所以她访问不了`data2`。

Then, in a matcher, you should check the role as follows:

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

Please refer to the [rbac_with_domains_model.conf](#) for examples.

(!) TOKEN NAME CONVENTION

Note: Conventionally, the domain token name in policy definition is `dom` and is placed as the second token (`sub, dom, obj, act`). Now, Golang Casbin supports customized token names and placement. If the domain token name is `dom`, the domain token can be placed at an arbitrary position without any additional action. If the domain token name is not `dom`, `e.SetFieldIndex()` for `constant.DomainIndex` should be called after the enforcer is initialized, regardless of its position.

```
# "domain" 在这里为 "dom"
[policy_definition]
p = sub, obj, act, domain
```

```
e.SetFieldIndex("p", constant.DomainIndex, 3) // index
starts from 0
users := e.GetAllUsersByDomain("domain1") // without
SetFieldIndex, it will raise an error
```

RBAC with Conditions

Conditional RoleManager

`ConditionalRoleManager` supports custom condition functions at the policy level.

For example, when we need a temporary role policy, we can follow the following approach:

`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
```

`g = _, _, (_, _)` uses `(_, _)` to contain a list of arguments to pass to the condition function and `_` as a parameter placeholder

policy.csv

```
p, alice, data1, read
p, data2_admin, data2, write
p, data3_admin, data3, read
p, data4_admin, data4, write
p, data5_admin, data5, read
p, data6_admin, data6, write
p, data7_admin, data7, read
p, data8_admin, data8, write

g, alice, data2_admin, 0000-01-01 00:00:00, 0000-01-02 00:00:00
g, alice, data3_admin, 0000-01-01 00:00:00, 9999-12-30 00:00:00
g, alice, data4_admin, _, _
g, alice, data5_admin, _, 9999-12-30 00:00:00
g, alice, data6_admin, _, 0000-01-02 00:00:00
g, alice, data7_admin, 0000-01-01 00:00:00, _
g, alice, data8_admin, 9999-12-30 00:00:00, _
```

Basic Usage

Add a conditional function for the role policy(g type policy) through `AddNamedLinkConditionFunc`, and when enforcing is executed, the corresponding parameters will be automatically obtained and passed in the conditional function for checking. If the check passes, then the corresponding role policy(g type policy) is valid, otherwise it is invalid

```
e.AddNamedLinkConditionFunc("g", "alice", "data2_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data3_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data4_admin",
util.TimeMatchFunc)
```

Custom condition functions

Custom conditional functions need to conform to the following function types

```
type LinkConditionFunc = func(args ...string) (bool, error)
```

for example:

```
// TimeMatchFunc is the wrapper for TimeMatch.
func TimeMatchFunc(args ...string) (bool, error) {
    if err := validateVariadicStringArgs(2, args...); err != nil {
        return false, fmt.Errorf("%s: %s", "TimeMatch", err)
    }
    return TimeMatch(args[0], args[1])
}

// TimeMatch determines whether the current time is between
// startTime and endTime.
// You can use "_" to indicate that the parameter is ignored
func TimeMatch(startTime, endTime string) (bool, error) {
    now := time.Now()
    if startTime != "_" {
        if start, err := time.Parse("2006-01-02 15:04:05",
startTime); err != nil {
            return false, err
        } else if !now.After(start) {
            return false, nil
        }
    }

    if endTime != "_" {
        if end, err := time.Parse("2006-01-02 15:04:05",
endTime); err != nil {
            return false, err
        } else if now.After(end) {
            return false, nil
        }
    }
}
```

Conditional RoleManager with domains

model.conf

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

[policy_definition]
p = sub, dom, obj, act

[role_definition]
g = _, _, _, (_, _)

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

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

policy.csv

```
p, alice, data1, read
p, data2_admin, data2, write
p, data3_admin, data3, read
p, data4_admin, data4, write
p, data5_admin, data5, read
p, data6_admin, data6, write
p, data7_admin, data7, read
p, data8_admin, data8, write

g, alice, data2_admin, domain2, 0000-01-01 00:00:00, 0000-01-02
```

Basic Usage

Add a conditional function for the role policy(g type policy) through `AddNamedDomainLinkConditionFunc`, and when enforcing is executed, the corresponding parameters will be automatically obtained and passed in the conditional function for checking. If the check passes, then the corresponding role policy(g type policy) is valid, otherwise it is invalid

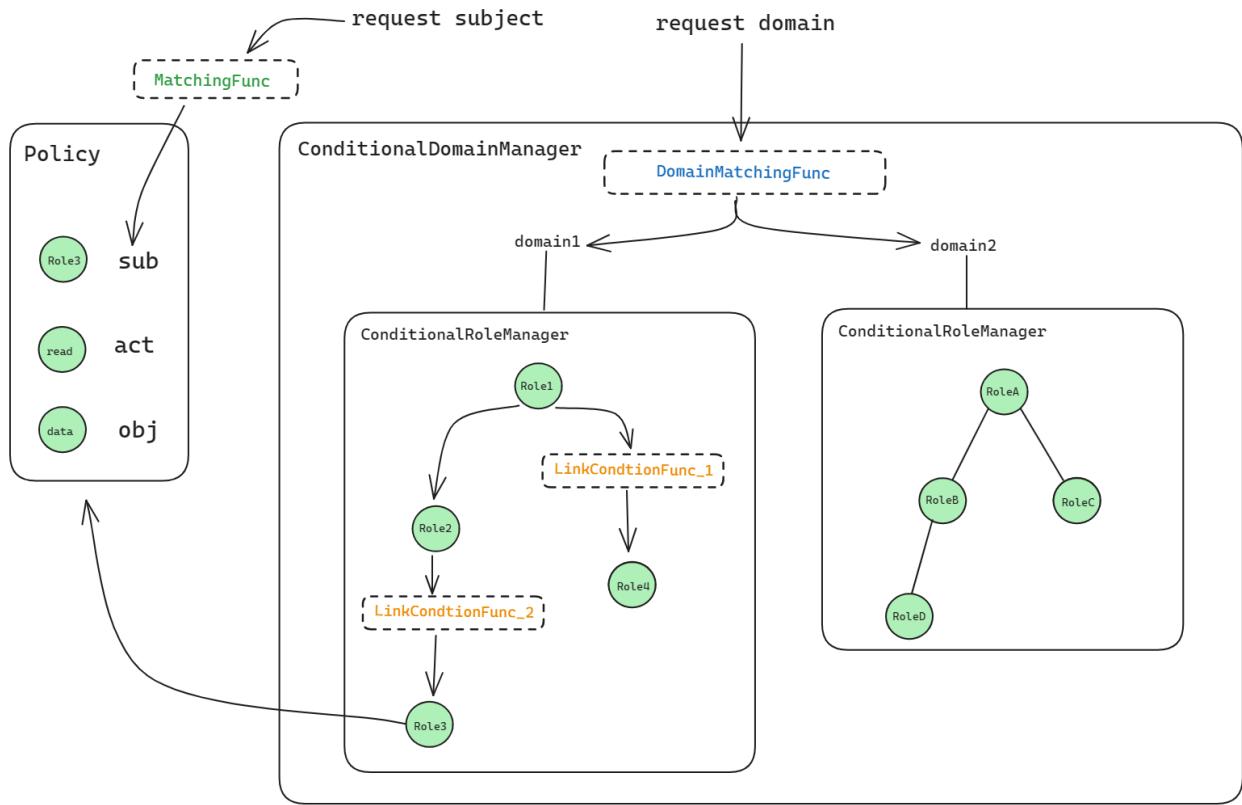
```
e.AddNamedDomainLinkConditionFunc("g", "alice", "data2_admin",
"domain2", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data3_admin",
"domain3", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data4_admin",
"domain4", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data5_admin",
"domain5", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data6_admin",
"domain6", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data7_admin",
"domain7", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data8_admin",
"domain8", util.TimeMatchFunc)

e.enforce("alice", "domain1", "data1", "read")          //
except: true
e.enforce("alice", "domain2", "data2", "write")         //
except: false
e.enforce("alice", "domain3", "data3", "read")          //
except: true
e.enforce("alice", "domain4", "data4", "write")         //
except: true
e.enforce("alice", "domain5", "data5", "read")          //
except: true
```

Custom condition functions

Like the basic `Conditional RoleManager`, custom functions are supported, and there is no difference in use.

Note that `DomainMatchingFunc`, `MatchingFunc`, and `LinkConditionFunc` are at different levels and are used in different situations.



Casbin RBAC和RBAC96

Casbin RBAC和RBAC96

In this document, we will compare Casbin RBAC with [RBAC96](#).

Casbin RBAC supports nearly all the features of RBAC96 and adds new features on top of that.

RBAC Version	支持级别	说明
RBAC0	Fully Supported	RBAC0是RBAC96的基本版本。 It clarifies the relationship between Users, Roles, and Permissions.
RBAC1	Fully Supported	RBAC1 adds role hierarchies on top of RBAC0. This means that if <code>alice</code> has <code>role1</code> , <code>role1</code> has <code>role2</code> , then <code>alice</code> will also have <code>role2</code> and inherit its permissions.
RBAC2	Mutually Exclusive Handling Supported (like this)	RBAC2 adds constraints on RBAC0. This allows RBAC2 to handle mutually exclusive policies. However, quantitative limits are not supported.

RBAC Version	支持级别	说明
RBAC3	Mutually Exclusive Handling Supported (like this)	RBAC3是RBAC1和RBAC2的组合。 It supports role hierarchies and constraints found in RBAC1 and RBAC2. However, quantitative limits are not supported.

The Difference Between Casbin RBAC and RBAC96

1. In Casbin, the distinction between User and Role is not as clear as in RBAC96.

在Casbin中， 用户和角色都被视为字符串。 For example, consider the following policy file:

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

If you call the method `GetAllSubjects()` using an instance of the Casbin Enforcer:

```
e.GetAllSubjects()
```

the return value will be:

```
[admin alice]
```

This is because in Casbin, subjects include both Users and Roles.

However, if you call the method `GetAllRoles()`:

```
e.GetAllRoles()
```

the return value will be:

```
[admin]
```

From this, you can see that there is a distinction between Users and Roles in Casbin, but it is not as sharp as in RBAC96. Of course, you can add a prefix to your policies such as `user::alice` and `role::admin` to clarify their relationships.

2. Casbin RBAC provides more permissions than RBAC96.

RBAC96 defines only 7 permissions: read, write, append, execute, credit, debit, and inquiry.

然而，在Casbin中，我们将权限视为字符串。 This allows you to create permissions that better suit your needs.

3. Casbin RBAC supports domains.

In Casbin, you can perform authorizations based on domains. This feature makes your Access Control Model more flexible.

ABAC

What is the ABAC model?

ABAC stands for Attribute-Based Access Control. It allows you to control access by using the attributes (properties) of the subject, object, or action instead of using the string values themselves. You may have heard of a complicated ABAC access control language called XACML. Casbin's ABAC, on the other hand, is much simpler. In Casbin's ABAC, you can use structs or class instances instead of strings for model elements.

Let's take a look at the official ABAC example:

```
[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
```

In the matcher, we use `r.obj.Owner` instead of `r.obj`. The `r.obj` passed in the `Enforce()` function will be a struct or class instance rather than a string. 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
}
```

If you want to pass parameters to the enforcer through JSON, you need to enable the function with `e.EnableAcceptJsonRequest(true)`.

For example:

```
e, _ := NewEnforcer("examples/abac_model.conf")
e.EnableAcceptJsonRequest(true)

data1Json := `{"Name": "data1", "Owner": "bob"}

ok, _ := e.Enforce("alice", data1Json, "read")
```

① 备注

Enabling the function of accepting JSON parameters may result in a performance drop of 1.1 to 1.5 times.

如何使用ABAC?

To use ABAC, you need to do two things:

1. 在模型匹配器中指定属性。
2. Pass in the struct or class instance for the element as an argument to Casbin's `Enforce()` function.

🔥 危险

Currently, only request elements like `r.sub`, `r.obj`, `r.act`, and so on support ABAC. You cannot use it on policy elements like `p.sub` because there is no way to define a struct or class in Casbin's policy.

💡 提示

You can use multiple ABAC attributes in a matcher. For example: `m = r.sub.Domain == r.obj.Domain`.

💡 提示

If you need to use a comma in a policy that conflicts with CSV's separator, you can escape it by surrounding the statement with quotation marks. 例如, `"keyMatch("bob", r.sub.Role)"` 将不会被视为csv文件分割。

Scaling the model for complex and large numbers of ABAC rules

The above implementation of the ABAC model is simple at its core. However, in many cases, the authorization system requires a complex and large number of ABAC rules. To accommodate this requirement, it is recommended to add the rules in the policy instead of the model. This can be done by introducing an `eval()` functional construct. Here is an example:

This is the definition of the `CONF` file used to define the ABAC model.

```
[request_definition]
```

In this example, `p.sub_rule` is a struct or class (user-defined type) that contains the necessary attributes to be used in the policy.

这是针对 Enforcement 模型使用的策略 Now, you can use the object instance passed to `eval()` as a parameter to define certain ABAC constraints.

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

优先级模型

Casbin supports loading policies with priority.

Load Policy with Implicit Priority

It's quite simple: the order determines the priority; policies that appear earlier have higher priority.

model.conf:

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

Load Policy with Explicit Priority

另见: [casbin#550](#)

A smaller priority value indicates a higher priority. If there's a non-numerical character in the priority, it will be placed last instead of throwing an error.

⚠ TOKEN名称协议

The conventionally used priority token name in the policy definition is "priority". To use a custom one, you need to invoke `e.SetFieldIndex()` and reload the policies (see the full example on [TestCustomizedFieldIndex](#)).

model.conf:

```
[policy_definition]
p = customized_priority, sub, obj, act, eft
```

Golang代码示例:

```
e, _ := NewEnforcer("./example/
priority_model_explicit_customized.conf",
                    "./example/
priority_policy_explicit_customized.csv")
// Due to the customized priority token, the enforcer
fails to handle the priority.
ok, err := e.Enforce("bob", "data2", "read") // the result
will be `true, nil`
// Set PriorityIndex and reload
e.SetFieldIndex("p", constant.PriorityIndex, 0)
err := e.LoadPolicy()
if err != nil {
    log.Fatalf("LoadPolicy: %v", err)
}
ok, err := e.Enforce("bob", "data2", "read") // the result
will be `false, nil`
```

Currently, explicit priority only supports `AddPolicy` & `AddPolicies`. If `UpdatePolicy` has been called, you shouldn't change the priority attribute.

model.conf:

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

```
alice, data1, write --> true // because `p, 1, alice, data1,
write, allow` has the highest priority
bob, data2, read --> false
bob, data2, write --> true // because bob has the role of
`data2_allow_group` which has the right to write data2, and
there's no deny policy with higher priority
```

基于角色和用户层次结构以优先级加载策略

角色和用户的继承结构只能是多棵树，而不是图。 If a user has multiple roles, you have to make sure the user has the same level in different trees. If two roles have the same level, the policy (associated with the role) that appeared earlier has higher priority. For more details, also see [casbin#833](#) and [casbin#831](#).

model.conf:

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

[policy_definition]
p = sub, obj, act, eft

[role_definition]
g = _, _

[policy_effect]
e = subjectPriority(p.eft) || deny

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

policy.csv

```
p, root, data1, read, deny
p, admin, data1, read, deny

p, editor, data1, read, deny
p, subscriber, data1, read, deny

p, jane, data1, read, allow
p, alice, data1, read, allow

g, admin, root

g, editor, admin
g, subscriber, admin

g, jane, editor
g, alice, subscriber
```

请求:

```
jane, data1, read --> true // because jane is at the bottom,  
her priority is higher than that of editor, admin, and root  
alice, data1, read --> true
```

The role hierarchy looks like this:

```
角色: 根  
  └ 角色: 管理员  
    ┌ 角色编辑器  
    | └ 用户: 简  
    └ 角色: 订阅者  
      └ 用户: 爱丽丝
```

The priority automatically looks like this:

```
role: root # 自动优先级: 30  
--role: admin# 自动优先级: 20  
--role: editor # 自动优先级: 10  
--role: subscriber # 自动优先级: 10
```

超级管理员

The Super Admin is the administrator of the entire system. It can be used in models such as RBAC, ABAC, and RBAC with domains. 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"
```

This example illustrates that, with the defined `request_definition`, `policy_definition`, `policy_effect`, and `matchers`, Casbin determines whether the request can match the policy. One important aspect is checking if the `sub` is root. If the judgment is correct, authorization is granted, and the user has permission to perform all actions.

Similar to the root user in Linux systems, being authorized as root grants access to all files and settings. If we want a `sub` to have full access to the entire system, we can assign it the role of Super Admin, granting the `sub` permission to perform all actions.



>

存储

存储



Model 的存储

Model 的存储



Policy的存储

Policy的存储



策略子集加载

Loading filtered policies

Model 的存储

Unlike the policy, the model can only be loaded, it cannot be saved. We believe that the model is not a dynamic component and should not be modified at runtime, so we have not implemented an API to save the model into storage.

However, there is good news. We provide three equivalent ways to load a model, either statically or dynamically:

从 .CONF 文件中加载 model

这是使用Casbin的最常见方式。 It is easy to understand for beginners and convenient for sharing when you need help from the Casbin team.

The content of the `.CONF` file `examples/rbac_model.conf` is as follows:

```
[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
```

Then you can load the model file as follows:

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

从代码加载 model

The model can be initialized dynamically from code instead of using a `.CONF` file.

以下是RBAC模式的一个示例：

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

// 从Go代码初始化模型
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")

// 从CSV文件adapter加载策略规则
// Replace it with your adapter to avoid using files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")

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

从字符串加载的 model

Alternatively, you can load the entire model text from a multi-line string. The advantage of this approach is that you do not need to maintain a model file.

```
import (
    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin/v2/model"
)

// 从字符串初始化模型
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)

// 从CSV文件adapter加载策略规则
// Replace it with your adapter to avoid using files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")
```


Policy的存储

在Casbin, 策略存储作为 [适配器](#) 来实现。

Loading policy from a .CSV file

这是使用Casbin的最常见方式。 It is easy to understand for beginners and convenient for sharing when you ask the Casbin team for help.

The content of the [.CSV](#) file [examples/rbac_policy.csv](#) is as follows:

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

备注

If your file contains commas, you should wrap them in double quotes. For example:

```
p, alice, "data1,data2", read      --correct
p, alice, data1,data2, read        --incorrect (the whole
phrase "data1,data2" should be wrapped in double quotes)
```

If your file contains commas and double quotes, you should enclose the

field in double quotes and double any embedded double quotes.

```
p, alice, data, "r.act in (''get'', ''post'')"  
correct  
p, alice, data, "r.act in ('get', 'post')"  
incorrect (you should use "" to escape "")
```

相关问题: [issue#886](#)

适配器 API

接口名	类型	描述
LoadPolicy()	基本设置	从持久层中加载policy规则
SavePolicy()	基本设置	将policy规则保存至持久层
AddPolicy()	可选	添加单条policy规则至持久层
RemovePolicy()	可选	从持久层删除单条policy规则
RemoveFilteredPolicy()	可选	从持久层删除符合筛选条件的policy规则

数据库存储格式

Your policy file

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

Corresponding database structure (such as MySQL)

id	ptype	v0	v1	v2	v3	v4	v5
1	p	data2_admin	data2	可读			
2	p	data2_admin	data2	可写			
3	g	alice	admin				

Meaning of each column

- `id`: The primary key in the database. It does not exist as part of the `casbin policy`. The way it is generated depends on the specific adapter.
- `ptype`: 它对应 `p`, `g`, `g2`, 等等。
- `v0 - v5`: The column names have no specific meaning and correspond to the values in the `policy csv` from left to right. 列数取决于您自己定义的数量。 In theory, there can be an infinite number of columns, but generally only 6 columns are implemented in the adapter. If this is not enough for you, please submit an issue to the corresponding adapter repository.

适配器详情

For more details about the use of the adapter API and database table structure design, please visit: [/docs/adapters](#)

策略子集加载

一些adapter支持过滤策略管理。 This means that the policy loaded by Casbin is a subset of the policy stored in the database based on a given filter. This allows for efficient policy enforcement in large, multi-tenant environments where parsing the entire policy becomes a performance bottleneck.

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/v2"
    fileadapter "github.com/casbin/casbin/v2/persist/file-
adapter"
)

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"},
}
```

There is another method that supports the subset loading feature:

`LoadIncrementalFilteredPolicy`. `LoadIncrementalFilteredPolicy` is similar to `LoadFilteredPolicy`, but it does not clear the previously loaded policy. It only appends the filtered policy to the existing policy.

扩展功能

执行器

The Enforcer is the main structure in Casbin that acts as an interface for users to perform operations on policy rules and models.

适配器

支持的适配器和用法

监视器

Maintaining consistency between multiple Casbin enforcer instances

调度器

Dispatchers provide a way to synchronize incremental changes of policy.

角色管理器

The role manager is used to manage the RBAC role hierarchy in Casbin.

中间件

Casbin 中间件

Graphql-中间件

Authorization for GraphQL endpoints

云端原生中间件

云端原生中间件

执行器

The `Enforcer` is the main structure in Casbin. It acts as an interface for users to perform operations on policy rules and models.

Supported Enforcers

A complete list of Casbin enforcers is provided below. Any 3rd-party contribution on a new enforcer is welcomed. Please inform us, and we will add it to this list :)

[Go](#)

[Python](#)

Enforcer	Author	Description
<code>Enforcer</code>	Casbin	The <code>Enforcer</code> is the basic structure for users to interact with Casbin policies and models. You can find more details about the <code>Enforcer</code> API here .
<code>CachedEnforcer</code>	Casbin	The <code>CachedEnforcer</code> is based on the <code>Enforcer</code> and supports caching the evaluation result of a request in memory using a map. It provides the ability to clear caches within a specified expiration time. Moreover, it guarantees thread safety with a Read-Write lock. You can use <code>EnableCache</code> to enable caching of

Enforcer	Author	Description
		<p>evaluation results (default is enabled). The other API methods of <code>CachedEnforcer</code> are the same as <code>Enforcer</code>.</p>

<code>DistributedEnforcer</code>	Casbin	<p>The <code>DistributedEnforcer</code> supports multiple instances in distributed clusters. It wraps the <code>SyncedEnforcer</code> for the dispatcher. You can find more details about the dispatcher here.</p>
----------------------------------	--------	--

<code>SyncedEnforcer</code>	Casbin	<p>The <code>SyncedEnforcer</code> is based on the <code>Enforcer</code> and provides synchronized access. It is thread-safe.</p>
-----------------------------	--------	---

<code>SyncedCachedEnforcer</code>	Casbin	<p>The <code>SyncedCachedEnforcer</code> wraps the <code>Enforcer</code> and provides decision sync cache.</p>
-----------------------------------	--------	--

Enforcer	Author	Description
<code>Enforcer</code>	Casbin	<p>The <code>Enforcer</code> is the basic structure for users to interact with Casbin policies and models. You can find more details about the <code>Enforcer</code> API here.</p>
<code>DistributedEnforcer</code>	Casbin	<p>The <code>DistributedEnforcer</code> supports multiple instances in distributed clusters. It</p>

Enforcer	Author	Description
		wraps the <code>SyncedEnforcer</code> for the dispatcher. You can find more details about the dispatcher here .
<code>SyncedEnforcer</code>	Casbin	The <code>SyncedEnforcer</code> is based on the <code>Enforcer</code> and provides synchronized access. It is thread-safe.
<code>AsyncEnforcer</code>	Casbin	The <code>AsyncEnforcer</code> provides async API.
<code>FastEnforcer</code>	Casbin	The <code>FastEnforcer</code> uses a new model which is 50x faster than the normal model. You can find more here

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](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Ruby](#) [Swift](#) [Lua](#)

适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Filtered File Adapter (内置)	File	File	✗	对于 CSV (逗号分隔的值) 个带策略子集加载支持的文件
SQL Adapter	SQL	@Blank-Xu	✓	MySQL, PostgreSQL, SQL Server和SQLite3在 <code>master</code> 分支中受到 <code>database/sql</code> 的支持，Oacle在 <code>oracle</code> 分支中也受到 <code>database/sql</code> 的支持
Xorm Adapter	ORM	Casbin	✓	通过 Xorm 实现，支持 MySQL, PostgreSQL, Sqlite3, SQL Server 等多种存储引擎的 adapter

适配器	类型	作者	自动保存	描述
GORM Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, Sqlite3, SQL Server are supported by GORM
GORM Adapter Ex	ORM	Casbin	✓	MySQL, PostgreSQL, Sqlite3, SQL Server are supported by GORM
Ent Adapter	ORM	Casbin	✓	MySQL, MariaDB, PostgreSQL, SQLite, 基于 Gremlin的图数据库由 [ent ORM](https://entgo.io/) 支持。
Beego ORM Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, Sqlite3 are supported by Beego ORM
SQLX Adapter	ORM	@memwey	✓	MySQL, PostgreSQL, SQLite, Oracle 由 SQLX 支持
Sqlx Adapter	ORM	@Blank-Xu	✓	MySQL, PostgreSQL, SQL Server, SQLite3 由 <code>master</code> 分支支持, Oracle由 <code>oracle</code> 分支受到 sqlx 的支持
GF ORM Adapter	ORM	@vance-liu	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
GoFrame ORM Adapter	ORM	@kotlin2018	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
gf-adapter	ORM	@zcyc	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
Gdb Adapter	ORM	@jxo-me	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM

适配器	类型	作者	自动保存	描述
GoFrame V2 Adapter	ORM	@hailaz	✓	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
Filtered PostgreSQL Adapter	SQL	Casbin	✓	用于PostgreSQL
Filtered pgx Adapter	SQL	@pckhoi	✓	使用 pgx 支持PostgreSQL
PostgreSQL Adapter	SQL	@cychiuae	✓	用于PostgreSQL
RQLite Adapter	SQL	EDOMO Systems	✓	用于 RQLite
MongoDB Adapter	NoSQL	Casbin	✓	基于 MongoDB Go Driver 用于 MongoDB
RethinkDB Adapter	NoSQL	@adityapandey9	✓	用于RethinkDB
Cassandra Adapter	NoSQL	Casbin	✗	用于Apache Cassandra DB
DynamoDB Adapter	NoSQL	HOOQ	✗	用于Amazon DynamoDB
Dynacasbin	NoSQL	NewbMiao	✓	用于Amazon DynamoDB
ArangoDB Adapter	NoSQL	@adamwasila	✓	用于ArangoDB

适配器	类型	作者	自动保存	描述
Amazon S3 Adapter	云	Soluto	✗	用于 Minio 和 Amazon S3
Azure Cosmos DB Adapter	云	@spacycoder	✓	用于 Microsoft Azure Cosmos DB
GCP Firestore Adapter	云	@reedom	✗	用于 Google Cloud Platform Firestore
GCP Cloud Storage Adapter	云	qurami	✗	用于 Google Cloud Platform Cloud Storage
GCP Cloud Storage Adapter	云	@flowerinthenight	✓	用于 Google Cloud Platform Cloud Spanner
Consul Adapter	KV store	@ankitm123	✗	用于 HashiCorp Consul
Redis 适配器 (Redigo)	KV store	Casbin	✓	用于 Redis
Redis Adapter (go-redis)	KV store	@milsen	✓	用于 Redis
Etcd 适配器	KV store	@sebastianliu	✗	用于 etcd
BoltDB	KV	@speza	✓	用于 Bolt

适配器	类型	作者	自动保存	描述
Adapter	store			
Bolt Adapter	KV store	@wirepair	✗	用于 Bolt
BadgerDB Adapter	KV store	@inits	✓	用于 BadgerDB
Protobuf Adapter	Stream	Casbin	✗	用于 Google Protocol Buffers
JSON Adapter	String	Casbin	✗	用于 JSON
String Adapter	String	@qiangmzsx	✗	用于 String
HTTP File Adapter	HTTP	@h4ckedneko	✗	用于 http.FileSystem
FileSystem Adapter	File	@naucon	✗	用于fs.FS 和 embed.FS
适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
JDBC	JDBC	Casbin	✓	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL 服

适配器	类型	作者	自动保存	描述
Adapter				服务器由 JDBC 支持
Hibernate Adapter	ORM	Casbin	✓	Oracle, DB2, SQL Server, Sybase, MySQL, PostgreSQL 由 Hibernate 支持
MyBatis Adapter	ORM	Casbin	✓	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server (与 JDBC 相同) 由 MyBatis 3 支持
Hutool Adapter	ORM	@mapleafgo	✓	MySQL, Oracle, PostgreSQL, SQLite 由 Hutool 支持
MongoDB Adapter	NoSQL	Casbin	✓	MongoDB 由 Mongodb-driver-sync 支持
DynamoDB Adapter	NoSQL	Casbin	✗	用于 Amazon DynamoDB
Redis Adapter	KV store	Casbin	✓	用于 Redis
适配器	实现要素	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Filtered File Adapter (内置)	File	Casbin	✗	对于 CSV (逗号分隔的值) 带策略子集加载支持的文件

适配器	实现要素	作者	自动保存	描述
String Adapter (内置)	String	@calebfaruki	✗	用于字符串
Basic Adapter	Native ORM	Casbin	✓	pg, mysql, mysql2, sqlite3, oracledb, mssql 是适配器本身支持的
Sequelize Adapter	ORM	Casbin	✓	MySQL、PostgreSQL、SQLite、Microsoft SQL Server 由 Sequelize 支持
TypeORM Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL, MongoDB 由 TypeORM 支持
Prisma Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, AWS Aurora, Azure SQL 由 Prisma 支持
Knex Adapter	ORM	@sarneeh and knex	✓	MSSQL, MySQL, PostgreSQL, SQLite3, Oracle 由 Knex.js 支持
Objection.js Adapter	ORM	@willsoto	✓	MSSQL, MySQL, PostgreSQL, SQLite3, Oracle 由 Objection.js 支持
MikroORM Adapter	ORM	@baisheng	✓	MongoDB, MySQL, MariaDB, PostgreSQL, SQLite are supported by MikroORM
Node PostgreSQL Native Adapter	SQL	@touchifyapp	✓	PostgreSQL 适配器，拥有高级策略子集加载支持以及由 [node-postgres](https://node-postgres.com/) 构建的更好的性能。

适配器	实现要素	作者	自动保存	描述
MongoDB Adapter	NoSQL	elastic.io 和 Casbin	✓	MongoDB 由 Mongoose 支持
Mongoose 适配器 (无交易)	NoSQL	minhducck	✓	MongoDB is supported by Mongoose
Node MongoDB Native Adapter	NoSQL	@juicycleff	✓	用于 Node MongoDB Native
DynamoDB Adapter	NoSQL	@fospitia	✓	用于 Amazon DynamoDB
Couchbase Adapter	NoSQL	@MarkMYoung	✓	用于 Couchbase
Redis Adapter	KV store	Casbin	✗	用于 Redis
Redis Adapter	KV store	@NandaKishorJeripothula	✗	For Redis
适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files

适配器	类型	作者	自动保存	描述
数据库适配器	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server 由 techone/database 支持
Zend Db 适配器	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Oracle, IBM DB2, Microsoft SQL Server, 其他 PDO Driver 由 zend-db 支持
Doctrine DBAL 适配器(建议)	ORM	Casbin	✓	强大的 PHP 数据库抽象层(DBAL)，具有数据库架构内省和管理的许多功能。
Medoo 适配器	ORM	Casbin	✓	Medoo 是一个用来加速开发的轻量PHP 数据库框架。 支持所有 SQL 数据库，包括 MySQL, MSSQL, SQLite, MariaDB, PostgreSQL, Sybase, Oracle 以及更多。
Laminas-db 适配器	ORM	Casbin	✓	MySQL, PostgreSQL, Oracle, IBM DB2, Microsoft SQL Server, PDO, etc. are supported by laminas-db
Zend-db 适配器	ORM	Casbin	✓	MySQL, PostgreSQL, Oracle, IBM DB2, Microsoft SQL Server, PDO, etc. are supported by zend-db
ThinkORM Adapter (ThinkPHP)	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Oracle, Microsoft SQL Server, MongoDB are supported by ThinkORM
Redis Adapter	KV store	@nsnake	✗	用于 Redis
适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files

适配器	类型	作者	自动保存	描述
Django ORM Adapter	ORM	Casbin	✓	PostgreSQL、 MariaDB、 MySQL、 Oracle、 SQLite、IBM DB2、 Microsoft SQL Server、 Firebird、 ODBC 都由 Django ORM 支 持
SQLObject Adapter	ORM	Casbin	✓	PostgreSQL、 MySQLite、 Microsoft SQL Server、 Firebird、 Sybase、MAX DB、 pyfirebirdsql 都 由 SQLObject 支持。
SQLAlchemy 适配器	ORM	Casbin	✓	PostgreSQL、 MySQLite、 Oracle、 Microsoft SQL Server、 Firebird、 Sybase 由 SQLAlchemy 支 持
Async SQLAlchemy	ORM	Casbin	✓	PostgreSQL, MySQL, SQLite,

适配器	类型	作者	自动保存	描述
Adapter				Oracle, Microsoft SQL Server, Firebird, Sybase are supported by SQLAlchemy
异步数据库适配器	ORM	Casbin	✓	PostgreSQL、 MySQLite、 Oracle、 Microsoft SQL Server、 Firebird、 Sybase 由 Databases 支 持。
Peewee 适配器	ORM	@shblhy	✓	PostgreSQL、 MySQL、 SQLite 由 Peewee 支持
MongoEngine 适配器	ORM	@zhangbailong945	✗	MongoDB 由 MongoEngine 支持
Pony ORM Adapter	ORM	@drorvinkler	✓	MySQL, PostgreSQL, SQLite, Oracle, CockroachDB 由 Pony ORM 支 持
Tortoise ORM Adapter	ORM	@thearchitector	✓	PostgreSQL (>=9.4)、

适配器	类型	作者	自动保存	描述
				MySQL、 MariaDB 和 SQLite 由 Tortoise ORM 支持
Async Ormar Adapter	ORM	@shepilov-vladislav	✓	PostgreSQL、 MySQL、 SQLite 由 Ormar 支持
SQLModel Adapter	ORM	@shepilov-vladislav	✓	PostgreSQL、 MySQL、 SQLite 由 SQLModel
Couchbase Adapter	NoSQL	ScienceLogic	✓ (没有 <code>remove_filtered_policy()</code>)	用于 Couchbase
DynamoDB Adapter	NoSQL	@abqadeer	✓	用于 DynamoDB
Pymongo 适配器	NoSQL	Casbin	✗	MongoDB 由 [Pymongo 支持](https://pypi.org/project/pymongo/)
Redis Adapter	KV store	Casbin	✓	For Redis
GCP Firebase Adapter	Cloud	@devrushi41	✓	用于 Google Cloud Platform Firebase

适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
EF 适配器	ORM	Casbin	✗	MySQL、PostgreSQL、SQLite、Microsoft SQL Server、Oracle, DB2 等都由 Entity Framework 6 支持。
EFCore 适配器	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server、Oracle、DB2, 等都由 Entity Framework Core 支持
Linq2DB Adapter	ORM	@Tirael	✓	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, Access, Firebird, Sybase, etc. are supported by linq2db
Azure Cosmos DB Adapter	Cloud	@sagarkhandelwal	✓	For Microsoft Azure Cosmos DB
适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Diesel 适配器	ORM	Casbin	✓	SQLite, PostgreSQL, MySQL 由 Diesel 支持
Sqlx	ORM	Casbin	✓	PostgreSQL, MySQL 由 Sqlx 支持, 可实现完

适配器	类型	作者	自动保存	描述
Adapter				全异步操作
SeaORM Adapter	ORM	@lingdu1234	✓	PostgreSQL, MySQL, SQLite are supported by SeaORM with fully asynchronous operation
SeaORM Adapter	ORM	@ZihanType	✓	PostgreSQL, MySQL, SQLite are supported by SeaORM with fully asynchronous operation
Rbatis Adapter	ORM	rbatis	✓	MySQL, PostgreSQL, SQLite, SQL Server, MariaDB, TiDB, CockroachDB, Oracle are supported by Rbatis
DynamodDB Adapter	NoSQL	@fospitia	✓	用于Amazon DynamoDB
MongoDB Adapter	MongoDB	@wangjun861205	✓	For MongoDB
JSON Adapter	String	Casbin	✓	用于JSON
YAML 适配器	String	Casbin	✓	用于 [YAML](https://yaml.org/)
适配器	类型	作者	自动保存	描述
File Adapter	File	Casbin	✗	For .CSV (Comma-Separated Values) files

适配器	类型	作者	自动保存	描述
(内置)				
Sequelize Adapter	ORM	CasbinRuby	✓	ADO, Amalgalite, IBM_DB, JDBC, MySQL, Mysql2, ODBC, Oracle, PostgreSQL, SQLAnywhere SQLite3, 和 TinyTDS 都由 Sequel 支持
适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Memory Adapter (内置)	内存	Casbin	✗	用于内存
Fluent Adapter	ORM	Casbin	✓	PostgreSQL、SQLite、MySQL、MongoDB 由 Fluent 支持
适配器	类型	作者	自动保存	描述
File Adapter (内置)	File	Casbin	✗	For .CSV (Comma-Separated Values) files
Filtered File Adapter (内置)	File	Casbin	✗	对于 CSV (逗号分隔的值) 个带策略子集加载支持的文件
LuaSQL Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite3 are supported by LuaSQL
4DaysORM Adapter	ORM	Casbin	✓	MySQL, SQLite3 由 4DaysORM 支持
OpenResty	ORM	@tom2nonames	✓	MySQL, PostgreSQL are supported by it

适配器	类型	作者	自动保存	描述
Adapter				

① 备注

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

我们为您提供以下示例作为参考:

文件适配器 (内置)

下面的代码演示了如何从File adapter初始化enforcer:

Go PHP Rust

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

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

use Casbin\Enforcer;

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

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](#) [PHP](#) [Rust](#)

```
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)

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

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

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](#) [Rust](#) [PHP](#)

```
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)

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

```
// 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_sqllsr,
    // mysqli,sqlanywhere,sqllsr,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

您可以参考如下代码来使用自建的 adapter

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

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

在不同适配器之间转移/转换

如果您想将适配器从 A 转换为 B, 您可以这样做:

1. 从 A 加载策略到内存

```
e, _ := NewEnforcer(m, A)
```

或者

```
e.SetAdapter(A)  
e.LoadPolicy()
```

2. 将您的适配器从 A 转换为 B

```
e.SetAdapter(B)
```

3. 将策略从内存保存到 B

```
e.LoadPolicy()
```

在运行时进行加载或保存配置信息

如果您在初始化后仍想重载model和policy的配置（或是保存policy的配置信息），那么可以参照如下方法：

```
// 从CONF配置文件中加载model  
e.LoadModel()  
// 从文件或数据库中加载policy  
e.LoadPolicy()  
// 保存当前的policy（通常在调用Casbin API改变了配置信息后）至文件或数据库  
e.SavePolicy()
```

自动保存

自动保存机制（Auto-Save）是adapter的特性之一。支持自动保存机制的adapter可以自动向存储回写内存中单个policy规则的变更（删除/更新）。与自动回写机制不同，调用SavePolicy()会直接删除所有存储中的policy规则并将当前Casbin enforcer存储在内存中的policy规则悉数持久化到存储中。因此，当内存中的policy规则过多时，直接调用SavePolicy()会引起一些性能问题。

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）。

ⓘ 备注

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. 查看上述adapter列表中的 `AutoSave`列，查看adapter是否支持 `Auto-Save`。

以下示例演示了 `Auto-Save` 的使用方法：

```
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:mysql_password@tcp(127.0.0.1:3306)/")
e := casbin.NewEnforcer("examples/basic_model.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(...)

// 启用自动保存选项。
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(...)
```

For more examples, please see: 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。

接口名	类型	描述
LoadPolicy()	必须	从持久层中加载policy规则
SavePolicy()	必须	将policy规则保存至持久层
AddPolicy()	可选	添加单条policy规则至持久层
RemovePolicy()	optional	从持久层删除单条policy规则
RemoveFilteredPolicy()	optional	从持久层删除符合筛选条件的policy规则

(i) 备注

如果适配器不支持 `自动保存`, 它应该为以下三个可选函数提供一个空实现。下面是Golang的一个例子:

```
// AddPolicy 添加一个policy规则至持久层
func (a *Adapter) AddPolicy(sec string, ptype string, rule []string) error {
    return errors.New("not implemented")
}

// RemovePolicy 从持久层删除单条policy规则
func (a *Adapter) RemovePolicy(sec string, ptype string, rule []string) error {
    return errors.New("not implemented")
}

// RemoveFilteredPolicy 从持久层删除符合筛选条件的policy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` 异常。

关于如何写入适配器的详细信息。

- 数据结构 适配器应 **最少** 支持六列。
- 数据库名称: 默认数据库名称应该是 `casbin`。
- 表格名称 默认表名应该是 `casbin_rule`。

- Ptype 栏。此列的名称应该是 `ptype` 而不是 `p_type` 或 `Ptype`。
- 表定义应该是 `(id int priorkey, ptype varchar, v0 varchar, v1 varchar, v2 varchar, v3 varchar, v4 varchar, v5 varchar)`
- 唯一的密钥索引应该建立在列 `ptype, v0, v1, v2, v3, v4, v5` 上。
- `LoadFilteredPolicy` 需要一个 `filter` 作为参数。Filter应该像这样。

```
{
  "p": [ [ "alice" ], [ "bob" ] ],
  "g": [ [ "", "book_group" ], [ "", "pen_group" ] ],
  "g2": [ [ "alice" ] ]
}
```

谁负责创建数据库？

我们通常约定，如果相应的数据库结构尚未建立，那么使用adapter作为policy的持久化工具时，它应具有自动创建一个名为 `casbin` 的数据库的能力。Please use the Xorm adapter as a reference implementation:

<https://github.com/casbin/xorm-adapter>

Context Adapter

[ContextAdapter](#) provides a context-aware interface for Casbin adapters.

Through context, you can implement features such as timeout control for the Adapter API

Example

[gormadapter](#) supports adapter with context, the following is a timeout control implemented using context

```
ca, _ := NewContextAdapter("mysql", "root:@tcp(127.0.0.1:3306)/", "casbin")
// Limited time 300s
ctx, cancel := context.WithTimeout(context.Background(), 300*time.Microsecond)
defer cancel()

err := ca.AddPolicyCtx(ctx, "p", "p", []string{"alice", "data1", "read"})
if err != nil {
    panic(err)
}
```

How to write an context adapter

`ContextAdapter` API only has an extra layer of context processing than ordinary `Adapter` API, and on the basis of implementing ordinary Adapter API, you can encapsulate your own processing logic for context

A simple reference to the `gormadapter`: [context_adapter.go](#)

监视器

We support the use of distributed messaging systems like [etcd](#) to maintain consistency between multiple Casbin enforcer instances. This allows our users to concurrently use multiple Casbin enforcers to handle a large number of permission checking requests.

Similar to policy storage adapters, we do not include watcher code in the main library. 任何对新消息系统的支持都应该作为watcher程序来实现。 A complete list of Casbin watchers is provided below. We welcome any third-party contributions for a new watcher, please inform us and we will add it to this list:)

[Go](#) [Java](#) [Node.js](#) [Python](#) [.NET](#) [Ruby](#) [PHP](#)

Watcher	类型	作者	描述
PostgreSQL WatcherEx	Database	@IguteChung	WatcherEx for PostgreSQL
Redis WatcherEx	KV store	Casbin	WatcherEx for Redis
Redis Watcher	KV store	@billcobbler	Redis 的监视器
Etcd Watcher	KV store	Casbin	etcd 的监视器
TiKV 监视器	KV store	Casbin	TiKV 的监视器
Kafka 监视器	Messaging system	@wgarunap	Apache Kafka的监视器
NATS Watcher	Messaging system	Soluto	NATS的监视器

Watcher	类型	作者	描述
ZooKeeper Watcher	Messaging system	Grepstr	Apache ZooKeeper的监视器
NATS, RabbitMQ, GCP Pub/Sub, AWS SNS & SQS, Kafka, InMemory	Messaging System	@rusenask	监视器基于与领先的云供应商和自托管基础设施运作的 Go Cloud Dev Kit
RocketMQ 观察器	Messaging system	@fmyxyz	Apache RocketMQ的监视器
Watcher	类型	作者	描述
Etcd Adapter	KV store	@mapleafgo	etcd的监视器
Redis Watcher	KV store	Casbin	Redis的监视器
Lettuce-Based Redis Watcher	KV store	Casbin	Watcher for Redis based on Lettuce)
Kafka 监视器	Messaging system	Casbin	Apache Kafka的监视器
Watcher	实现要素	作者	描述
Etcd Watcher	KV store	Casbin	etcd的监视器
Redis Watcher	KV store	Casbin	Redis的监视器
Pub/Sub Watcher	Messaging system	Casbin	Google Cloud Pub/Sub的监视器
MongoDB Change	数据库	Casbin	Watcher for MongoDB

Watcher	实现要素	作者	描述
Streams Watcher			Change Streams
Postgres Watcher	Database	Matteo Collina	PostgreSQL的监视器
Watcher	类型	作者	描述
Etc Watcher	KV store	Casbin	etcd的监视器
Redis Watcher	KV store	Casbin	Redis的监视器
Redis Watcher	KV store	ScienceLogic	Redis的监视器
PostgreSQL Watcher	Database	Casbin	PostgreSQL的监视器
RabbitMQ 监视器	Messaging system	Casbin	RabbitMQ 的监视器
Watcher	类型	作者	描述
Redis Watcher	KV store	@Sbou	Redis的监视器
Watcher	类型	作者	描述
Redis Watcher	KV store	CasbinRuby	Redis的监视器
RabbitMQ Watcher	Messaging system	CasbinRuby	Watcher for RabbitMQ
Watcher	类型	作者	描述
Redis Watcher	KV store	@Tinywan	Redis的监视器

WatcherEx

为了支持多个实例之间的增量同步，我们提供 `WatcherEx` 接口。我们希望它能够在策略改变时通知其他实例，但目前尚未实现 `WatcherEx`。我们推荐您使用调度器来实现它。

与 `Watcher` 接口相比，`WatcherEx` 可以区分收到的更新动作类型。`添加策略` 和 `删除策略`。

WatcherEx 的调用接口：

应用程序接口	描述
<code>SetUpdateCallback(func(string))</code> <code>error</code>	<code>SetUpdateCallback</code> sets the callback function that the watcher will call, when the policy in DB has been changed by other instances. 一种典型的回调是 <code>Enforcer.LoadPolicy()</code> 。
<code>Update()</code> <code>error</code>	<code>Update</code> calls the update callback of other instances to synchronize their policy. 它通常是在改变数据库中的策略之后进行的，如 <code>Enforcer.SavePolicy()</code> , <code>Enforcer.AddPolicy()</code> , <code>Enforcer.RemovePolicy()</code> 等。
<code>Close()</code>	<code>Close</code> stops and releases the watcher, the callback function will not be called any more.
<code>UpdateForAddPolicy(sec, ptype string, params ...string)</code> <code>error</code>	<code>UpdateForAddPolicy</code> calls the update callback of other instances to synchronize their policy. 它是在 <code>Enforcer.AddPolicy()</code> , <code>Enforcer.AddNamedPolicy()</code> , <code>Enforcer.AddGroupingPolicy()</code> 和 <code>Enforcer.AddNamedGroupingPolicy()</code> 后被调用。
<code>UpdateForRemovePolicy(sec, ptype string, params ...string)</code> <code>error</code>	<code>UpdateForRemovePolicy</code> calls the update callback of other instances to synchronize their

应用程序接口	描述
	policy. 它是在政策被Enforcer.RemovePolicy(), Enforcer.RemoveNamedPolicy(), Enforcer.RemoveGroupingPolicy() 和 Enforcer.RemoveNamedGroupingPolicy() 删除后被调用。
UpdateForRemoveFilteredPolicy(sec, ptype string, fieldIndex int, fieldValues ...string) error	UpdateForRemoveFilteredPolicy calls the update callback of other instances to synchronize their policy. 它是在 Enforcer.RemoveFilteredPolicy(), Enforcer.RemoveFilteredNamedPolicy(), Enforcer.RemoveFilteredGroupingPolicy() 和 Enforcer.RemoveFilteredNamedGroupingPolicy() 之后被调用
UpdateForSavePolicy(model model.Model) error	UpdateForSavePolicy calls the update callback of other instances to synchronize their policy. 它在 Enforcer.AddPolicy() 被调用后调用
UpdateForAddPolicies(sec string, ptype string, rules ...[]string) error	UpdateForAddPolicies calls the update callback of other instances to synchronize their policy. 它是在Enforcer.AddPolicies(), Enforcecer.AddNamedPolicies(), Enforcer.AddGroupingPolicies() 和 Enforcer.AddNamedGroupingPolicies() 之后被调用
UpdateForRemovePolicies(sec string, ptype string, rules ...[]string) error	UpdateForRemovePolicies calls the update callback of other instances to synchronize their policy. 它是在Enforcer.RemovePolicies(), Enforcecer.RemoveNamedPolicies(), Enforcer.RemoveGroupingPolicies() 和 Enforcer.RemoveNamedGroupingPolicies() 后被

应用程序接口	描述
	调用。

调度器

Dispatchers provide a way to synchronize incremental changes of policy. They should be based on consistency algorithms such as Raft to ensure the consistency of all enforcer instances. Through dispatchers, users can easily establish distributed clusters.

调度器的方法分为两部分。The first part is the method combined with Casbin. These methods should be called inside Casbin. Users can use the more complete API provided by Casbin itself.

The other part is the method defined by the dispatcher itself, including the dispatcher initialization method, and different functions provided by different algorithms, such as dynamic membership and config changes.

① 备注

We hope dispatchers only ensure the consistency of the Casbin enforcer at runtime. So if the policy is inconsistent during initialization, the dispatchers will not work properly. Users need to ensure that the state of all instances is consistent before using dispatchers.

A complete list of Casbin dispatchers is provided below. Any 3rd-party contributions on a new dispatcher are welcomed. Please inform us, and we will add it to this list.

[Go](#)

调度器	类型	作者	描述
Hashicorp Raft Dispatcher	Raft	Casbin	A dispatcher based on Hashicorp Raft
KDKYG/casbin-dispatcher	Raft	@KDKYG	A dispatcher based on Hashicorp Raft

分布式执行

DistributedEnforcer wraps SyncedEnforcer for the dispatcher.

Go

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

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）。A role manager can retrieve role data from Casbin policy rules or external sources such as LDAP, Okta, Auth0, Azure AD, etc. 我们支持不同的角色管理器实现。To keep the lightweight, we don't include role manager code in the main library (except the default role manager). A complete list of Casbin role managers is provided below. Any third-party contributions for a new role manager are welcome. Please inform us, and we will add it to this list:)

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#)

角色管理器	作者	描述
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy
Session Role Manager	EDOMO Systems	Supports role hierarchy stored in the Casbin policy, with time-range-based sessions
Okta Role Manager	Casbin	支持存储在Okta中的角色层次结构
Auth0 Role Manager	Casbin	支持存储在Auth0's Authorization Extension 授权扩展名中的角色层次结构

对于开发人员：所有角色管理器都必须实现 [RoleManager](#) 接口。The [Session Role Manager](#) can be used as a reference implementation.

角色管理器	作者	描述
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

对于开发人员：所有角色管理器都必须实现 [RoleManager](#) 接口。The [Default Role Manager](#) can be used as a reference implementation.

角色管理器	作者	描述
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy
Session Role Manager	Casbin	Supports role hierarchy stored in the Casbin policy, with time-range-based sessions

对于开发人员：所有角色管理器都必须实现 [RoleManager](#) 接口。The [Default Role Manager](#) can be used as a reference implementation.

角色管理器	作者	描述
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

对于开发人员：所有角色管理器都必须实现 [RoleManager](#) 接口。The [Default Role Manager](#) can be used as a reference implementation.

角色管理器	作者	描述
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

对于开发人员：所有角色管理器都必须实现 [RoleManager](#) 接口。The [Default Role Manager](#) can be used as a reference implementation.

API

See the [API](#) section for details.

中间件

Web frameworks

[Go](#) [Java](#) [Node.js](#) [PHP](#) [Python](#) [C++](#) [.NET](#) [Rust](#) [Lua](#)

[Swift](#)

名称	描述
Gin	一个有着更好性能的 HTTP 网络框架，支持类似于 Martini 的 API，通过以下插件实现： authz 或 gin-casbin
Beego	一个 Go 语言的开源、高性能网络框架，通过以下插件实现： plugins/authz
Caddy	快速、跨平台的有自动HTTPS的HTTP/2 web服务器，通过插件： caddy-authz 实现。
Traefik	云端本地应用程序代理，通过插件： Traefik-auth插件 实现
Kratos	Your ultimate Go microservices framework for the cloud-native era, via plugin: tx7do/kratos-casbin or overstarry/kratos-casbin
Go kit	一个用于微服务的工具包，通过内置插件： plugins/authz 实现。

名称	描述
Fiber	An Express inspired web framework written in Go, via middleware: casbin in gofiber/contrib or fiber-casbinrest or fiber-boilerplate or gofiber-casbin
Revel	一个用Go语言编制的高效、全栈的web框架，通过插件： auth/casbin 实现。
Echo	High performance, minimalist Go web framework, via plugin: echo-authz or echo-casbin or casbinrest or echo-boilerplate
Iris	(这个) 地球上用Go语言编写的最快的web框架。 HTTP/2 Ready-To-GO, via plugin: casbin or iris-middleware-casbin
GoFrame	模块化的，强力的，高性能的和企业级的Golang的应用开发框架，通过插件 gf-casbin 实现.
Negroni	Golang的惯用HTTP中间件，通过插件： negroni-authz 实现
Chi	一个用于构建 HTTP 服务的轻量级的、常用的和可组合的路由器, 通过插件: chi-authz 实现
Buffalo	基于Go的网络开发生态，致力于让你的生活更简单，通过插件: buffalo-mw-rbac 实现
Macaron	一个使用Go语言实现的高产能、模块化的网络框架，通过插件： authz 实现
DotWeb	简易的Go网络微框架，通过插件： authz 实现

名称	描述
Tango	微型 & 插拔式的Go网络框架, 通过插件 authz 实现
Baa	一个带有路由, 中间件, 依赖注入和http context 的express Go网络框架, 通过插件 authz 实现
Tyk	一个开源企业API网关, 通过插件支持REST、GraphQL、TCP和gRPC 协议: tyk-authz
Hertz	Go HTTP framework with high-performance and strong-extensibility for building micro-services, via plugin: casbin
名称	描述
Spring Boot	Makes it easy to create Spring-powered applications and services, via plugin: casbin-spring-boot-starter or Simple SpringBoot security demo with jCasbin
Apache Shiro	一个强大且易于使用的 Java 安全框架, 通过插件进行身份验证、授权、加密和会话管理, 通过插件: shirro-casbin 或 shiro-jcasbin-spring-boot-starter 实现
JFinal	一个简单、轻量、迅速、独立、可扩展的Java WEB + ORM 框架, 通过插件: jcasbin-jfinal-plugin 实现
Nutz	适合所有所有Java开发者的WEB框架 (MVC/IOC/AOP/DAO/JSON), 通过插件: jcasbin-nutz-plugin 实现
mangoo	一个直观, 轻量、高性能、全栈Java Web框架通过内置插件

名称	描述
I/O	AuthorizationService.java
名称	描述
Shield	一个建于 casbin 顶部的 authZ 服务器和 authZ 认知反向代理。
Express	用于node的快速，无意的，最小化的网络框架，通过插件 express-authz 实现
Koa	用于Node.JS的高表达性的中间件使用ES2017异步函数，通过 koa-authz 或者 koajs-starter 或 koa-casbin 插件实现
LoopBack 4	一个高扩展性的 Node.js 和 TypeScript 框架，通过插件构建API和微型服务： loopback4-authorization
Nest	使用TypeScript和JavaScript构建高效和可伸缩的服务器端应用程序的先进Node.js框架 通过插件： nest-authz or nest-casbin 或 NestJS Casbin 模块 或 nestjs-casbin 或 shanbe-api 或 acl-nest 或 nestjs-casbin-typeorm 实现
Fastify	用于Node.js的快速和低开销网页框架： fastify-casbin or fastify-casbin-rest
Egg	用于使用Node.Js & Koa更好地来构建企业框架和应用，通过 egg-authz 或者 egg-zrole 插件实现
hapi	简单、安全、值的开发者信赖的框架。 通过插件： hapi-authz 实现

名称	描述
Casbin JWT Express	使用无状态JWT token来使Casbin ACL(访问控制列表) 有效的授权中间件
名称	描述
Laravel	为网络工程师设计的PHP框架，通过插件 laravel-casbin 实现
Yii PHP Framework	快速、安全和高效的PHP框架，通过插件： yii-permission or yii-casbin
CakePHP	创建快速的稳定的PHP框架，通过 cake-casbin 插件实现
CodeIgniter	在 CodeIgniter4 网页框架中通过插件将具有角色和权限的用户联系起来，通过插件： CodeIgniter Permission 实现
ThinkPHP 5.1	ThinkPHP 5.1框架，通过插件 think-casbin 实现
ThinkPHP 6.0	ThinkPHP 6.0框架，通过 think-authz 插件实现
Symfony	Symfony PHP框架，通过插件： symfony-permission 或者 symfony-casbin
Hyperf	A coroutine framework that focuses on hyperspeed and flexibility, via plugin: hyperf-permission or donjan-deng/hyperf-casbin or cblink/hyperf-casbin

名称	描述
EasySwoole	基于 Swoole 扩展实现的一个分布式的，持续存储的 PHP 框架，通过插件: easyswoole-permission 或者 easyswoole-hyperfOrm-permission 实现
Slim	一个 PHP 微型框架，通过插件帮助您快速实现简单但强大的 Web 应用程序和API，通过插件: casbin-with-slim 实现
Phalcon	以C-扩展形式发送的全堆栈PHP框架，通过插件: phalcon-permission 实现
Webman	High performance HTTP Service Framework for PHP based on Workerman, via plugin: webman-permission or webman-casbin
名称	描述
Django	高级 Python Web 框架，通过插件: django-casbin 或 django-authority 即可实现
Flask	基于 Werkzeug 和 Jinja 的 Python 微型框架，通过插件 flask-authz , Flask-Casbin (第三方实现，或许更好) 或者 rbac-flask 。
FastAPI	使用 Python 3.6+ API 的现代、快速(高性能)、网页框架，基于标准 Python 类型提示，通过插件: fastapi-authz 或者 Fastapi-app 实现
OpenStack	部署最广泛的开放源码云软件，通过插件: openstack-patron 实现

名称	描述
Nginx	A HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, via plugin: nginx-casbin-module
名称	描述
ASP.NET Core	一个开放源码和跨平台框架，用于建立以云为基础的现代互联网连接应用。例如Web应用、IoT应用和移动后端，通过插件： Casbin.AspNetCore
ASP.NET Core	通过插件在ASP.NET核心框架中使用Casbin的简单演示： CasbinACL-aspNetCore
名称	描述
Actix	Rust 框架，通过插件 actix-casbin 实现
Actix web	一个小型、务实和快速的rust网络框架，通过插件： actix-casbin-auth 实现
Rocket	a web framework for Rust that makes it simple to write fast, secure web applications without sacrificing flexibility, usability, or type safety, via plugin: rocket-authz or rocket-casbin-auth
Axum 网络	一个符合人体工程学和模块化的rust网络框架，通过插件： axum-casbin-auth 实现
Poem 网页	通过插件 poem-casbin 便可使用Rust 编程语言的 web 框架，其功能齐全、易于操作。

名称	描述
OpenResty	基于 NGINX 和 LuaJIT 的动态网络平台，通过插件： lua-resty-casbin 和 casbin-openresty-example 实现
Kong	一个云原生、可伸展的 API 网关，有着高性能和扩展性能，通过插件： kong-authz 实现
APISIX	一个动态的、实时的高性能API网关，通过插件： apisix-authz 实现
名称	描述
Vapor	服务器端的Swift web 框架，通过插件： vapor-authz 实现

Cloud providers

Node.js

Name	Description
Okta	一个可信的平台通过插件保护身份： casbin-spring-boot-demo
Auth0	一个实现简单、可适应的认证和授权平台，通过插件： casbin-auth0-rbac 实现

Graphql-中间件

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 other words, Casbin should be placed between the GraphQL layer and your business logic.

```
// Casbin授权逻辑位于postRepository内
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 below. Any third-party contributions on a new GraphQL middleware are welcomed. Please inform us, and we will add it to this list:

中间件	GraphQL 实现	作者	描述
graphql-authz	graphql	Casbin	用于graphql-go的授权中间件
graphql-casbin	graphql	@esmaeilpour	An implementation of using Graphql and Casbin together
gqlgen_casbin_RBAC_example	gqlgen	@WenyXu	(empty)
中间件	GraphQL 实现	作者	描述
graphql-authz	GraphQL.js	Casbin	一个用于 GraphQL.js 的授权中间件
中间件	GraphQL 实现	作者	描述
graphql-authz	GraphQL-core 3	@Checho3388	一个用于 GraphQL.js 的授权中间件

云端原生中间件

Cloud Native Projects

[Go](#) [Node.js](#)

Project	作者	描述
k8s-authz	Casbin	Authorization middleware for Kubernetes
envoy-authz	Casbin	Authorization middleware for Istio and Envoy
kubesphere-authz	Casbin	Authorization middleware for kubeSphere

Project	作者	描述
ODPF Shield	Open Data Platform	ODPF Shield is a cloud native role-based authorization-aware reverse-proxy service.

API

API概述

Casbin API Usage

管理 API

原始的 API 为 Casbin 策略管理提供了充分支持

RBAC API

用于RBAC的更友好的 API 此 API 是管理 API 中的一个子集。 RBAC 用户可以使用此 API 来简化代码。

域内基于角色的访问控制 API

A more user-friendly API for RBAC with domains. This API is a subset of the Management API. RBAC users can use this API to simplify their code.

RBAC with Conditions API

A more user-friendly API for RBAC with conditions.

角色管理器API

The RoleManager API provides an interface for defining operations to manage roles. The addition of a matching function to the RoleManager allows the use of wildcards...

数据权限

Solutions for Data Permissions

API概述

This overview only shows you how to use Casbin APIs and doesn't explain how Casbin is installed or how it works. You can find those tutorials here: [Installation of Casbin](#) and [How Casbin Works](#). 因此，当你开始阅读本教程时，我们假设你已经完全安装并将Casbin导入你的代码中。

用于强制执行的API(Enforce API)

Let's start with the Enforce APIs of Casbin. We will load a RBAC model from `model.conf` and load policies from `policy.csv`. You can learn about the Model syntax [here](#), and we won't discuss it in this tutorial. 我们假设你能理解下面给出的配置文件。

`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]
```

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
```

阅读完配置文件后，请阅读以下代码：

```
// 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")
```

这段代码从本地文件加载访问控制模型和策略。The function

`casbin.NewEnforcer()` will return an enforcer. It will recognize its two parameters as file paths and load the files from there. Errors occurred in the process are stored in the variable `err`. This code uses the default adapter to load the model and policies, and of course, you can achieve the same result by using a third-party adapter.

The code `ok, err := enforcer.Enforce("alice", "data1", "read")` is used to confirm access permissions. If Alice can access data1 with the read operation, the returned value of `ok` will be `true`; otherwise, it will be `false`. 在这个例子中, `ok` 的值是 `true`。

EnforceEx API

Sometimes you may wonder which policy allowed the request, so we have prepared the function `EnforceEx()`. 你可以像这样使用它：

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

The `EnforceEx()` function will return the exact policy string in the return value `reason`. In this example, `amber` is an `admin` role, so the policy `p, admin, data1, read` allowed this request to be `true`. The output of this code is in the comment.

Casbin has provided many APIs similar to this one. These APIs add some extra functions to the basic ones. They include:

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

This function uses a matcher.

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

This is a combination of `EnforceWithMatcher()` and `EnforceEx()`.

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

This function allows for a list of jobs and returns an array.

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

following paragraphs.

管理 API

Get API

These APIs are used to retrieve specific objects in policies. In this example, we are loading an enforcer and retrieving something from it.

Please take a look at 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)
```

Similar to the previous example, the first four lines are used to load necessary information from local files. We won't discuss that here any further.

The code `allSubjects := enforcer.GetAllSubjects()` retrieves all the subjects in the policy file and returns them as an array. We then print that array.

Typically, 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 appear in the current

named policy.

Similarly, we have prepared `GetAll` functions for `Objects`, `Actions`, `Roles`. To access these functions, you simply need to replace the word `Subject` in the function name with the desired category.

Additionally, there are more getters available for policies. The method of calling and the return values are similar to the ones mentioned above.

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

添加, 删除, 更新 API

Casbin provides a variety of APIs for dynamically adding, deleting, or modifying

policies at runtime.

The following code demonstrates how to add, remove, and update policies, as well as how to check if 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 and use HasPolicy() to confirm
enforcer.AddPolicy("added_user", "data1", "read")
hasPolicy := enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // true, the policy was added
successfully

// remove a policy and use HasPolicy() to confirm
enforcer.RemovePolicy("alice", "data1", "read")
hasPolicy = enforcer.HasPolicy("alice", "data1", "read")
fmt.Println(hasPolicy) // false, the policy was removed
successfully

// update a policy and use HasPolicy() to confirm
enforcer.UpdatePolicy([]string{"added_user", "data1", "read"}, 
[]string{"added_user", "data1", "write"})
hasPolicy = enforcer.HasPolicy("added_user", "data1", "read")
fmt.Println(hasPolicy) // false, the original policy has expired
hasPolicy = enforcer.HasPolicy("added_user", "data1", "write")
fmt.Println(hasPolicy) // true, the new policy is in effect
```

By using these APIs, you can edit your policies dynamically. Similarly, we have provided similar APIs for `FilteredPolicy`, `NamedPolicy`, `FilteredNamedPolicy`, `GroupingPolicy`, `NamedGroupingPolicy`,

`FilteredGroupingPolicy`, `FilteredNamedGroupingPolicy`. To use them, simply replace the word `Policy` in the function name with the appropriate category.

Furthermore, by changing the parameters to arrays, you can perform batch editing of your policies.

For example, consider functions like this:

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

If we change `Policy` to `Policies` and modify the parameters as follows:

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

then we can perform batch editing of these policies.

The same operations can also be applied to `GroupingPolicy`, `NamedGroupingPolicy`.

AddEx API

Casbin provides the AddEx series of APIs to help users add rules in batches.

```
AddPoliciesEx(rules [][]string) (bool, error)  
AddNamedPoliciesEx(ptype string, rules [][]string) (bool, error)  
AddGroupingPoliciesEx(rules [][]string) (bool, error)  
AddNamedGroupingPoliciesEx(ptype string, rules [][]string)
```

The difference between these methods and the methods without the Ex suffix is that if one of the rules already exists, they will continue checking the next rule instead of returning false immediately.

For example, let's compare `AddPolicies` and `AddPoliciesEx`.

You can run and observe the following code by copying it into the test under casbin.

```
func TestDemo(t *testing.T) {
    e, err := NewEnforcer("examples/basic_model.conf",
"examples/basic_policy.csv")
    if err != nil {
        fmt.Printf("Error, details: %s\n", err)
    }
    e.ClearPolicy()
    e.AddPolicy("user1", "data1", "read")
    fmt.Println(e.GetPolicy())
    testGetPolicy(t, e, [][]string{{"user1", "data1", "read"}})

    // policy {"user1", "data1", "read"} now exists

    // Use AddPolicies to add rules in batches
    ok, _ := e.AddPolicies([][]string{{"user1", "data1",
"read"}, {"user2", "data2", "read"}})
    fmt.Println(e.GetPolicy())
    // {"user2", "data2", "read"} failed to add because
    // {"user1", "data1", "read"} already exists
    // AddPolicies returns false and no other policies are
    // checked, even though they may not exist in the existing ruleset
    // ok == false
    fmt.Println(ok)
    testGetPolicy(t, e, [][]string{{"user1", "data1", "read"}})

    // Use AddPoliciesEx to add rules in batches
```

基于角色的访问控制接口

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

Here, we only show you how to use the RBAC APIs of Casbin and won't talk about RBAC itself. 您可以从[这里](#)获取详情。

We use the following code to load the 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, we can use an instance of the 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()` returns an array that contains all the roles that amber has. In this example, amber has only one role, which is admin, so the array `roles` is `[admin]`. Similarly, you can use `GetUsersForRole()` to get the users who belong to a role. 此函数的返回值也是一个数组。

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

您可以使用 `HasRoleForUser()` 来确认用户是否属于该角色。在这个例子中，`amber` 是管理员的成员，所以这个函数的返回值是 `true`。

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

您可以使用 `DeletePermission()` 来删除权限。

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

然后使用 `DeletePermissionForUser()` 来删除用户的权限。

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

管理 API

提供对Casbin策略管理完全支持的基本API。

筛选的 API

几乎所有的带有过滤器的api有着相同的参数 `(fieldIndex int, fieldValues ...string)`. `field index` 是匹配起始点的索引。 `field value` 表示结果应该有的值。请注意字段值中的空字符串可以是任意单词。

示例:

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

```
e.GetFilteredPolicy(1, "book") // 将返回: [Alice book read] [bob book  
read] [bob book write]]  
  
e.GetFilteredPolicy(1, "book", "read") // 将返回: [[Alice book read]  
[bob book read]]  
  
e.GetFilteredPolicy(0, "Alice", "", "read") // 将返回: [[Alice book  
read]]  
  
e.GetFilteredPolicy(0, "Alice") // 将返回: [Alice book read] [Alice  
penge]]
```

参考

全局变量 `e` 是 Enforcer 实例。

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

```
const e = await newEnforcer('examples/rbac_model.conf', 'examples/
rbac_policy.csv')
```

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

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

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

```
let mut e = Enforce::new("examples/rbac_model.conf", "examples/
rbac_policy.csv").await?;
```

```
Enforcer e = new Enforcer("examples/rbac_model.conf", "examples/
rbac_policy.csv");
```

Enforce()

Enforce决定“主体”是否能够用“动作”操作访问“对象”，输入参数通常是: (sub, obj, act)。

例如:

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

const ok = await e.enforce(request);

$ok = $e->enforcer($request);

ok = e.enforcer(request)

boolean ok = e.enforce(request);
```

EnforceWithMatcher()

EnforceWithMatcher使用自定义匹配器来决定“subject”是否可以通过“action”操作访问“object”，输入参数通常为：(matcher, sub, obj, act)，在匹配器为""时默认使用模型匹配器。

例如：

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

$ok = $e->enforceWithMatcher($matcher, $request);

ok = e.enforce_with_matcher(matcher, request)

boolean ok = e.enforceWithMatcher(matcher, request);
```

EnforceEx()

EnforceEx 通过通知匹配的规则来解释执行

例如:

[Go](#) [Node.js](#) [PHP](#) [Python](#)

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

const ok = await e.enforceEx(request);

list($ok, $reason) = $e->enforceEx($request);

ok, reason = e.enforce_ex(request)
```

EnforceExWithMatcher()

EnforceExWEMatcher 使用自定义匹配器并通过通知匹配的规则来解释强制执行。

例如:

[Go](#)

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

BatchEnforce()

BatchEnforce 强制执行每个请求并返回一个布尔数组的结果

例如:

Go Node.js Java

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

const boolArray = await e.batchEnforce(requests);

List<Boolean> boolArray = e.batchEnforce(requests);
```

GetAllSubjects()

GetAllSubjects 获取当前策略中显示的主题列表。

例如:

Go Node.js PHP Python .NET Rust Java

```
allSubjects := e.GetAllSubjects()

const allSubjects = await e.getAllSubjects()

$allSubjects = $e->getAllSubjects();

all_subjects = e.get_all_subjects()

var allSubjects = e.GetAllSubjects();

let all_subjects = e.get_all_subjects();

List<String> allSubjects = e.getAllSubjects();
```

GetAllNamedSubjects()

GetAllNamedSubjects 获取当前命名策略中显示的主题列表。

例如:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allNamedSubjects := e.GetAllNamedSubjects("p")  
  
const allNamedSubjects = await e.getAllNamedSubjects('p')  
  
$allNamedSubjects = $e->getAllNamedSubjects("p");  
  
all_named_subjects = e.get_all_named_subjects("p")  
  
var allNamedSubjects = e.GetAllNamedSubjects("p");  
  
let all_named_subjects = e.get_all_named_subjects("p");  
  
List<String> allNamedSubjects = e.getAllNamedSubjects("p");
```

GetAllObjects()

GetAllObjects 获取当前策略中显示的对象列表。

例如:

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allObjects := e.GetAllObjects()

const allObjects = await e.getAllObjects()

$allObjects = $e->getAllObjects();

all_objects = e.get_all_objects()

var allObjects = e.GetAllObjects();

let all_objects = e.get_all_objects();

List<String> allObjects = e.getAllObjects();
```

GetAllNamedObjects()

GetAllNamedObjects 获取当前命名策略中显示的对象列表。

例如:

Go Node.js PHP Python .NET Rust Java

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

const allNamedObjects = await e.getAllNamedObjects('p')

$allNamedObjects = $e->getAllNamedObjects("p");

all_named_objects = e.get_all_named_objects("p")

var allNamedObjects = e.GetAllNamedObjects("p");
```

```
let all_named_objects = e.get_all_named_objects("p");

List<String> allNamedObjects = e.getAllNamedObjects("p");
```

GetAllActions()

GetAllActions 获取当前策略中显示的操作列表。

例如：

Go Node.js PHP Python .NET Rust Java

```
allActions := e.GetAllActions()

const allActions = await e.getAllActions()

$allActions = $e->getAllActions();

all_actions = e.get_all_actions()

var allActions = e.GetAllActions();

let all_actions = e.get_all_actions();

List<String> allActions = e.getAllActions();
```

GetAllNamedActions()

GetAllNamedActions 获取当前命名策略中显示的操作列表。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const allNamedActions = await e.getAllNamedActions('p')

$allNamedActions = $e->getAllNamedActions("p");

all_named_actions = e.get_all_named_actions("p")

var allNamedActions = e.GetAllNamedActions("p");

let all_named_actions = e.get_all_named_actions("p");

List<String> allNamedActions = e.getAllNamedActions("p");
```

GetAllRoles()

GetAllRoles 获取当前策略中显示的角色列表。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
allRoles = e.GetAllRoles()

const allRoles = await e.getAllRoles()

$allRoles = $e->getAllRoles();
```

```
all_roles = e.get_all_roles()

var allRoles = e.GetAllRoles();

let all_roles = e.get_all_roles();

List<String> allRoles = e.getAllRoles();
```

GetAllNamedRoles()

GetAllNamedRoles 获取当前命名策略中显示的角色列表。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const allNamedRoles = await e.getAllNamedRoles('g')

$allNamedRoles = $e->getAllNamedRoles('g');

all_named_roles = e.get_all_named_roles("g")

var allNamedRoles = e.GetAllNamedRoles("g");

let all_named_roles = e.get_all_named_roles("g");

List<String> allNamedRoles = e.getAllNamedRoles("g");
```

GetPolicy()

GetPolicy 获取策略中的所有授权规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
policy = e.GetPolicy()

const policy = await e.getPolicy()

$policy = $e->getPolicy();

policy = e.get_policy()

var policy = e.GetPolicy();

let policy = e.get_policy();

List<List<String>> policy = e.getPolicy();
```

GetFilteredPolicy()

GetFilteredPolicy 获取策略中的所有授权规则，可以指定字段筛选器。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const filteredPolicy = await e.getFilteredPolicy(0, 'alice')

$filteredPolicy = $e->getFilteredPolicy(0, "alice");

filtered_policy = e.get_filtered_policy(0, "alice")

var filteredPolicy = e.GetFilteredPolicy(0, "alice");

let filtered_policy = e.get_filtered_policy(0,
    vec!["alice".to_owned()]);

List<List<String>> filteredPolicy = e.getFilteredPolicy(0, "alice");
```

GetNamedPolicy()

GetNamedPolicy 获取命名策略中的所有授权规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const namedPolicy = await e.getNamedPolicy('p')

$namedPolicy = $e->getNamedPolicy("p");

named_policy = e.get_named_policy("p")
```

```
var namedPolicy = e.GetNamedPolicy("p");

let named_policy = e.get_named_policy("p");

List<List<String>> namedPolicy = e.getNamedPolicy("p");
```

GetFilteredNamedPolicy()

GetFilteredNamedPolicy 获取命名策略中的所有授权规则，可以指定字段过滤器。

例如：

Go Node.js PHP Python .NET Rust Java

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

const filteredNamedPolicy = await e.getFilteredNamedPolicy('p', 0,
  'bob')

$filteredNamedPolicy = $e->getFilteredNamedPolicy("p", 0, "bob");

filtered_named_policy = e.get_filtered_named_policy("p", 0, "alice")

var filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "alice");

let filtered_named_policy = e.get_filtered_named_policy("p", 0,
  vec!["bob".to_owned()]);

List<List<String>> filteredNamedPolicy = e.getFilteredNamedPolicy("p",
  0, "bob");
```

GetGroupingPolicy()

GetGroupingPolicy 获取策略中的所有角色继承规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
groupingPolicy := e.GetGroupingPolicy()

const groupingPolicy = await e.getGroupingPolicy()

$groupingPolicy = $e->getGroupingPolicy();

grouping_policy = e.get_grouping_policy()

var groupingPolicy = e.GetGroupingPolicy();

let grouping_policy = e.get_grouping_policy();

List<List<String>> groupingPolicy = e.getGroupingPolicy();
```

GetFilteredGroupingPolicy()

GetFilteredGroupingPolicy 获取策略中的所有角色继承规则，可以指定字段筛选器。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const filteredGroupingPolicy = await e.getFilteredGroupingPolicy(0,
  'alice')

$filteredGroupingPolicy = $e->getFilteredGroupingPolicy(0, "alice");

filtered_grouping_policy = e.get_filtered_grouping_policy(0, "alice")

var filteredGroupingPolicy = e.GetFilteredGroupingPolicy(0, "alice");

let filtered_grouping_policy = e.get_filtered_grouping_policy(0,
  vec!["alice".to_owned()]);
}

List<List<String>> filteredGroupingPolicy =
e.getFilteredGroupingPolicy(0, "alice");
```

GetNamedGroupingPolicy()

GetNamedGroupingPolicy 获取策略中的所有角色继承规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const namedGroupingPolicy = await e.getNamedGroupingPolicy('g')

$namedGroupingPolicy = $e->getNamedGroupingPolicy("g");
```

```
named_grouping_policy = e.get_named_grouping_policy("g")

var namedGroupingPolicy = e.GetNamedGroupingPolicy("g");

let named_grouping_policy = e.get_named_grouping_policy("g");

List<List<String>> namedGroupingPolicy = e.getNamedGroupingPolicy("g");
```

GetFilteredNamedGroupingPolicy()

GetFilteredNamedGroupingPolicy 获取策略中的所有角色继承规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const namedGroupingPolicy = await
e.getFilteredNamedGroupingPolicy('g', 0, 'alice')

$namedGroupingPolicy = $e->getFilteredNamedGroupingPolicy("g", 0,
"alice");

named_grouping_policy = e.get_filtered_named_grouping_policy("g", 0,
"alice")

var namedGroupingPolicy = e.GetFilteredNamedGroupingPolicy("g", 0,
"alice");

let named_grouping_policy = e.get_filtered_named_groupingPolicy("g",
```

```
List<List<String>> filteredNamedGroupingPolicy =  
e.getFilteredNamedGroupingPolicy("g", 0, "alice");
```

HasPolicy()

HasPolicy 确定是否存在授权规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
hasPolicy := e.HasPolicy("data2_admin", "data2", "read")  
  
const hasPolicy = await e.hasPolicy('data2_admin', 'data2', 'read')  
  
$hasPolicy = $e->hasPolicy('data2_admin', 'data2', 'read');  
  
has_policy = e.has_policy("data2_admin", "data2", "read")  
  
var hasPolicy = e.HasPolicy("data2_admin", "data2", "read");  
  
let has_policy = e.has_policy(vec!["data2_admin".to_owned(),  
"data2".to_owned(), "read".to_owned()]);  
  
boolean hasPolicy = e.hasPolicy("data2_admin", "data2", "read");
```

HasNamedPolicy()

HasNamedPolicy 确定是否存在命名授权规则。

例如：

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

const hasNamedPolicy = await e.hasNamedPolicy('p', 'data2_admin',
    'data2', 'read')

$hasNamedPolicy = $e->hasNamedPolicy("p", "data2_admin", "data2",
    "read");

has_named_policy = e.has_named_policy("p", "data2_admin", "data2",
    "read");

var hasNamedPolicy = e.HasNamedPolicy("p", "data2_admin", "data2",
    "read");

let has_named_policy = e.has_named_policy("p",
    vec![ "data2_admin".to_owned(), "data2".to_owned(), "read".to_owned() ]);

boolean hasNamedPolicy = e.hasNamedPolicy("p", "data2_admin", "data2",
    "read");
```

AddPolicy()

AddPolicy 向当前策略添加授权规则。如果规则已经存在，函数返回false，并且不会添加规则。否则，函数通过添加新规则并返回true。

例如：

```

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

const p = ['eve', 'data3', 'read']
const added = await e.addPolicy(...p)

$added = $e->addPolicy('eve', 'data3', 'read');

added = e.add_policy("eve", "data3", "read")

var added = e.AddPolicy("eve", "data3", "read");
or
var added = await e.AddPolicyAsync("eve", "data3", "read");

let added = e.add_policy(vec!["eve".to_owned(), "data3".to_owned(),
"read".to_owned()]);

```

boolean added = e.addPolicy("eve", "data3", "read");

AddPolicies()

AddPolicy 向当前策略添加授权许多规则。该操作本质上是原子的 因此，如果授权规则由不符合现行政策的规则组成，函数返回false，当前政策中没有添加任何政策规则。如果所有授权规则都符合政策规则，则函数返回true，每项政策规则都被添加到目前的政策中。

例如：

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```

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

```

```

const rules = [
  ['jack', 'data4', 'read'],
  ['katy', 'data4', 'write'],
  ['leyo', 'data4', 'read'],
  ['ham', 'data4', 'write']
];

const areRulesAdded = await e.addPolicies(rules);

rules = [
  ["jack", "data4", "read"],
  ["katy", "data4", "write"],
  ["leyo", "data4", "read"],
  ["ham", "data4", "write"]
]
are_rules_added = e.add_policies(rules)

let rules = vec![
  vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
  vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
  vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];
let are_rules_added = e.add_policies(rules).await?

String[][] rules = {
  {"jack", "data4", "read"},
  {"katy", "data4", "write"},
  {"leyo", "data4", "read"},
  {"ham", "data4", "write"},
};

boolean areRulesAdded = e.addPolicies(rules);

```

AddPoliciesEx()

AddPoliciesEx adds authorization rules to the current policy. If the rule already exists, the rule will not be added. But unlike AddPolicies, other non-existent rules are added instead of returning false directly

例如：

Go

```
ok, err := e.AddPoliciesEx([][]string{{"user1", "data1", "read"},  
{"user2", "data2", "read"}})
```

AddNamedPolicy()

AddNamedPolicy 向当前命名策略添加授权规则。如果规则已经存在，函数返回false，并且不会添加规则。否则，函数通过添加新规则并返回true。

例如：

Go

Node.js

PHP

Python

.NET

Rust

Java

```
added := e.AddNamedPolicy("p", "eve", "data3", "read")  
  
const p = ['eve', 'data3', 'read']  
const added = await e.addNamedPolicy('p', ...p)  
  
$added = $e->addNamedPolicy("p", "eve", "data3", "read");  
  
added = e.add_named_policy("p", "eve", "data3", "read")
```

```

var added = e.AddNamedPolicy("p", "eve", "data3", "read");
or
var added = await e.AddNamedPolicyAsync("p", "eve", "data3", "read");

let added = e.add_named_policy("p", vec![eve.to_owned(),
"data3".to_owned(), "read".to_owned()]).await?;

boolean added = e.addNamedPolicy("p", "eve", "data3", "read");

```

AddNamedPolicies()

AddNamedPolicies 向当前命名策略中添加授权规则。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回false，当前政策中没有添加任何政策规则。如果所有授权规则都符合政策规则，则函数返回true，每项政策规则都被添加到目前的政策中。

例如：

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```

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

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

const rules = [
    ['jack', 'data4', 'read'],
    ['katy', 'data4', 'write'],
    ['leyo', 'data4', 'read'],
    ['ham', 'data4', 'write']
]

```

```

rules = [
    ["jack", "data4", "read"],
    ["katy", "data4", "write"],
    ["leyo", "data4", "read"],
    ["ham", "data4", "write"]
]
are_rules_added = e.add_named_policies("p", rules)

let rules = vec![
    vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
    vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];
let are_rules_added := e.add_named_policies("p", rules).await?;

List<List<String>> rules = Arrays.asList(
    Arrays.asList("jack", "data4", "read"),
    Arrays.asList("katy", "data4", "write"),
    Arrays.asList("leyo", "data4", "read"),
    Arrays.asList("ham", "data4", "write")
);
boolean areRulesAdded = e.addNamedPolicies("p", rules);

```

AddNamedPoliciesEx()

AddNamedPoliciesEx adds authorization rules to the current named policy. If the rule already exists, the rule will not be added. But unlike AddNamedPolicies, other non-existent rules are added instead of returning false directly

例如：

[Go](#)

```
ok, err := e.AddNamedPoliciesEx("p", [][]string{{"user1", "data1", "read"}, {"user2", "data2", "read"}})
```

SelfAddPoliciesEx()

SelfAddPoliciesEx adds authorization rules to the current named policy with autoNotifyWatcher disabled. If the rule already exists, the rule will not be added. But unlike SelfAddPolicies, other non-existent rules are added instead of returning false directly

例如：

[Go](#)

```
ok, err := e.SelfAddPoliciesEx("p", "p", [][]string{{"user1", "data1", "read"}, {"user2", "data2", "read"}})
```

RemovePolicy()

RemovePolicy 从当前策略中删除授权规则。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

```
const p = ['alice', 'data1', 'read']
const removed = await e.removePolicy(...p)
```

```
$removed = $e->removePolicy("alice", "data1", "read");

removed = e.remove_policy("alice", "data1", "read")

var removed = e.RemovePolicy("alice", "data1", "read");
or
var removed = await e.RemovePolicyAsync("alice", "data1", "read");

let removed = e.remove_policy(vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;

boolean removed = e.removePolicy("alice", "data1", "read");
```

RemovePolicies()

RemovePolicies 从当前策略中删除授权规则。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回 `false`，当前政策中没有任何政策规则被删除。如果所有授权规则都符合政策规则，则函数返回`true`，每项政策规则都从现行政策中删除。

例如：

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

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

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

```

const rules = [
    ['jack', 'data4', 'read'],
    ['katy', 'data4', 'write'],
    ['leyo', 'data4', 'read'],
    ['ham', 'data4', 'write']
];

const areRulesRemoved = await e.removePolicies(rules);

rules = [
    ["jack", "data4", "read"],
    ["katy", "data4", "write"],
    ["leyo", "data4", "read"],
    ["ham", "data4", "write"]
]
are_rules_removed = e.remove_policies(rules)

let rules = vec![
    vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
    vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];
let are_rules_removed = e.remove_policies(rules).await?;

String[][] rules = {
    {"jack", "data4", "read"},
    {"katy", "data4", "write"},
    {"leyo", "data4", "read"},
    {"ham", "data4", "write"},
};
boolean areRulesRemoved = e.removePolicies(rules);

```

RemoveFilteredPolicy()

RemoveFilteredPolicy 移除当前策略中的授权规则，可以指定字段筛选器。 RemovePolicy 从

当前策略中删除授权规则。

例如：

Go Node.js PHP Python .NET Rust Java

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

const p = ['alice', 'data1', 'read']
const removed = await e.removeFilteredPolicy(0, ...p)

$removed = $e->removeFilteredPolicy(0, "alice", "data1", "read");

removed = e.remove_filtered_policy(0, "alice", "data1", "read")

var removed = e.RemoveFilteredPolicy("alice", "data1", "read");
or
var removed = await e.RemoveFilteredPolicyAsync("alice", "data1",
"read");

let removed = e.remove_filtered_policy(0, vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;

boolean removed = e.removeFilteredPolicy(0, "alice", "data1", "read");
```

RemoveNamedPolicy()

RemoveNamedPolicy 从当前命名策略中删除授权规则。

例如：

Go Node.js PHP Python .NET Rust Java

```

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

const p = ['alice', 'data1', 'read']
const removed = await e.removeNamedPolicy('p', ...p)

$removed = $e->removeNamedPolicy("p", "alice", "data1", "read");

removed = e.remove_named_policy("p", "alice", "data1", "read")

var removed = e.RemoveNamedPolicy("p", "alice", "data1", "read");
or
var removed = await e.RemoveNamedPolicyAsync("p", "alice", "data1",
"read");

let removed = e.remove_named_policy("p", vec!["alice".to_owned(),
"data1".to_owned(), "read".to_owned()]).await?;

boolean removed = e.removeNamedPolicy("p", "alice", "data1", "read");

```

RemoveNamedPolicies()

`RemoveNamedPolicies` 从当前命名策略中删除授权规则。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回 `false`，当前政策中没有任何政策规则被删除。如果所有授权规则都符合政策规则，则函数返回`true`，每项政策规则都从现行政策中删除。

例如：

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```

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

```

```

const rules = [
    ['jack', 'data4', 'read'],
    ['katy', 'data4', 'write'],
    ['leyo', 'data4', 'read'],
    ['ham', 'data4', 'write']
];

const areRulesRemoved = await e.removeNamedPolicies('p', rules);

rules = [
    ["jack", "data4", "read"],
    ["katy", "data4", "write"],
    ["leyo", "data4", "read"],
    ["ham", "data4", "write"]
]
are_rules_removed = e.remove_named_policies("p", rules)

let rules = vec![
    vec!["jack".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["katy".to_owned(), "data4".to_owned(), "write".to_owned()],
    vec!["leyo".to_owned(), "data4".to_owned(), "read".to_owned()],
    vec!["ham".to_owned(), "data4".to_owned(), "write".to_owned()],
];

```

```

let areRulesRemoved = e.remove_named_policies("p", rules).await?;

List<List<String>> rules = Arrays.asList(
    Arrays.asList("jack", "data4", "read"),
    Arrays.asList("katy", "data4", "write"),
    Arrays.asList("leyo", "data4", "read"),
    Arrays.asList("ham", "data4", "write")
);
boolean areRulesRemoved = e.removeNamedPolicies("p", rules);

```

RemoveFilteredNamedPolicy()

RemoveFilteredNamedPolicy 从当前命名策略中移除授权规则，可以指定字段筛选器。

例如：

Go Node.js PHP Python .NET Rust Java

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

const p = ['alice', 'data1', 'read']
const removed = await e.removeFilteredNamedPolicy('p', 0, ...p)

$removed = $e->removeFilteredNamedPolicy("p", 0, "alice", "data1",
"read");

removed = e.remove_filtered_named_policy("p", 0, "alice", "data1",
"read")

var removed = e.RemoveFilteredNamedPolicy("p", 0, "alice", "data1",
"read");
or
var removed = e.RemoveFilteredNamedPolicyAsync("p", 0, "alice",
"data1", "read");

let removed = e.remove_filtered_named_policy("p", 0,
vec!["alice".to_owned(), "data1".to_owned(),
"read".to_owned()]).await?;

boolean removed = e.removeFilteredNamedPolicy("p", 0, "alice",
"data1", "read");
```

HasGroupingPolicy()

HasGroupingPolicy 确定是否存在角色继承规则。

例如：

Go Node.js PHP Python .NET Rust Java

```
has := e.HasGroupingPolicy("alice", "data2_admin")  
  
const has = await e.hasGroupingPolicy('alice', 'data2_admin')  
  
$has = $e->hasGroupingPolicy("alice", "data2_admin");  
  
has = e.has_grouping_policy("alice", "data2_admin")  
  
var has = e.HasGroupingPolicy("alice", "data2_admin");  
  
let has = e.has_grouping_policy(vec![ "alice".to_owned(),  
"data2_admin".to_owned()]);  
  
boolean has = e.hasGroupingPolicy("alice", "data2_admin");
```

HasNamedGroupingPolicy()

HasNamedGroupingPolicy 确定是否存在命名角色继承规则。

例如：

Go Node.js PHP Python .NET Rust Java

```
has := e.HasNamedGroupingPolicy("g", "alice", "data2_admin")  
  
const has = await e.hasNamedGroupingPolicy('g', 'alice', 'data2_admin')  
  
$has = $e->hasNamedGroupingPolicy("g", "alice", "data2_admin");
```

```
has = e.has_named_grouping_policy("g", "alice", "data2_admin")  
  
var has = e.HasNamedGroupingPolicy("g", "alice", "data2_admin");  
  
let has = e.has_named_grouping_policy("g", vec!["alice".to_owned(),  
"data2_admin".to_owned()]);  
  
boolean has = e.hasNamedGroupingPolicy("g", "alice", "data2_admin");
```

AddGroupingPolicy()

AddGroupingPolicy 向当前策略添加角色继承规则。如果规则已经存在，函数返回false，并且不会添加规则。否则，函数通过添加新规则并返回true。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
added := e.AddGroupingPolicy("group1", "data2_admin")  
  
const added = await e.addGroupingPolicy('group1', 'data2_admin')  
  
$added = $e->addGroupingPolicy("group1", "data2_admin");  
  
added = e.add_grouping_policy("group1", "data2_admin")  
  
var added = e.AddGroupingPolicy("group1", "data2_admin");  
or  
var added = await e.AddGroupingPolicyAsync("group1", "data2_admin");  
  
let added = e.add_grouping_policy(vec![ "group1".to_owned(),
```

```
boolean added = e.addGroupingPolicy("group1", "data2_admin");
```

AddGroupingPolicies()

AddGroupingPolicy 向当前策略添加角色继承规则。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回false，当前政策中没有添加任何政策规则。如果所有授权规则都符合政策规则，则函数返回true，每项政策规则都被添加到目前的政策中。

例如：

[Go](#) [Node.js](#) [Python](#) [Rust](#) [Java](#)

```
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesAdded := e.AddGroupingPolicies(rules)

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesAdded = await e.addGroupingPolicies(groupingRules);

rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]

are_rules_added = e.add_grouping_policies(rules)
```

```
let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
    vec!["jack".to_owned(), "data5_admin".to_owned()],
];

let areRulesAdded = e.add_grouping_policies(rules).await?;

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesAdded = e.addGroupingPolicies(groupingRules);
```

AddGroupingPoliciesEx()

AddGroupingPoliciesEx adds role inheritance rules to the current policy. If the rule already exists, the rule will not be added. But unlike AddGroupingPolicies, other non-existent rules are added instead of returning false directly

例如：

Go

```
ok, err := e.AddGroupingPoliciesEx([][]string{{"user1", "member"}, {"user2", "member"}})
```

AddNamedGroupingPolicy()

AddNamedGroupingPolicy 将命名角色继承规则添加到当前策略。如果规则已经存在，函数返回false，并且不会添加规则。否则，函数通过添加新规则并返回true。

例如：

```
added := e.AddNamedGroupingPolicy("g", "group1", "data2_admin")  
  
const added = await e.addNamedGroupingPolicy('g', 'group1',  
    'data2_admin')  
  
$added = $e->addNamedGroupingPolicy("g", "group1", "data2_admin");  
  
added = e.add_named_grouping_policy("g", "group1", "data2_admin")  
  
var added = e.AddNamedGroupingPolicy("g", "group1", "data2_admin");  
or  
var added = await e.AddNamedGroupingPolicyAsync("g", "group1",  
    "data2_admin");  
  
let added = e.add_named_grouping_policy("g", vec![ "group1".to_owned(),  
    "data2_admin".to_owned() ]).await?;  
  
boolean added = e.addNamedGroupingPolicy("g", "group1", "data2_admin");
```

AddNamedGroupingPolicies()

AddNamedGroupingPolicies 将命名角色继承规则添加到当前策略。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回false，当前政策中没有添加任何政策规则。如果所有授权规则都符合政策规则，则函数返回true，每项政策规则都被添加到目前的政策中。

例如：

```

rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

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

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesAdded = await e.addNamedGroupingPolicies('g',
groupingRules);

rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]

are_rules_added = e.add_named_grouping_policies("g", rules)

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
    vec!["jack".to_owned(), "data5_admin".to_owned()],
];

let are_rules_added = e.add_named_grouping_policies("g", rules).await?;

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesAdded = e.addNamedGroupingPolicies("g", groupingRules);

```

AddNamedGroupingPoliciesEx()

AddNamedGroupingPoliciesEx adds named role inheritance rules to the current policy. If the rule already exists, the rule will not be added. But unlike AddNamedGroupingPolicies, other non-existent rules are added instead of returning false directly

例如：

Go

```
ok, err := e.AddNamedGroupingPoliciesEx("g", [][]string{{"user1", "member"}, {"user2", "member"}})
```

RemoveGroupingPolicy()

RemoveGroupingPolicy 从当前策略中删除角色继承规则。

例如：

Go Node.js PHP Python .NET Rust Java

```
removed := e.RemoveGroupingPolicy("alice", "data2_admin")  
  
const removed = await e.removeGroupingPolicy('alice', 'data2_admin')  
  
$removed = $e->removeGroupingPolicy("alice", "data2_admin");  
  
removed = e.remove_grouping_policy("alice", "data2_admin")  
  
var removed = e.RemoveGroupingPolicy("alice", "data2_admin");
```

```
let removed = e.remove_grouping_policy(vec!["alice".to_owned(),
"data2_admin".to_owned()]).await?;

boolean removed = e.removeGroupingPolicy("alice", "data2_admin");
```

RemoveGroupingPolicies()

RemoveGroupingPolicy 从当前策略中删除角色继承规则。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回 `false`，当前政策中没有任何政策规则被删除。如果所有授权规则都符合政策规则，则函数返回`true`，每项政策规则都从现行政策中删除。

例如：

[Go](#) [Node.js](#) [Rust](#) [Python](#) [Java](#)

```
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

areRulesRemoved := e.RemoveGroupingPolicies(rules)

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesRemoved = await e.removeGroupingPolicies(groupingRules);

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],
    vec!["jack".to_owned(), "data5_admin".to_owned()],
];
```

```
rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]

are_rules_removed = e.remove_grouping_policies(rules)

String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesRemoved = e.removeGroupingPolicies(groupingRules);
```

RemoveFilteredGroupingPolicy()

RemoveFilteredGroupingPolicy 从当前策略中移除角色继承规则，可以指定字段筛选器。

例如：

Go Node.js PHP Python .NET Rust Java

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

const removed = await e.removeFilteredGroupingPolicy(0, 'alice')

$removed = $e->removeFilteredGroupingPolicy(0, "alice");

removed = e.remove_filtered_grouping_policy(0, "alice")

var removed = e.RemoveFilteredGroupingPolicy(0, "alice");
or
var removed = await e.RemoveFilteredGroupingPolicyAsync(0, "alice");
```

```
let removed = e.remove_filtered_grouping_policy(0,
vec!["alice".to_owned()]).await?;

boolean removed = e.removeFilteredGroupingPolicy(0, "alice");
```

RemoveNamedGroupingPolicy()

RemoveNamedGroupingPolicy 从当前命名策略中移除角色继承规则。

例如：

Go Node.js PHP Python .NET Rust Java

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

const removed = await e.removeNamedGroupingPolicy('g', 'alice')

$removed = $e->removeNamedGroupingPolicy("g", "alice");

removed = e.remove_named_grouping_policy("g", "alice", "data2_admin")

var removed = e.RemoveNamedGroupingPolicy("g", "alice");
or
var removed = await e.RemoveNamedGroupingPolicyAsync("g", "alice");

let removed = e.remove_named_grouping_policy("g",
vec!["alice".to_owned()]).await?;

boolean removed = e.removeNamedGroupingPolicy("g", "alice");
```

RemoveNamedGroupingPolicies()

RemoveNamedGroupingPolicies 从当前策略中删除角色继承规则。该操作本质上是原子的。因此，如果授权规则由不符合现行政策的规则组成，函数返回 `false`，当前政策中没有任何政策规则被删除。如果所有授权规则都符合政策规则，则函数返回`true`，每项政策规则都从现行政策中删除。

例如：

Go Node.js Python Rust Java

```
rules := [][] string {
    []string {"ham", "data4_admin"},
    []string {"jack", "data5_admin"},
}

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

const groupingRules = [
    ['ham', 'data4_admin'],
    ['jack', 'data5_admin']
];

const areRulesRemoved = await e.removeNamedGroupingPolicies('g',
groupingRules);

rules = [
    ["ham", "data4_admin"],
    ["jack", "data5_admin"]
]
are_rules_removed = e.remove_named_grouping_policies("g", rules)

let rules = vec![
    vec!["ham".to_owned(), "data4_admin".to_owned()],

```

```
String[][] groupingRules = {
    {"ham", "data4_admin"},
    {"jack", "data5_admin"}
};
boolean areRulesRemoved = e.removeNamedGroupingPolicies("g",
groupingRules);
```

RemoveFilteredNamedGroupingPolicy()

RemoveFilteredNamedGroupingPolicy 从当前命名策略中移除角色继承规则，可以指定字段筛选器。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

const removed = await e.removeFilteredNamedGroupingPolicy('g', 0,
'alice')

$removed = $e->removeFilteredNamedGroupingPolicy("g", 0, "alice");

removed = e.remove_filtered_named_grouping_policy("g", 0, "alice")

var removed = e.RemoveFilteredNamedGroupingPolicy("g", 0, "alice");
or
var removed = await e.RemoveFilteredNamedGroupingPolicyAsync("g", 0,
"alice");

let removed = e.remove_filtered_named_groupingPolicy("g", 0,
vec!["alice"].to_owned()).await?;
```

```
boolean removed = e.removeFilteredNamedGroupingPolicy("g", 0, "alice");
```

UpdatePolicy()

UpdatePolicy 把旧的政策更新到新的政策

例如：

[Go](#) [Node.js](#) [Python](#) [Java](#)

```
updated, err := e.UpdatePolicy([]string{"eve", "data3", "read"},  
[]string{"eve", "data3", "write"})  
  
const update = await e.updatePolicy(["eve", "data3", "read"], ["eve",  
"data3", "write"]);  
  
updated = e.update_policy(["eve", "data3", "read"], ["eve", "data3",  
"write"])  
  
boolean updated = e.updatePolicy(Arrays.asList("eve", "data3",  
"read"), Arrays.asList("eve", "data3", "write"));
```

UpdatePolicies()

UpdatePolicies 将所有的旧政策更新到新政策

例如：

[Go](#) [Python](#)

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

AddFunction()

AddFunction 添加自定义函数。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [Rust](#) [Java](#)

```
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  
}
```

```

function customFunction(key1, key2){
    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
    }
}

e.addFunction("keyMatchCustom", customFunction);

func customFunction($key1, $key2) {
    if ($key1 == "/alice_data2/myid/using/res_id" && $key2 ==
"/alice_data/:resource") {
        return true;
    } elseif ($key1 == "/alice_data2/myid/using/res_id" && $key2 ==
"/alice_data2/:id/using/:resId") {
        return true;
    } else {
        return false;
    }
}

func customFunctionWrapper(...$args){
    $key1 := $args[0];
    $key2 := $args[1];

    return customFunction($key1, $key2);
}

$e->addFunction("keyMatchCustom", customFunctionWrapper);

def custom_function(key1, key2):
    return ((key1 == "/alice_data2/myid/using/res_id" and key2 ==
"/alice_data/:resource") or (key1 == "/alice_data2/myid/using/res_id"

```

```

fn custom_function(key1: SString, key2: String) {
    key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data/:resource" || key1 == "/alice_data2/myid/using/res_id" &&
key2 == "/alice_data2/:id/using/:resId"
}

e.add_function("keyMatchCustom", custom_function);

public static class CustomFunc extends CustomFunction {
    @Override
    public AviatorObject call(Map<String, Object> env, AviatorObject
arg1, AviatorObject arg2) {
        String key1 = FunctionUtils.getStringValue(arg1, env);
        String key2 = FunctionUtils.getStringValue(arg2, env);
        if (key1.equals("/alice_data2/myid/using/res_id") &&
key2.equals("/alice_data/:resource")) {
            return AviatorBoolean.valueOf(true);
        } else if (key1.equals("/alice_data2/myid/using/res_id") &&
key2.equals("/alice_data2/:id/using/:resId")) {
            return AviatorBoolean.valueOf(true);
        } else {
            return AviatorBoolean.valueOf(false);
        }
    }

    @Override
    public String getName() {
        return "keyMatchCustom";
    }
}
}

FunctionTest.CustomFunc customFunc = new FunctionTest.CustomFunc();
e.addFunction(customFunc.getName(), customFunc);

```

LoadFilteredPolicy()

LoadFilteredPolicy从文件/数据库加载过滤完成的政策

For example:

[Go](#) [Node.js](#) [Python](#) [Java](#)

```
err := e.LoadFilteredPolicy()

const ok = await e.loadFilteredPolicy();

class Filter:
    P = []
    G = []

adapter =
casbin.persist.adapters.FilteredAdapter("rbac_with_domains_policy.csv")
e = casbin.Enforcer("rbac_with_domains_model.conf", adapter)
filter = Filter()
filter.P = ["", "domain1"]
filter.G = ["", "", "domain1"]
e.load_filtered_policy(filter)

e.loadFilteredPolicy(new String[] { "", "domain1" });
```

LoadIncrementalFilteredPolicy()

LoadFilteredPolicy从文件/数据库添加过滤完成的政策

For example:

[Go](#) [Node.js](#) [Python](#)

```
err := e.LoadIncrementalFilteredPolicy()

const ok = await e.loadIncrementalFilteredPolicy();

adapter =
casbin.persist.adapters.FilteredAdapter("rbac_with_domains_policy.csv")
```

UpdateGroupingPolicy()

UpdateGroupingPolicy 在 g 段更新 oldRule 到 newRule

For example:

[Go](#) [Java](#)

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

UpdateNamedGroupingPolicy()

UpdateNamedGroupingPolicy 在 g 部分将名为 ptype 的旧策略更新为新策略

For example:

[Go](#) [Java](#)

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

SetFieldIndex()

设置 FieldIndex 子端口自定义常规名称和位置子类, obj, 域 和 优先级.

```
[policy_definition]
p = customized_priority, obj, act, eft, subject
```

For example:

[Go](#)

```
e.SetFieldIndex("p", constant.PriorityIndex, 0)
e.SetFieldIndex("p", constant.SubjectIndex, 4)
```

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 Enforcer 实例。

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

```
const e = await newEnforcer('examples/rbac_model.conf',  
'examples/rbac_policy.csv')
```

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

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

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

```
let mut e = Enforcer::new("examples/rbac_model.conf", "examples/
```

```
Enforcer e = new Enforcer("examples/rbac_model.conf", "examples/rbac_policy.csv");
```

GetRolesForUser()

GetRolesForUser 获取用户具有的角色。

例如:

Go Node.js PHP Python .NET Rust Java

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

```
const res = await e.getRolesForUser('alice')
```

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

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

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

```
let roles = e.get_roles_for_user("alice", None); // 无域
```

```
List<String> res = e.getRolesForUser("alice");
```

GetUsersForRole()

GetUsersForRole 获取具有角色的用户。

例如:

Go Node.js PHP Python .NET Rust Java

```
res := e.GetUsersForRole("data1_admin")  
  
const res = await e.getUsersForRole('data1_admin')  
  
$res = $e->getUsersForRole("data1_admin");  
  
users = e.get_users_for_role("data1_admin")  
  
var res = e.GetUsersForRole("data1_admin");  
  
let users = e.get_users_for_role("data1_admin", None); // No  
domain  
  
List<String> res = e.getUsersForRole("data1_admin");
```

HasRoleForUser()

HasRoleForUser 确定用户是否具有角色。

例如:

Go Node.js PHP Python .NET Rust Java

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

```
const res = await e.hasRoleForUser('alice', 'data1_admin')

$res = $e->hasRoleForUser("alice", "data1_admin");

has = e.has_role_for_user("alice", "data1_admin")

var res = e.HasRoleForUser("alice", "data1_admin");

let has = e.has_role_for_user("alice", "data1_admin", None); //  
No domain

boolean 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，则返回false。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

await e.addRoleForUser('alice', 'data2_admin')

$e->addRoleForUser("alice", "data2_admin");

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

```
var added = e.AddRoleForUser("alice", "data2_admin");
or
var added = await e.AddRoleForUserAsync("alice", "data2_admin");

let added = e.add_role_for_user("alice", "data2_admin",
None).await?; // No domain

boolean added = e.addRoleForUser("alice", "data2_admin");
```

AddRolesForUser()

AddRolesForUser 为用户添加多个角色。如果用户已经拥有该角色，则返回false。(不会受影响)

例如:

[Go](#) [Node.js](#) [Rust](#)

```
var roles = []string{"data2_admin", "data1_admin"}
e.AddRolesForUser("alice", roles)

const roles = ["data1_admin", "data2_admin"];
roles.map((role) => e.addRoleForUser("alice", role));

let roles = vec!["data1_admin".to_owned(),
"data2_admin".to_owned()];
let all_added = e.add_roles_for_user("alice", roles,
None).await?; // 无域
```

DeleteRoleForUser()

DeleteRoleForUser 删除用户的角色。如果用户没有该角色，则返回false。

例如：

Go

Node.js

PHP

Python

.NET

Rust

Java

```
e.DeleteRoleForUser("alice", "data1_admin")  
  
await e.deleteRoleForUser('alice', 'data1_admin')  
  
$e->deleteRoleForUser("alice", "data1_admin");  
  
e.delete_role_for_user("alice", "data1_admin")  
  
var deleted = e.DeleteRoleForUser("alice", "data1_admin");  
or  
var deleted = await e.DeleteRoleForUser("alice", "data1_admin");  
  
let deleted = e.delete_role_for_user("alice", "data1_admin",  
None).await?; // No domain  
  
boolean deleted = e.deleteRoleForUser("alice", "data1_admin");
```

DeleteRolesForUser()

DeleteRolesForUser 删除用户的所有角色。如果用户没有任何角色，则返回false。

例如：

Go Node.js PHP Python .NET Rust Java

```
e.DeleteRolesForUser("alice")

await e.deleteRolesForUser('alice')

$e->deleteRolesForUser("alice");

e.delete_roles_for_user("alice")

var deletedAtLeastOne = e.DeleteRolesForUser("alice");
or
var deletedAtLeastOne = await
e.DeleteRolesForUserAsync("alice");

let deleted_at_least_one = e.delete_roles_for_user("alice",
None).await?; // 无域

boolean deletedAtLeastOne = e.deleteRolesForUser("alice");
```

DeleteUser()

DeleteUser 删除一个用户。如果用户不存在，则返回false（也就是说不受影响）。

例如：

Go Node.js PHP Python .NET Rust Java

```
e.DeleteUser("alice")

await e.deleteUser('alice')

$e->deleteUser("alice");

e.delete_user("alice")

var deleted = e.DeleteUser("alice");
or
var deleted = await e.DeleteUserAsync("alice");

let deleted = e.delete_user("alice").await?;

boolean deleted = e.deleteUser("alice");
```

DeleteRole()

DeleteRole 删除一个角色。

例如:

Go Node.js PHP Python .NET Rust Java

```
e.DeleteRole("data2_admin")

await e.deleteRole("data2_admin")
```

```
$e->deleteRole("data2_admin");

e.delete_role("data2_admin")

var deleted = e.DeleteRole("data2_admin");
or
var deleted = await e.DeleteRoleAsync("data2_admin");

let deleted = e.delete_role("data2_admin").await?;

e.deleteRole("data2_admin");
```

DeletePermission()

DeletePermission 删除权限。 如果权限不存在，则返回false（aka不受影响）。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e.DeletePermission("read")

await e.deletePermission('read')

$e->deletePermission("read");

e.delete_permission("read")
```

```
var deleted = e.DeletePermission("read");
or
var deleted = await e.DeletePermissionAsync("read");

let deleted =
e.delete_permission(vec!["read".to_owned()]).await?;

boolean deleted = e.deletePermission("read");
```

AddPermissionForUser()

AddPermissionForUser 为用户或角色添加权限。如果用户或角色已经拥有该权限(aka不受影响)，则返回false。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

await e.addPermissionForUser('bob', 'read')

$e->addPermissionForUser("bob", "read");

e.add_permission_for_user("bob", "read")

var added = e.AddPermissionForUser("bob", "read");
or
var added = await e.AddPermissionForUserAsync("bob", "read");
```

```
let added = e.add_permission_for_user("bob",
    vec!["read".to_owned()]).await?;

boolean added = e.addPermissionForUser("bob", "read");
```

AddPermissionsForUser()

AddPermissionForUser 为用户或角色添加多个权限。如果用户或角色已经有一个权限，则返回 false (不会受影响)。

例如：

[Go](#) [Node.js](#) [Rust](#)

```
var permissions = [][]string{{"data1",
    "read"}, {"data2", "write"}}
for i := 0; i < len(permissions); i++ {
    e.AddPermissionsForUser("alice", permissions[i])
}

const permissions = [
    ["data1", "read"],
    ["data2", "write"],
];
permissions.map((permission) => e.addPermissionForUser("bob",
    ...permission));

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

DeletePermissionForUser()

DeletePermissionForUser 删除用户或角色的权限。如果用户或角色没有权限（aka不受影响），则返回false。

例如：

Go Node.js PHP Python .NET Rust Java

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

await e.deletePermissionForUser("bob", "read")

$e->deletePermissionForUser("bob", "read");

e.delete_permission_for_user("bob", "read")

var deleted = e.DeletePermissionForUser("bob", "read");
or
var deleted = await e.DeletePermissionForUserAsync("bob",
"read");

let deleted = e.delete_permission_for_user("bob",
vec!["read".to_owned()]).await?;

boolean deleted = e.deletePermissionForUser("bob", "read");
```

DeletePermissionsForUser()

DeletePermissionsForUser 删除用户或角色的权限。如果用户或角色没有任何权限(aka不受影响) , 则返回false。

例如:

Go Node.js PHP Python .NET Rust Java

```
e.DeletePermissionsForUser("bob")  
  
await e.deletePermissionsForUser('bob')  
  
$e->deletePermissionsForUser("bob");  
  
e.delete_permissions_for_user("bob")  
  
var deletedAtLeastOne = e.DeletePermissionsForUser("bob");  
or  
var deletedAtLeastOne = await  
e.DeletePermissionsForUserAsync("bob");  
  
let deleted_at_least_one =  
e.delete_permissions_for_user("bob").await?;  
  
boolean deletedAtLeastOne = e.deletePermissionForUser("bob");
```

GetPermissionsForUser()

GetPermissionsForUser 获取用户或角色的权限。

例如：

Go Node.js PHP Python .NET Java

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

```
await e.getPermissionsForUser('bob')
```

```
$e->getPermissionsForUser("bob");
```

```
e.get_permissions_for_user("bob")
```

```
var permissions = e.GetPermissionsForUser("bob");
```

```
List<List<String>> permissions = e.getPermissionsForUser("bob");
```

HasPermissionForUser()

HasPermissionForUser 确定用户是否具有权限。

例如：

Go Node.js PHP Python .NET Rust Java

```
e.HasPermissionForUser("alice", []string{"read"})  
  
await e.hasPermissionForUser('alice', 'read')  
  
$e->hasPermissionForUser("alice", []string{"read"});  
  
has = e.has_permission_for_user("alice", "read")  
  
var has = e.HasPermissionForUser("bob", "read");  
  
let has = e.has_permission_for_user("alice",  
    vec!["data1".to_owned(), "read".to_owned()]);  
  
boolean has = e.hasPermissionForUser("alice", "read");
```

GetImplicitRolesForUser()

GetImplicitRolesForUser 获取用户具有的隐式角色。与GetRolesForUser()相比，该函数除了直接角色外还检索间接角色。

例如：

g, Alice, role:admin

g, role:admin, role:user

GetRolesForUser("Alice") 只能获取：["role:admin"]。

但 GetImplicitRolesForUser("alice") 将获取：["role:admin", "role:user"]。

例如：

```
e.GetImplicitRolesForUser("alice")  
  
await e.getImplicitRolesForUser("alice")  
  
$e->getImplicitRolesForUser("alice");  
  
e.get_implicit_roles_for_user("alice")  
  
var implicitRoles = e.GetImplicitRolesForUser("alice");  
  
e.get_implicit_roles_for_user("alice", None); // No domain  
  
List<String> implicitRoles = e.getImplicitRolesForUser("alice");
```

GetImplicitUsersForRole()

GetImplicitUsersForRole 获取所有继承该角色的用户 与GetUsersForRole() 相比, 这个函数检索间接用户。

例如:

g, Alice, role:admin

g, role:admin, role:user

GetUsersForRole("role:user") 仅会得到: ["role:admin"].

But GetImplicitUsersForRole("role:user") 将会得到: ["role:admin", "alice"].

例如:

[Go](#) [Node.js](#) [Java](#)

```
users := e.GetImplicitUsersForRole("role:user")  
  
const users = e.getImplicitUsersForRole("role:user");  
  
List<String> users = e.getImplicitUsersForRole("role:user");
```

GetImplicitPermissionsForUser()

GetImplicitPermissionsForUser 获得用户或角色的隐含权限。

与GetPermissionsForUser() 相比，此函数获取继承角色的权限。

例如：

```
p, admin, data1, read  
p, alice, data2, read  
g, alice, admin
```

GetPermissionsForUser("alice") 只能获取：[["alice", "data 2", "read"]]。

但GetImplicitPermissionsForUser("alice") 将获取：[["admin", "data1", "read"], ["alice", "data2", "read"]]。

例如：

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

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

```
await e.getImplicitPermissionsForUser("alice")  
  
$e->getImplicitPermissionsForUser("alice");  
  
e.get_implicit_permissions_for_user("alice")  
  
var implicitPermissions =  
e.GetImplicitPermissionsForUser("alice");  
  
e.get_implicit_permissions_for_user("alice", None); // 无域  
  
List<List<String>> implicitPermissions =  
e.getImplicitPermissionsForUser("alice");
```

GetNamedImplicitPermissionsForUser()

GetNamedImplicitPermissionsForUser 通过命名策略获得用户或角色的隐含权限 与 GetImplicitPermissionPermissionsForUser() 相比，此函数允许您指定策略名称。

例如： p, admin, data1, read p2, admin, creaty g, alice, admin

GetImplicitPermissionsForUser("Alice") 只能获取到默认为“p”的权限: [[{"admin", "data1", "read"}]]

但是你可以指定策略为 "p2" 来获取: [[{"admin", "create"}]] 由
GetNamedImplicitPermissionsForUser("p2","Alice")

例如：

[Go](#) [Python](#)

```
e.GetNamedImplicitPermissionsForUser("p2", "alice")  
e.get_named_implicit_permissions_for_user("p2", "alice")
```

GetDomainsForUser()

GetDomainsForUser 获取用户拥有的所有域名。

例如： 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") 可以获取 ["domain1", "domain2"]

例如：

Go

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

GetImplicitResourcesForUser()

GetImplicitResourcesForUser 返回为true的策略给用户。

例如：

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

`GetImplicitResourcesForUser("alice")` 将会返回 `[[alice data1 read] [alice data2 read] [alice data2 write]]`

[Go](#)

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

GetImplicitUsersForPermission()

`GetImplicitUsersForPermission` gets implicit users for a permission.

For example:

```
p, admin, data1, read  
p, bob, data1, read  
g, alice, admin
```

`GetImplicitUsersForPermission("data1", "read")` will return: `["alice", "bob"]`.

Note: only users will be returned, roles (2nd arg in "g") will be excluded.

[Go](#)

```
users, err := e.GetImplicitUsersForPermission("data1", "read")
```

GetAllowedObjectConditions()

`GetAllowedObjectConditions` returns a string array of object conditions that the

user can access.

For example:

```
p, alice, r.obj.price < 25, read  
p, admin, r.obj.category_id = 2, read  
p, bob, r.obj.author = bob, write  
  
g, alice, admin
```

e.GetAllowedObjectConditions("alice", "read", "r.obj.") will return ["price < 25", "category_id = 2"], nil

Note:

0. prefix: You can customize the prefix of the object conditions, and "r.obj." is commonly used as a prefix. After removing the prefix, the remaining part is the condition of the object. If there is an obj policy that does not meet the prefix requirement, an errors.ERR_OBJ_CONDITION will be returned.
1. If the 'objectConditions' array is empty, return errors.ERR_EMPTY_CONDITION This error is returned because some data adapters' ORM return full table data by default when they receive an empty condition, which tends to behave contrary to expectations.(e.g. GORM) If you are using an adapter that does not behave like this, you can choose to ignore this error.

Go

```
conditions, err := e.GetAllowedObjectConditions("alice",  
"read", "r.obj.")
```

GetImplicitUsersForResource()

GetImplicitUsersForResource return implicit user based on resource.

For example:

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

GetImplicitUsersForResource("data2") will return `[["bob", "data2", "write"], ["alice", "data2", "read"] ["alice", "data2", "write"]]`, nil.

GetImplicitUsersForResource("data1") will return `[["alice", "data1", "read"]]`, nil.

Go

```
ImplicitUsers, err := e.GetImplicitUsersForResource("data2")
```

ⓘ 备注

Only users will be returned, roles (2nd arg in "g") will be excluded.

域内基于角色的访问控制 API

A more user-friendly API for RBAC with domains. This API is a subset of the Management API. RBAC users can use this API to simplify their code.

参考

The global variable `e` represents the Enforcer instance.

[Go](#) [Node.js](#) [PHP](#) [Python](#) [.NET](#) [Rust](#) [Java](#)

```
e, err := NewEnforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv")
```

```
const e = await newEnforcer('examples/
rbac_with_domains_model.conf', 'examples/
rbac_with_domains_policy.csv')
```

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

```
e = casbin.Enforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv")
```

```
var e = new Enforcer("examples/rbac_with_domains_model.conf",
"examples/rbac_with_domains_policy.csv");
```

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

```
Enforcer e = new Enforcer("examples/
rbac_with_domains_model.conf", "examples/
rbac_with_domains_policy.csv");
```

GetUsersForRoleInDomain()

The `GetUsersForRoleInDomain()` function retrieves the users that have a role within a domain.

例如:

[Go](#) [Node.js](#) [Python](#)

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

```
const res = e.getUsersForRoleInDomain("admin", "domain1")
```

```
res = e.get_users_for_role_in_domain("admin", "domain1")
```

GetRolesForUserInDomain()

The `GetRolesForUserInDomain()` function retrieves the roles that a user has within a domain.

例如:

Go Node.js Python Java

```
res := e.GetRolesForUserInDomain("admin", "domain1")  
  
const res = e.getRolesForUserInDomain("alice", "domain1")  
  
res = e.get_roles_for_user_in_domain("alice", "domain1")  
  
List<String> res = e.getRolesForUserInDomain("admin",  
"domain1");
```

GetPermissionsForUserInDomain()

The `GetPermissionsForUserInDomain()` function retrieves the permissions for a user or role within a domain.

例如:

Go Java

```
res := e.GetPermissionsForUserInDomain("alice", "domain1")  
  
List<List<String>> res =  
e.getPermissionsForUserInDomain("alice", "domain1");
```

AddRoleForUserInDomain()

The `AddRoleForUserInDomain()` function adds a role for a user within a domain.

It returns `false` if the user already has the role (no changes made).

例如:

Go Python Java

```
ok, err := e.AddRoleForUserInDomain("alice", "admin", "domain1")  
  
ok = e.add_role_for_user_in_domain("alice", "admin", "domain1")  
  
boolean ok = e.addRoleForUserInDomain("alice", "admin",  
"domain1");
```

DeleteRoleForUserInDomain()

The `DeleteRoleForUserInDomain()` function removes a role for a user within a domain. It returns `false` if the user does not have the role (no changes made).

例如:

Go Java

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

DeleteRolesForUserInDomain()

The `DeleteRolesForUserInDomain()` function removes all roles for a user within a domain. It returns `false` if the user does not have any roles (no changes made).

例如:

[Go](#)

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

GetAllUsersByDomain()

The `GetAllUsersByDomain()` function retrieves all users associated with the given domain. It returns an empty string array if no domain is defined in the model.

例如:

[Go](#)

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

DeleteAllUsersByDomain()

The `DeleteAllUsersByDomain()` function deletes all users associated with the given domain. It returns `false` if no domain is defined in the model.

例如:

[Go](#)

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

DeleteDomains()

DeleteDomains 将删除所有相关的用户和角色。如果没有提供参数，它会删除所有域。

例如:

[Go](#)

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

GetAllDomains()

GetAllDomains 将获得所有域。

例如:

[Go](#)

```
res, _ := e.GetAllDomains()
```

ⓘ 备注

如果您正在处理类似 `name::domain` 的域，这可能会导致意外的行为。In Casbin, `::` is a reserved keyword, just like `for`, `if` in a programming language, we should never put `::` in a domain.

GetImplicitUsersForResourceByDomain()

`GetImplicitUsersForResourceByDomain` return implicit user based on resource and domain.

For example:

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

`GetImplicitUsersForResourceByDomain("data1", "domain1")` will return `[["alice", "domain1", "data1", "read"], ["alice", "domain1", "data1", "write"]]`, `nil`

Go

```
ImplicitUsers, err :=  
e.GetImplicitUsersForResourceByDomain("data1", "domain1")
```

① 备注

Only users will be returned, roles (2nd arg in "g") will be excluded.

RBAC with Conditions API

A more user-friendly API for [RBAC with conditions](#).

Reference

AddNamedLinkConditionFunc

`AddNamedLinkConditionFunc` Add condition function fn for Link `userName->roleName`, when fn returns true, Link is valid, otherwise invalid

[Go](#)

```
e.AddNamedLinkConditionFunc("g", "userName", "roleName",  
YourLinkConditionFunc)
```

AddNamedDomainLinkConditionFunc

`AddNamedDomainLinkConditionFunc` Add condition function fn for Link `userName-> {roleName, domain}`, when fn returns true, Link is valid, otherwise invalid

[Go](#)

```
e.AddNamedDomainLinkConditionFunc("g", "userName", "roleName",
"domainName", YourLinkConditionFunc)
```

SetNamedLinkConditionFuncParams

`SetNamedLinkConditionFuncParams` Sets the parameters of the condition function fn for Link `userName->roleName`

[Go](#)

```
e.SetNamedLinkConditionFuncParams("g", "userName", "roleName",
"YourConditionFuncParam")
e.SetNamedLinkConditionFuncParams("g", "userName2",
"roleName2", "YourConditionFuncParam_1",
"YourConditionFuncParam_2")
```

SetNamedDomainLinkConditionFuncParams

`SetNamedDomainLinkConditionFuncParams` Sets the parameters of the condition function fn for Link `userName->{roleName, domain}`

[Go](#)

```
e.SetNamedDomainLinkConditionFuncParams("g", "userName",
"roleName", "domainName", "YourConditionFuncParam")
e.SetNamedDomainLinkConditionFuncParams("g", "userName2",
"roleName2", "domainName2", "YourConditionFuncParam_1",
"YourConditionFuncParam_2")
```


角色管理器API

角色管理器

The RoleManager provides an interface for defining operations to manage roles. The addition of a matching function to the RoleManager allows the use of wildcards in role names and domains.

AddNamedMatchingFunc()

The `AddNamedMatchingFunc` function adds a `MatchingFunc` by Ptype to the RoleManager. The `MatchingFunc` will be used when performing role matching.

[Go](#) [Node.js](#)

```
e.AddNamedMatchingFunc("g", "", util.KeyMatch)
_, _ = e.AddGroupingPolicies([][]string{{"*", "admin",
"domain1"}})
_, _ = e.GetRoleManager().HasLink("bob", "admin",
"domain1") // -> true, nil

await e.addNamedMatchingFunc('g', Util.keyMatchFunc);
await e.addGroupingPolicies([['*', 'admin', 'domain1']]);
await e.getRoleManager().hasLink('bob', 'admin', 'domain1');
```

例如:

[Go](#) Node.js

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

const e = await newEnforcer('path/to/model', 'path/to/
policy');
await e.addNamedMatchingFunc('g', Util.keyMatchFunc);
```

AddNamedDomainMatchingFunc()

The `AddNamedDomainMatchingFunc` function adds a `MatchingFunc` by Ptype to the RoleManager. The `DomainMatchingFunc` is similar to the `MatchingFunc` listed above.

For example:

[Go](#) Node.js

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

const e = await newEnforcer('path/to/model', 'path/to/
policy');
await e.addNamedDomainMatchingFunc('g', Util.keyMatchFunc);
```

GetRoleManager()

The `GetRoleManager` function gets the current role manager for `g`.

例如：

[Go](#) [Node.js](#) [Python](#)

```
rm := e.GetRoleManager()  
  
const rm = await e.getRoleManager();  
  
rm = e.get_role_manager()
```

GetNamedRoleManager()

The `GetNamedRoleManager` function gets the role manager by named Ptype.

例如：

[Go](#) [Node.js](#) [Python](#)

```
rm := e.GetNamedRoleManager("g2")  
  
const rm = await e.getNamedRoleManager("g2");  
  
rm = e.get_named_role_manager("g2")
```

SetRoleManager()

The `SetRoleManager` function sets the current role manager for `g`.

例如：

[Go](#) [Node.js](#) [Python](#)

```
e.SetRoleManager(rm)
```

```
e.setRoleManager(rm);
```

```
rm = e.set_role_manager(rm)
```

SetNamedRoleManager()

The `SetNamedRoleManager` function sets the role manager by named Ptype.

For example:

[Go](#) [Python](#)

```
rm := e.SetNamedRoleManager("g2", rm)
```

```
rm = e.set_role_manager("g2", rm)
```

Clear()

The `Clear` function clears all stored data and resets the role manager to its initial state.

例如：

[Go](#) [Node.js](#) [Python](#)

```
rm.Clear()  
  
await rm.clear();  
  
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 域是角色的前缀(可以用于其他目的)。

例如：

[Go](#) [Node.js](#) [Python](#)

```
rm.AddLink("u1", "g1", "domain1")  
  
await rm.addLink('u1', 'g1', 'domain1');
```

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

DeleteLink()

DeleteLink 删除两个角色之间的继承链接。 角色: 名称1 和 角色: 名称2 域是角色的前缀(可以用于其他目的)。

例如:

[Go](#) [Node.js](#) [Python](#)

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

```
await rm.deleteLink('u1', 'g1', 'domain1');
```

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

HasLink()

HasLink 决定两种角色之间是否存在联系。 role: name1 继承自 role: name2. 域是角色的前缀(可以用于其他目的)。

例如:

[Go](#) [Node.js](#) [Python](#)

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

```
await rm.hasLink('u1', 'g1', 'domain1');

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

GetRoles()

GetRoles 获取一个用户所继承的角色 域是角色的前缀(可以用于其他目的)。

例如:

[Go](#) [Node.js](#) [Python](#)

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

await rm.getRoles('u1', 'domain1');

rm.get_roles("u1", "domain")
```

GetUsers()

GetUsers 获取继承自一个角色的用户 域是用户的前缀(可以用于其他目的)。

例如:

[Go](#) [Node.js](#) [Python](#)

```
rm.GetUsers("g1")  
  
await rm.getUsers('g1');  
  
rm.get_users("g1")
```

PrintRoles()

PrintRoles 打印所有的角色到日志。

例如:

[Go](#) [Node.js](#) [Python](#)

```
rm.PrintRoles()  
  
await rm.printRoles();  
  
rm.print_roles()
```

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 获取用户拥有的域

例如:

Go

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

数据权限

We have two solutions for data permissions (filtering): using implicit assignment APIs or using the `BatchEnforce()` API.

1. Query Implicit Roles or Permissions

When a user inherits a role or permission via an RBAC hierarchy instead of being directly assigned them in a policy rule, we refer to this type of assignment as "implicit". To query such implicit relations, you need to use the following two APIs: `GetImplicitRolesForUser()` and `GetImplicitPermissionsForUser()`, instead of `GetRolesForUser()` and `GetPermissionsForUser()`. For more details, please refer to [this GitHub issue](#).

2. 使用 `BatchEnforce()`

`BatchEnforce()` enforces each request and returns the results in a boolean array.

例如:

Go Node.js Java

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

```
const boolArray = await e.batchEnforce(requests);
```

```
List<Boolean> boolArray = e.batchEnforce(requests);
```



> 高级用法

高级用法

多线程

Utilizing Casbin in a multi-threading environment

Benchmarks

Overhead of Policy Enforcement in Casbin

性能优化

性能优化

Kubernetes的授权

Kubernetes (k8s) RBAC & ABAC授权基于Casbin 的中间件

Admission Webhook for K8s

Kubernetes (K8s) RBAC & ABAC Authorization Middleware based on Casbin

使用 Envoy 实现 Service Mesh 权限管理

使用 Envoy 实现 Service Mesh 权限管理

多线程

When using Casbin in a multi-threading environment, you can employ the synchronized wrapper of the Casbin enforcer: https://github.com/casbin/casbin/blob/master/enforcer_synced.go (GoLang) and https://github.com/casbin/casbin-cpp/blob/master/casbin/enforcer_synced.cpp (C++).

Furthermore, it also provides support for the "AutoLoad" feature, allowing the Casbin enforcer to automatically load the latest policy rules from the database if any changes occur. To initiate the automatic loading of policies periodically, call the "StartAutoLoadPolicy()" function. Likewise, to stop this automatic loading, call the "StopAutoLoadPolicy()" function.

Benchmarks

Go Python C++ Lua (JIT)

The overhead of policy enforcement has been benchmarked in `model_b_test.go`. The testbed configuration is as follows:

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

Following are the benchmarking results obtained by running `go test -bench=. -benchmem` (op = an `Enforce()` call, ms = millisecond, KB = kilobytes):

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

The overhead of policy enforcement in `Pycasbin` has been benchmarked in the `tests/benchmarks` directory. The testbed configuration is as follows:

Intel(R) Xeon(R) CPU E5-2673 v4 @ 2.30GHz (Runned by Github actions)
platform linux -- Python 3.11.4, pytest-7.0.1, pluggy-1.2.0

Here are the benchmarking results obtained from executing `casbin_benchmark` (op = an `enforce()` call, ms = millisecond):

测试用例	规则大小	时间开销 (ms/op)
ACL	2 规则 (2用户)	0.067691

测试用例	规则大小	时间开销 (ms/op)
RBAC	5条规则 (2用户, 1个角色)	0.080045
RBAC (小型)	1100条规则 (1000用户, 100个角色)	0.853590
RBAC (中型)	11000条规则 (10000用户, 1000个角色)	6.986668
RBAC (大型)	110000条规则 (100000用户, 10000个角色)	77.922851
具有资源角色的RBAC	6条规则 (2用户, 2个角色)	0.106090
带有域/租户的RBAC	6 条规则 (2个用户, 1个角色, 2个域)	0.103628
ABAC	0 规则 (0用户)	0.053213
RESTful	5 规则 (3用户)	NA
拒绝改写	6条规则 (2用户, 1个角色)	NA
优先级	9条规则 (2用户, 2个角色)	0.084684

The overhead of policy enforcement in Casbin CPP has been benchmarked in the `tests/benchmarks` directory using Google's [benchmarking tool](#). The testbed configuration is as follows:

英特尔 酷睿 i5-6300HQ CPU @ 2.30GHz, 4 核, 4 线程

Here are the benchmarking results obtained from executing the `casbin_benchmark` target built in the `Release` configuration (`op = an enforce() call, ms = millisecond`):

测试用例	规则大小	时间开销 (ms/op)
ACL	2 规则 (2用户)	0.0195
RBAC	5条规则 (2用户, 1个角色)	0.0288
RBAC (小型)	1100条规则 (1000用户, 100个角色)	0.300
RBAC (中型)	11000条规则 (10000用户, 1000个角色)	2.113
RBAC (大型)	110000条规则 (100000用户, 10000个角色)	21.450
具有资源角色的RBAC	6条规则 (2用户, 2个角色)	0.03
带有域/租户的RBAC	6 条规则 (2个用户, 1个角色, 2个域)	0.041
ABAC	0 规则 (0用户)	NA
RESTful	5 规则 (3用户)	NA

测试用例	规则大小	时间开销 (ms/op)
拒绝改写	6条规则 (2用户, 1个角色)	0.0246
优先级	9条规则 (2用户, 2个角色)	0.035

The overhead of policy enforcement in [Lua Casbin](#) has been benchmarked in `bench.lua`. The testbed configuration is as follows:

AMD Ryzen(TM) 5 4600H CPU @ 3.0GHz, 6 核, 12 Threads

Here are the benchmarking results obtained by running `luajit bench.lua` (op = an `enforce()` call, ms = millisecond):

Test case	Rule size	Time overhead (ms/op)
ACL	2 rules (2 users)	0.0533
RBAC	5 rules (2 users, 1 role)	0.0972
RBAC (small)	1100 rules (1000 users, 100 roles)	0.8598
RBAC (medium)	11000 rules (10000 users, 1000 roles)	8.6848
RBAC (large)	110000 rules (100000 users, 10000 roles)	90.3217
RBAC with resource roles	6 rules (2 users, 2 roles)	0.1124
RBAC with domains/tenants	6 rules (2 users, 1 role, 2 domains)	0.1978
ABAC	0 rule (0 user)	0.0305
RESTful	5 rules (3 users)	0.1085
Deny-override	6 rules (2 users, 1 role)	0.1934
Priority	9 rules (2 users, 2 roles)	0.1437

Benchmark monitoring

In the embedded web page below, you can see the performance changes of Casbin for each commit.

You can also directly browse it at: <https://v1.casbin.org/casbin/benchmark-monitoring>

Last Update:
Repository:

[Download data as JSON](#)

Powered by [github-action-benchmark](#)

性能优化

When applied in a production environment with millions of users or permissions, you may encounter a performance downgrade in Casbin enforcement. There are usually two causes:

高访问量

The number of incoming requests per second is too large, for example, 10,000 requests/s for a single Casbin instance. In such cases, a single Casbin instance is usually not enough to handle all the requests. There are two possible solutions:

1. 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。For more details, see: [Multi-threading](#).
2. Deploy Casbin instances to a cluster (multiple machines) and use Watcher to ensure all Casbin instances are consistent. For more details, see: [Watchers](#).

备注

You can use both of the above methods at the same time, for example, deploy Casbin to a 10-machine cluster where each machine has 5 threads simultaneously serving Casbin enforcement requests.

大量的策略规则

In a cloud or multi-tenant environment, millions of policy rules may be required. Each enforcement call or even loading the policy rules at the initial time can be very slow. 这类事件通常可以通过以下几种方式缓解：

1. Check if your Casbin model or policy is well-designed. A well-written model and policy abstracts out the duplicated logic for each user/tenant and reduces the number of rules to a very small level (< 100). For example, you can share some default rules across all tenants and allow users to customize their rules later. Customized rules can override the default rules. If you have any further questions, please open a GitHub issue on the Casbin repository.
2. Do sharding to let a Casbin enforcer only load a small set of policy rules. For example, enforcer_0 can serve tenant_0 to tenant_99, while enforcer_1 can serve tenant_100 to tenant_199. To load only a subset of all policy rules, see: [Policy Subset Loading](#).
3. 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)。 So, given a user like Alice, Casbin only takes O(1) time to query the RBAC tree for the role-user relationship and perform enforcement. If your g rules don't change often, then the RBAC tree won't need to be constantly updated. See the details of this discussion here:
<https://github.com/casbin/casbin/issues/681#issuecomment-763801583>

ⓘ 备注

You can try all of the above methods at the same time.

Kubernetes的授权

[K8s-authz](#) is a Kubernetes (k8s) authorization middleware based on Casbin that utilizes RBAC (Role-Based Access Control) and ABAC (Attribute-Based Access Control) for policy enforcement. This middleware integrates with the K8s validation admission webhook to validate the policies defined by Casbin for each request made to K8s resources. Custom admission controllers are registered with Kubernetes using the `ValidatingAdmissionWebhook` to perform validations on request objects forwarded by the API server and provide a response indicating whether the request should be allowed or rejected.

To determine when to send incoming requests to the admission controller, a validation webhook has been implemented. This webhook proxies requests for any type of K8s resource or sub-resource and performs policy verification. Users are only allowed to perform operations on these resources if they are authorized by the Casbin enforcer. The [enforcer](#) checks the roles of the user as defined in the policies. The K8s cluster is the deployment target for this middleware.

需求

Before proceeding, ensure that you have the following:

- A running Kubernetes cluster. You can set up a local cluster using Docker or set up a complete Kubernetes ecosystem on your server. For detailed instructions, refer to this [guide](#) for setting up a local Kubernetes cluster on Windows or this [guide](#) for setting up a cluster on Linux.
- Kubectl CLI. Instructions for installing Kubectl on Windows can be found [here](#), and for Linux [here](#).

- OpenSSL

使用方法

Follow these steps to use K8s-authz:

1. Generate certificates and keys for each user using OpenSSL. Run the script below:

```
./gen_cert.sh
```

2. Build the Docker image from the [Dockerfile](#) manually by running the following command. Remember to change the build version in the command and in the deployment [file](#) accordingly.

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

3. Define the Casbin policies in the [model.conf](#) and [policy.csv](#) files. For more information on how these policies work, refer to the [documentation](#).
4. Before deploying, you can modify the ports in the [main.go](#) file, as well as in the validation webhook configuration [file](#), based on your specific requirements.
5. Deploy the validation controller and the webhook on the Kubernetes cluster by running the following command:

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

6. For a production server, it is recommended to create a Kubernetes [secret](#) to

secure the certificates:

```
kubectl create secret generic casbin -n default \
--from-file=key.pem=certs/casbin-key.pem \
--from-file=cert.pem=certs/casbin-crt.pem
```

7. After completing the above steps, you need to update the certificate directory in [main.go](#) and the [manifests](#) with the directory of the created `secret`.

Now, the server should be up and running, ready to validate requests made to K8s resources and enforce policies accordingly.

Admission Webhook for K8s

1. 概述 Casbin K8s-Gatekeeper

Casbin K8s-GateKeeper is a Kubernetes admission webhook that integrates Casbin as the Access Control tool. By using Casbin K8s-GateKeeper, you can establish flexible rules to authorize or intercept any operation on K8s resources, WITHOUT writing any piece of code, but only several lines of declarative configurations of Casbin models and policies, which are part of the Casbin ACL (Access Control List) language.

Casbin K8s-GateKeeper is developed and maintained by the Casbin community. The repository of this project is available here: <https://github.com/casbin/k8s-gatekeeper>

0.1 A Simple Example

For example, you don't need to write any code, but use the following lines of configuration to achieve this function: "Forbid images with some specified tags to be used in any deployments":

模型:

```
[request_definition]
r = obj

[policy_definition]
p = obj,eft

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

[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
== "deployments" && \
contain(split(accessWithWildcard(${OBJECT}.Spec.Template.Spec.Containers , "*",
"Image"),":",1) , p.obj)
```

策略:

```
p, "1.14.1", 否认
```

这些都是用常见的Casbin ACL语言编写的。如若你已阅读过相关章节，那就很容易理解。

Casbin K8s-Gatekeeper具有以下优势:

- 它简单易用。 Writing several lines of ACL is far better than writing lots of code.

- It allows hot updates of configurations. 您无需关闭整个插件来修改配置。
- It is flexible. Arbitrary rules can be made on any K8s resource, which can be explored with `kubectl gatekeeper`.
- It simplifies the implementation of K8s admission webhook, which is very complicated. You don't need to know what K8s admission webhook is or how to write code for it. All you need to do is to know the resource on which you want to put constraints and then write Casbin ACL. Everyone knows that K8s is complex, but by using Casbin K8s-Gatekeeper, your time can be saved.
- It is maintained by the Casbin community. Feel free to contact us if anything about this plugin confuses you or if you encounter any problems when trying this.

1.1 How Casbin K8s-Gatekeeper Works?

K8s-Gatekeeper is an admission webhook for K8s that uses [Casbin](#) to apply arbitrary user-defined access control rules to help prevent any operation on K8s that the administrator doesn't want.

Casbin是一个强大、高效的开放源码访问控制库。 它支持根据各种出入控制模式执行授权。 For more details about Casbin, see [Overview](#).

K8 中的许可webhooks 是 HTTP 回调，负责接收“许可请求”并执行相关程序。 In particular, K8s-Gatekeeper is a special type of admission webhook: 'ValidatingAdmissionWebhook', which can decide whether to accept or reject this admission request or not. 准入请求，是指特定的 K8 资源操作的 HTTP 请求(例如，创建/删除一个部署)。 For more about admission webhooks, see [K8s official documentation](#).

1.2 An Example Illustrating How It Works

For example, when somebody wants to create a deployment containing a pod running nginx (using kubectl or K8s clients), K8s will generate an admission request, which (if translated into YAML format) can be something like this:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 1
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.1
```

This request will go through the process of all the middleware shown in the picture, including our K8s-Gatekeeper. K8s-Gatekeeper can detect all the Casbin enforcers stored in K8s's etcd, which is created and maintained by the user (via `kubectl` or the Go client we provide). 每个执行器都有Casbin模型和Casbin政策。 The admission request will be processed by every enforcer, one by one, and only by passing all enforcers can a request be accepted by this K8s-Gatekeeper.

(If you do not understand what a Casbin enforcer, model, or policy is, see this document: [Get Started](#)).

For example, for some reason, the administrator wants to forbid the appearance of the image 'nginx:1.14.1' while allowing 'nginx:1.3.1'. An enforcer containing the following rule and policy can be created (We will explain how to create an enforcer, what these models and policies are, and how to write them in the following chapters).

Model:

```
[request_definition]
r = obj

[policy_definition]
p = obj,eft

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

[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
== "deployments" && \
access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image") == p.obj
```

Policy:

```
p, "nginx:1.13.1",allow
p, "nginx:1.14.1",deny
```

By creating an enforcer containing the model and policy above, the previous admission request will be rejected by this enforcer, which means K8s won't create this deployment.

2 安装 K8s-gatekeeper

There are three methods available for installing K8s-gatekeeper: External webhook, Internal webhook, and Helm.

备注

Note: These methods are only meant for users to try out K8s-gatekeeper and are not secure. If you wish to use it in a productive environment, please ensure that you read [Chapter 5. Advanced settings](#) and make

any necessary modifications before installation.

2.1 内部 webhook

2.1.1 Step 1: Build the image

For the internal webhook method, the webhook itself will be implemented as a service within Kubernetes. To create the necessary service and deployment, you need to build an image of K8s-gatekeeper. You can build your own image by running the following command:

```
docker build --target webhook -t k8s-gatekeeper .
```

This command will create a local image called 'k8s-gatekeeper:latest'.

① 备注

Note: If you are using minikube, please execute `eval $(minikube -p minikube docker-env)` before running 'docker build'.

2.1.2 第2步：为K8s-gatekeeper设置服务和部署

Run the following commands:

```
kubectl apply -f config/rbac.yaml  
kubectl apply -f config/webhook_deployment.yaml  
kubectl apply -f config/webhook_internal.yaml
```

This will start running K8s-gatekeeper, and you can confirm this by running `kubectl get pods`.

2.1.3 Step 3: Install CRD Resources for K8s-gatekeeper

Run the following commands:

```
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml  
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
```

2.2 内部 webhook

For the external webhook method, K8s-gatekeeper will be running outside of Kubernetes, and Kubernetes will access K8s-gatekeeper as it would access a regular website. Kubernetes has a mandatory requirement that the admission webhook must be HTTPS. For the purpose of trying out K8s-gatekeeper, we have provided a set of certificates and a private key (although this is not secure). 如果您想用自己的证书, 请参阅 [第五章。Advanced](#)

[settings](#) for instructions on adjusting the certificate and private key.

The certificate we provide is issued for 'webhook.domain.local'. So, modify the host (e.g., /etc/hosts) and point 'webhook.domain.local' to the IP address on which K8s-gatekeeper is running.

Then execute the following command:

```
go mod tidy
go mod vendor
go run cmd/webhook/main.go
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
kubectl apply -f config/webhook_external.yaml
```

2.3 Install K8s-gatekeeper via Helm

2.3.1 Step 1: Build the image

Please refer to [Chapter 2.1.1](#).

2.3.2 Helm installation

Run the command `helm install k8sgatekeeper ./k8sgatekeeper`.

3. 试用K8s-gatekeeper

3.1 创建Casbin 模型和策略

You have two methods to create a model and policy: via kubectl or via the go-client we provide.

3.1.1 通过 kubectl 创建/更新Casbin 模型和策略

In K8s-gatekeeper, the Casbin model is stored in a CRD resource called 'CasbinModel'. Its definition is located in `config/auth.casbin.org_casbinmodels.yaml`.

`example/allowed_repo/model.yaml` 中有示例。Pay attention to the following fields:

- `metadata.name`: the name of the model. This name MUST be the same as the name of the CasbinPolicy object related to this model, so that K8s-gatekeeper can pair them and create an enforcer.
- `spec.enable`: if this field is set to "false", this model (as well as the CasbinPolicy object related to this model) will be ignored.
- `spec.modelText`: a string that contains the model text of a Casbin model.

The Casbin Policy is stored in another CRD resource called 'CasbinPolicy', whose definition can be found in `config/auth.casbin.org_casbinpolicies.yaml`.

`example/allowed_repo/policy.yaml` 中有示例。Pay attention to the following fields:

- `metadata.name`: the name of the policy. This name MUST be the same as the name of the CasbinModel object related to this policy, so that K8s-gatekeeper can pair them and create an enforcer.
- `spec.policyItem`: a string that contains the policy text of a Casbin model.

After creating your own CasbinModel and CasbinPolicy files, use the following command to apply them:

```
kubectl apply -f <filename>
```

Once a pair of CasbinModel and CasbinPolicy is created, K8s-gatekeeper will be able to detect it within 5 seconds.

3.1.2 Create/Update Casbin Model and Policy via the go-client we provide

We understand that there may be situations where it is not convenient to use the shell to execute commands directly on a node of the K8s cluster, such as when you are building an automatic cloud platform for your corporation. Therefore, we have developed a go-client to create and maintain CasbinModel and CasbinPolicy.

The go-client library is located in `pkg/client`.

In `client.go`, we provide a function to create a client.

```
func NewK8sGateKeeperClient(externalClient bool) (*K8sGateKeeperClient, error)
```

The `externalClient` parameter determines whether K8s-gatekeeper is running inside the K8s cluster or not.

In `model.go`, we provide various functions to create, delete, and modify CasbinModel. You can find out how to use these interfaces in `model_test.go`.

In `policy.go`, we provide various functions to create, delete, and modify CasbiPolicy. You can find out how to use these interfaces in `policy_test.go`.

3.1.2 Try Whether K8s-gatekeeper Works

Suppose you have already created the exact model and policy in `example/allowed_repo`. Now, try the following command:

```
kubectl apply -f example/allowed_repo/testcase/reject_1.yaml
```

You should find that K8s will reject this request and mention that the webhook was the reason why this request is rejected. However, when you try to apply `example/allowed_repo/testcase/approve_2.yaml`, it will be accepted.

4. How to Write Model and Policy with K8s-gatekeeper

First of all, make sure you are familiar with the basic grammar of Casbin Models and Policies. If you are not, please read the [Get Started](#) section first. In this chapter, we assume that you already understand what Casbin Models and Policies are.

4.1 模型的请求定义

When K8s-gatekeeper is authorizing a request, the input is always an object: the Go object of the Admission Request. This means that the enforcer will always be used like this:

```
ok, err := enforcer.Enforce(admission)
```

where `admission` is an `AdmissionReview` object defined by K8s's official go api "`"k8s.io/api/admission/v1"`". You can find the definition of this struct in this repository: <https://github.com/kubernetes/api/blob/master/admission/v1/types.go>. For more information, you can also refer to <https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-request-and-response>.

Therefore, for any model used by K8s-gatekeeper, the definition of the `request_definition` should always be like this:

```
[request_definition]
r = obj
```

The name 'obj' is not mandatory, as long as the name is consistent with the name used in the `[matchers]` part.

4.2 模型匹配器

You are supposed to use the ABAC feature of Casbin to write your rules. However, the expression evaluator integrated in Casbin does not support indexing in maps or arrays(slices), nor the expansion of arrays. Therefore, K8s-gatekeeper provides various 'Casbin functions' as extensions to implement these features. If you still find that your demand cannot be fulfilled by these extensions, feel free to start an issue, or create a pull request.

If you are not familiar with Casbin functions, you can refer to [Function](#) for more information.

Here are the extension functions:

4.2.1 Extension functions

4.2.1.1 访问

Access is used to solve the problem that Casbin does not support indexing in maps or arrays. The example

`example/allowed_repo/model.yaml` demonstrates the usage of this function:

```
[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image") == p.obj
```

In this matcher, `access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image")` is equivalent to `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Image`, where `r.obj.Request.Object.Object.Spec.Template.Spec.Containers` is a slice.

Access can also call simple functions that have no parameters and return a single value. The example `example/container_resource_limit/model.yaml` demonstrates this:

```
[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
parseFloat(access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0,
"Resources","Limits","cpu","Value")) >= parseFloat(p.cpu) && \
parseFloat(access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0,
"Resources","Limits","memory","Value")) >= parseFloat(p.memory)
```

In this matcher, `access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Resources","Limits","cpu","Value")` is equivalent to `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Resources.Limits["cpu"].Value()`, where `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Resources.Limits` is a map, and `Value()` is a simple function that has no parameters and returns a single value.

4.2.1.2 访问Withildcard

Sometimes, you may have a demand like this: all elements in an array must have a prefix "aaa". However, Casbin does not support `for` loops. With `accessWithWildcard` and the "map/slice expansion" feature, you can easily implement such a demand.

For example, suppose `a.b.c` is an array `[aaa, bbb, ccc, ddd, eee]`, then the result of `accessWithWildcard(a, "b", "c", "*", "*")` will be a slice `[aaa, bbb, ccc, ddd, eee]`. By using the wildcard `*`, the slice is expanded.

Similarly, the wildcard can be used more than once. For example, the result of `accessWithWildcard(a, "b", "c", "*", "*", "*")` will be `[a.b.c[0][0], a.b.c[0][1], ..., a.b.c[1][0], a.b.c[1][1], ...]`.

4.2.1.3 Functions Supporting Variable-length Arguments

In the expression evaluator of Casbin, when a parameter is an array, it will be automatically expanded as a

variable-length argument. Utilizing this feature to support array/slice/map expansion, we have also integrated several functions that accept an array/slice as a parameter:

- `contain()`: accepts multiple parameters and returns whether any parameter (except the last parameter) equals the last parameter.
- `split(a,b,c...,sep,index)`: returns a slice that contains `[splits(a,sep)[index], splits(b,sep)[index], splits(a,sep)[index], ...]`.
- `len()`: returns the length of the variable-length argument.
- `matchRegex(a,b,c...,regex)`: returns whether all of the given parameters (`a`, `b`, `c`, ...) match the given regex.

Here is an example in `example/disallowed_tag/model.yaml`:

```
[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
contain(split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1) , p.obj)
```

Assuming that `accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image")` returns `["a:b", "c:d", "e:f", "g:h"]`, because `splits` supports variable-length arguments and performs the `splits` operation on each element, the element at index 1 will be selected and returned. Therefore, `split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1)` returns `["b", "d", "f", "h"]`. And `contain(split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1) , p.obj)` returns whether `p.obj` is contained in `["b", "d", "f", "h"]`.

4.2.1.2 Type Conversion Functions

- `ParseFloat()`: Parses an integer to a float (this is necessary because any number used in comparison must be converted into a float).
- `ToString()`: Converts an object to a string. This object must have a basic type of string (for example, an object of type `XXX` when there is a statement `type XXX string`).
- `IsNil()`: Returns whether the parameter is nil.

5. 高级设置

5.1 关于证书

In Kubernetes (k8s), it is mandatory that a webhook should use HTTPS. There are two approaches to achieve this:

- Use self-signed certificates (examples in this repository use this method)
- 使用普通证书

5.1.1 自签名证书

Using a self-signed certificate means that the Certificate Authority (CA) issuing the certificate is not one of the well-known CAs. Therefore, you must let k8s know about this CA.

Currently, the example in this repository uses a self-made CA, whose private key and certificate are stored in `config/certificate/ca.crt` and `config/certificate/ca.key` respectively. The certificate for the webhook is `config/certificate/server.crt`, which is issued by the self-made CA. The domains of this certificate are "webhook.domain.local" (for external webhook) and "casbin-webhook-svc.default.svc" (for internal webhook).

Information about the CA is passed to k8s via webhook configuration files. Both `config/webhook_external.yaml` and `config/webhook_internal.yaml` have a field called "CABundle", which contains a base64 encoded string of the CA's certificate.

In case you need to change the certificate/domain (for example, if you want to put this webhook into another namespace of k8s while using an internal webhook, or if you want to change the domain while using an external webhook), the following procedures should be followed:

1. Generate a new CA:

- Generate the private key for the fake CA:

```
openssl genrsa -des3 -out ca.key 2048
```

- Remove the password protection of the private key:

```
openssl rsa -in ca.key -out ca.key
```

2. Generate a private key for the webhook server:

```
openssl genrsa -des3 -out server.key 2048  
openssl rsa -in server.key -out server.key
```

3. Use the self-generated CA to sign the certificate for the webhook:

- 复制您系统的 openssl 配置文件临时使用。 You can find out the location of the config file by running `openssl version -a`, usually called `openssl.cnf`.
- In the config file:
 - Find the `[req]` paragraph and add the following line: `req_extensions = v3_req`
 - Find the `[v3_req]` paragraph and add the following line: `subjectAltName = @alt_names`

- Append the following lines to the file:

```
[alt_names]
DNS.2=<The domain you want>
```

Note: Replace 'casbin-webhook-svc.default.svc' with the real service name of your own service if you decide to modify the service name.

- Use the modified config file to generate a certificate request file:

```
openssl req -new -nodes -keyout server.key -out server.csr -config openssl.cnf
```

- Use the self-made CA to respond to the request and sign the certificate:

```
openssl x509 -req -days 3650 -in server.csr -out server.crt -CA ca.crt -CAkey ca.key -CAcreateserial -extensions v3_req -extensions SAN -extfile openssl.cnf
```

4. Replace the 'CABundle' field: Both `config/webhook_external.yaml` and `config/webhook_internal.yaml` have a field called "CABundle", which contains a base64 encoded string of the certificate of the CA. Update this field with the new certificate.
5. If you are using helm, similar changes need to be applied to the helm charts.

5.1.2 法律证书

If you use legal certificates, you do not need to go through all these procedures. Remove the "CABundle" field in `config/webhook_external.yaml` and `config/webhook_internal.yaml`, and change the domain in these files to the domain you own.

使用 Envoy 实现 Service Mesh 权限管理

[Envoy-authz](#) is a middleware for Envoy that performs external RBAC & ABAC authorization through casbin. This middleware uses [Envoy's external authorization API](#) via a gRPC server. This proxy can be deployed on any type of Envoy-based service mesh, such as Istio.

需求

- Envoy 1.17 以上版本
- Istio or any other type of service mesh
- gRPC 依赖项

Dependencies are managed using `go.mod`.

Working of the Middleware

- A client makes an HTTP request.
- The Envoy proxy sends the request to the gRPC server.
- The gRPC server authorizes the request based on casbin policies.
- If authorized, the request is forwarded; otherwise, it is denied.

The gRPC server is based on protocol buffer from [external_auth.proto](#) in Envoy.

```
// A generic interface for performing authorization checks on
incoming
// requests to a networked service.
service Authorization {
    // Performs an authorization check based on the attributes
associated with the
    // incoming request and returns a status of `OK` or not `OK`.
    rpc Check(v2.CheckRequest) returns (v2.CheckResponse);
}
```

From the above proto file, we need to use the `Check()` service in the authorization server.

用法

- Define the Casbin policies in the config files following this [guide](#).

You can verify/test your policies using the online [casbin-editor](#).

- Start the authentication server by running:

```
go build .
./authz
```

- Load the Envoy configuration:

```
envoy -c authz.yaml -l info
```

Once Envoy starts, it will intercept requests for the authorization process.

Integrating with Istio

To make this middleware work, you need to send custom headers containing usernames in the JWT token or headers. You can refer to the official [Istio documentation](#) for more information on modifying `Request Headers`.



>

管理

管理

Admin Portal

Casbin 管理门户网站

Casbin服务

Using Casbin as a Service

日志 & 错误处理

Logging and error handling in Casbin

前端使用

Casbin.js is a Casbin addon that facilitates your access-control management in the frontend application

Admin Portal

We provide a web-based portal called [Casdoor](#) for model management and policy management:

PML Model Editor X

[request_definition]
r = tenant, sub, obj, act, service

[policy_definition]
p = tenant, sub, obj, act, service, eft

[role_definition]
g = ...

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

[matchers]
m = r.tenant == p.tenant && g(r.sub, p.sub) && keyMatch(r.obj, p.obj) && (r.act == p.act || p.act == "*") && (r.service == p.service || p.service == "*")

Save

Rule Type	Tenant	User	Resource Path	Action	Service	Auth Effect	Option
p	tenant1	admin1	/*	*	*	allow	
p	tenant1	user12	/*	*	nova	allow	
p	tenant1	user13	/*	*	glance	allow	
g	user11	admin1					

Save

There are also third-party admin portal projects that use Casbin as an authorization engine. You can get started building your own Casbin service based on these projects.

[Go](#) [Java](#) [Node.js](#) [Python](#) [PHP](#)

Project	作者	前端	后端	描述
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-	@piexlmax	Vue +	Gin	基于 Gin + GORM + Vue

Project	作者	前端	后端	描述
admin		Element UI		
gin-admin	@LyricTian	React + Ant Design	Gin	RBAC脚手架基于Gin + GORM + Casbin + Ant设计反应
go-admin	@hequan2017	无	Gin	基于Gin + GORM + JWT + RBAC (Casbin) 的 RESTful API网关
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 (前端, 后端)	@RealLiuSha	Vue 2.x + Element UI	Echo	基于 Echo + Gorm + Casbin + Uber-FX 的管理门户网站
Spec-	@atul-	无	Mux	基于 Casbin + MongoDB

Project		作者		前端	后端	描述
Center		wankhade				的 Golang RESTful 平台
Project		作者		前端	后端	描述
spring-boot-web		@BazookaW	无	SpringBoot		基于 SpringBoot 2.0 + MyBatisPlus + Casbin 的管理门户网站
Project		作者		前端	后端	描述
node-mysql-rest-api		@JoemaNequinto	无		Express	一个使用 Express, Sequelize, JWT 和 Casbin 在Node.js 建造RESTful APIs 微型服务的应用。
Casbin-Role-Mgt-Dashboard-RBAC		@alikhan866		React + Material UI	Express	带有Enforcer可以检查过程中执行结果且对新手友好的RBAC 管理
Project		作者		前端	后端	描述
fastapi-mysql-generator		@CoderCharm	无		FastAPI	FastAPI + MySQL + JWT + Casbin
FastAPI-		@xingxingzaixian	无		FastAPI	FastAPI +

Project	作者	前端	后端	描述
MySQL-Tortoise-Casbin				MySQL + Tortoise + Casbin
openstack-policy-editor	Casbin	Bootstrap	Django	Casbin 的 Web 界面
Project	作者	前端	后端	描述
Tadmin	@leeqvip	AmazeUI	ThinkPHP	基于 ThinkPHP 的一款非侵入式后台开发框架
video.tinywan.com	@Tinywanner	LayUI	ThinkPHP	基于 ThinkPHP5 + ORM + JWT + RBAC (Casbin) 的 RESTful API 网关
laravel-casbin-admin	@pl1998	Vue + Element UI	Laravel	基于 vue-element-admin 和 Laravel 的 RBAC 许可管理系统
larke-admin (前端, 后端)	@deatil	Vue 2 + Element	Laravel 8	基于 Laravel 8, JWT 和 RBAC

Project	作者	前端	后端	描述
		UI		的门户网站
hyperf-vuetify-admin	@TragicMale	Vue + Vuetify 2.x	Hyperf	基于Hyperf、 Vuetify和 Casbin的管理 门户网站

Casbin服务

How to Use Casbin as a Service?

名称	描述
Casbin服务	The official "Casbin as a Service" solution based on gRPC . Both Management API and RBAC API are provided.
middleware-acl	基于 Casbin 的 RESTful 访问控制插件。
Buttress	基于 Casbin 访问控制的服务解决方案。
auth-server	验证服务器校对服务。

日志 & 错误处理

日志

Casbin uses the built-in `log` to print logs to the console by default, like:

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

Logging is not enabled by default. 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。

ⓘ 备注

We already support logging the model, enforce request, role, and policy in Golang. 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。The pycasbin package makes a call to `logging.getLogger()` to set the logger. 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。If no logging is initialized within the parent application, you will not see any log messages from pycasbin. At the same time, when you enable log in pycasbin, it will use the [default log configuration](#). For other pycasbin extensions, you can refer to the [Django logging docs](#) if you are a Django user. For other Python users, you should refer to the [Python logging docs](#) to configure the logger. :::

Use different loggers for different enforcers

Every enforcer can have its own logger to log information, and it can be

changed at runtime.

And you can use a proper logger via the last parameter of `NewEnforcer()`.

If you are using this way to initialize your enforcer, you don't need to use the enabled parameter because the priority of the enabled field in the logger is higher.

```
// 设置默认记录器作为执行器e1的记录器。  
// This operation can also be seen as changing the logger  
// of e1 at runtime.  
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)  
  
// Set your logger when initializing enforcer e3.  
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。

Go PHP

记录器	作者	描述
<u>Default logger (built-in)</u>	Casbin	默认使用golang日志。

记录器	作者	描述
Zap logger	Casbin	Using zap , provide json encoded log and you can customize more with your own zap-logger.
记录器	作者	描述
psr3-bridge 记录器	Casbin	提供一个 PSR-3 兼容桥。

如何编写一个记录器

您的记录器应该实现 [Logger](#) 接口。

接口名	实现要素	描述
EnableLog()	必须	Control whether to print the message.
IsEnabled()	必须	显示当前日志启用的状态。
LogModel()	必须	Log info related to the model.
LogEnforce()	必须	Log info related to enforcing.
LogRole()	必须	Log info related to the role.
LogPolicy()	mandatory	Log info related to the policy.

您可以将您的自定义 [记录器](#) 传给 [Enforcer.SetLogger\(\)](#) 函数。

Here is an example of how to customize a logger for Golang:

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

// 默认日志是使用golang日志的日志实现。
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())
}
```

错误处理

Errors or panics may occur when you use Casbin for reasons like:

1. Invalid syntax in the model file (.conf).
2. Invalid syntax in the policy file (.csv).
3. Custom errors from storage adapters, e.g., MySQL fails to connect.
4. Casbin的bug。

There are five main functions you may need to be aware of for errors or panics:

Function	异常时表现
<u>NewEnforcer()</u>	Returns an error
<u>LoadModel()</u>	Returns an error
<u>LoadPolicy()</u>	Returns an error
<u>SavePolicy()</u>	Returns an error
<u>Enforce()</u>	Returns an error

`NewEnforcer()` calls `LoadModel()` and `LoadPolicy()` internally. So you don't have to call the latter two when using `NewEnforcer()`.

⋮

Enable and disable

可以通过 `Enforcer.EnableEnforce()` 函数禁用执行器。当它被禁用时, `Enforcer.Enforce()` 将总是返回 `true`。Other operations like adding or removing policies are not affected. 下面是一个示例:

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

// 将会返回false
// 默认情况下enforcer是启用的
e.Enforce("non-authorized-user", "data1", "read")

// Disable the enforcer at runtime.
e.EnableEnforce(false)

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

// 打开enforcer
e.EnableEnforce(true)

// 将会返回false
e.Enforce("non-authorized-user", "data1", "read")
```


前端使用

Casbin.js is a Casbin addon that facilitates your access-control management in the frontend application.

安装

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

或者

```
yarn add casbin.js
```

前端中间件

中间件	类型	作者	描述
react-authz	React	Casbin	Casbin.js 的 React 包装器
rbac-react	React	@daobeng	React 基于角色的访问控制, 使用 HOCs, CASL 和 Casbin.js

中间件	类型	作者	描述
vue-authz	Vue	Casbin	Casbin.js 的 Vue 包装器
angular-authz	Angular	Casbin	Casbin.js的Angular包装器

快速入门

You can use the `manual` mode in your frontend application and set the permissions whenever you wish.

```
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");
```

Now we have an authorizer, `authorizer`. We can get permission rules from it by using the `authorizer.can()` and `authorizer.cannot()` APIs. 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");
```

The `cannot()` API is used in the same way:

```
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
```

In the code above, the `success` variable in the parameters means the request gets the result without throwing an error and doesn't mean that the permission rule is `true`. The `failed` variable is also unrelated to the permission rules. 只有在请求过程中出现错误时才有意义。

You can refer to our [React example](#) to see a practical usage of Casbin.js.

Permission Object

Casbin.js will accept a JSON object to manipulate the corresponding permission of a visitor. For example:

```
{
    "read": ["data1", "data2"],
    "write": ["data1"]
}
```

The permission object above shows that the visitor can `read` the `data1` and `data2` objects, while they can only `write` the `data1` objects.

高级用法

Casbin.js provides a perfect solution for integrating your frontend access-control management with your backend Casbin service.

Use the `auto` mode and specify your endpoint when initializing the Casbin.js `Authorizer`, it will automatically sync the permission and manipulate the frontend status.

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

// Set your backend Casbin service URL
const authorizer = new casbinjs.Authorizer(
    'auto', // mode
    {endpoint: 'http://your_endpoint/api/casbin'}
);

// Set your visitor.
// 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", "data1");
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前端。在你的 API 控制器中，调用 `CasbinJs GetUserPermission` 以创建权限对象。下面是一个 Beego 框架的示例：

① 备注

注意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。 (The key is "casbin_subject")  
    visitor := c.Input().Get("casbin_subject")  
    // `e` is an initialized instance of Casbin Enforcer  
    c.Data["perm"] = casbin.CasbinJsGetPermissionForUser(e,  
visitor)  
    // Pass the data to the frontend.  
    c.ServeJSON()  
}
```

① 备注

Currently, the `CasbinJsGetPermissionForUser` API is only supported in Go Casbin and Node-Casbin. If you want this API to be supported in other languages, please [raise an issue](#) or leave a comment below.

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<object>)

Check if the user can perform `action` on all objects in `objects`.

canAny(action: string, objects: Array<object>)

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

People may wonder about the difference between Node-Casbin and Casbin.js. In a word, Node-Casbin is the core of Casbin implemented in the NodeJS environment, and it's normally used as an access-controlling management toolkit at the server ends. Casbin.js is a frontend library that helps you use Casbin to authorize your webpage users at the client side.

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：

1. When someone turns on the client, the enforcer will be initialized, and it will pull all the policies from the backend persistent layers. 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。
2. Loading all policies to the client side could bring security risks.
3. It is difficult to separate the client and server as well as facilitate agile development.

We need a tool that eases the process of using Casbin at the frontend. Actually, the core of Casbin.js is the manipulation of the current user's permission at the client side. 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 后端服务同步权限。After having the permission data, developers can use Casbin.js interfaces to manage the behavior of the user at the frontend side.

Casbin.js avoids the two problems mentioned above: Casbin service will no longer be pulled up repeatedly, and the size of passing messages between the client and the server is reduced. We also avoid storing all the policies at the frontend. The user can only access their own permission, but has no knowledge about the access-control model and other users' permissions. 此外，Casbin.js 还能有效地在授权管理方面分离客户端和服务端。



>

编辑器

编辑器

**Online Editor**

Writing Casbin model and policy in a web browser

**IDE 插件**

Casbin IDE plugins

Online Editor

You can also use the [online editor](#) to write your Casbin model and policy in your web browser. It provides functionality such as "syntax highlighting" and "code completion", just like an IDE for a programming language.

Usage Pattern

If you are using "RBAC with pattern" or "RBAC with all pattern", the pattern matching function is specified in the lower left corner.

The screenshot shows the Casbin Online Editor interface. On the left, a code editor displays Casbin configuration code. A red arrow points from the text 'keyMatch' in line 12 to the 'Request' panel on the right. The code editor contains the following lines:

```
12     *  
13         matchingDomainForGFunction:  
14             'keyMatch'  
15         */  
16         matchingForGFunction:  
17             'keyMatch2',  
18     };  
19 })();
```

On the right, a 'Request' panel shows three numbered items:

- 1 /book/1
- 2 /book/1
- 3

If you want to write the equivalent code, you need to specify the pattern matching function through the relevant API. Refer to [RBAC with Pattern](#) for more information.

备注

编辑器基于 [node-casbin](#)。 Due to the synchronization delay between different versions of Casbin, the authentication result of the "editor" may differ from the authentication result of the Casbin version you are using. If you encounter any issues, please submit them to the Casbin repository you are using.

IDE 插件

We offer plugins for the following IDEs:

JetBrains IDEs

- Download: <https://plugins.jetbrains.com/plugin/14809-casbin>
- Source code: <https://github.com/will7200/casbin-idea-plugin>

VSCode

- Source code: <https://github.com/casbin/casbin-vscode-plugin>



>

更多

更多

使用者

Casbin's Adopters

参与贡献

Contributing to Casbin

隐私政策

Casbin 网站隐私政策

服务条款

Casbin 服务条款



Refund Policy

Casbin Website Refund Policy

使用者

Direct Integration

[Go](#) [Java](#) [Node.js](#) [Python](#)

名称	描述	模型	策略
VMware Harbor	VMware的开源可信云本地注册表项目，用于存储、签名和扫描内容。	Code	Beego ORM
Intel RMD	英特尔的资源管理守护进程。	.conf	.csv
VMware Dispatch	用于部署和管理无服务器风格应用程序的框架。	Code	Code
Skydive	一个开源的实时网络拓扑和协议分析器。	Code	.csv
Zenpress	用Golang编写的CMS系统。	.conf	Gorm
Argo CD	为Kubernetes持续提供的GitOps。	.conf	.csv
Muxi Cloud	PaaS of Muxi Cloud, an easier way to manage Kubernetes clusters.	.conf	Code
EngineerCMS	CMS管理工程师的知识。	.conf	SQLite

名称	描述	模型	策略
Cyber Auth API	一个Golang身份验证API项目。	.conf	.csv
Metadata DB	BB归档元数据数据库。	.conf	.csv
Qilin API	ProtocolONE's licenses management tool for game content.	Code	.csv
Devtron Labs	Kubernetes软件发送Workflow。	.conf	Xorm
名称	描述	模型	策略
lighty.io	OpenDaylight's solution for SDN controllers.	README	无
名称	描述	模型	策略
Notadd	A micro-service development architecture based on Nest.js.	.conf	DB adapter
ARC API	A Catalog of Microservices based on Loopback Created by SourceFuse.	Usage	Provider
名称	描述	模型	策略
dtrace	EduScaled的跟踪系统。	Commit	无

Integration via Plugin

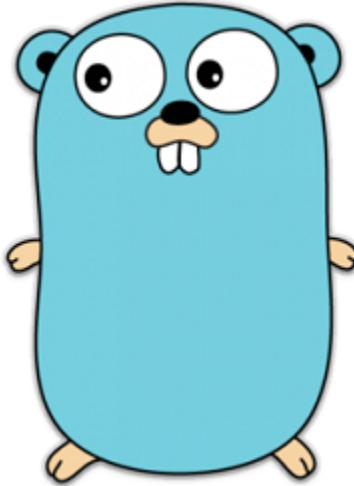
名称	描述	插件	模型	策略
Docker	全球领先的软件容器平台	casbin-authz-plugin (Docker 推荐)	.conf	.csv
Gobis	用go编写的Orange的轻量级API网关	casbin	Code	请求

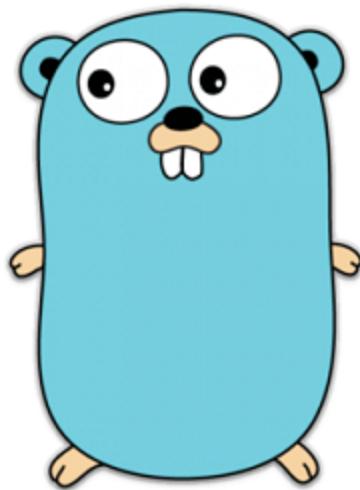
参与贡献

Casbin is a powerful authorization library that supports access control models with implementations in many programming languages. If you are proficient in any programming language, you can contribute to the development of Casbin. New contributors are always welcome.

Currently, there are two main types of projects:

- **Algorithms-oriented projects** - These projects involve implementing algorithms in different programming languages. Casbin supports a wide range of languages, including Golang, Java, C++, Elixir, Dart, and Rust, along with their related products.

 The Gopher logo, a blue cartoon gopher character with large white eyes and a small orange nose, standing next to the word "Go".	 The Java logo, a stylized orange flame or coffee cup icon.
Casbin	jCasbin



Go



Production-ready

Production-ready



PyCasbin

Casbin.NET

Production-ready

Production-ready

- Application-oriented projects - These projects are related to applications built on top of Casbin.

Project	Demo	详情	Skill Stacks
Casdoor	Casdoor	Casdoor is a UI-first centralized authentication/Single-Sign-On (SSO) platform based on OAuth 2.0/ OIDC.	JavaScript + React 和 Golang + Beego + SQL
Casnnode	Casbin 论坛	Casnnode is a next-generation forum software.	JavaScript + React 和 Golang + Beego + SQL
Casbin-OA	OA system	Casbin-OA is an official manuscript processing, evaluation, and display system for Casbin technical writers.	JavaScript + React 和 Golang + Beego + MySQL
Casbin Editor	Casbin Editor	Casbin-editor is a web-based editor for Casbin models and policies.	TypeScript + React

Getting Involved

There are many ways to contribute to Casbin. Here are some ideas to get started:

- Use Casbin and report issues! When using Casbin, report any issues you encounter to help promote the development of Casbin. Whether it's a bug or a

proposal, filing an issue on [GitHub](#) is recommended. However, it would be better to have a discussion first on [Discord](#) or [GitHub Discussions](#) before filing an issue.

Note: When reporting an issue, please use English to describe the details of your problem.

- **Help with documentation!** Contributing to the documentation is a good starting point for your contribution.
- **Help solve issues!** We have prepared a table containing easy tasks suitable for beginners, with different levels of challenges labeled with different tags. You can check the table [here](#).

合并请求

Casbin uses GitHub as its development platform, so pull requests are the main way to contribute.

Before opening a pull request, there are a few things you need to know:

- Explain why you are sending the pull request and what it will do for the repository.
- Make sure the pull request does only one thing. If there are multiple changes, please split them into separate pull requests.
- If you are adding new files, please include the Casbin license at the top of the new file(s).

```
// Copyright 2021 The casbin Authors. All Rights Reserved.
```

- In projects like [Casdoor](#), [Casnodel](#), and [Casbin OA](#), you may need to set up a demo to show the maintainer how your pull request helps with the development of the project.
- When opening a pull request and committing your contribution, it is recommended to use semantic commits with the following format:
`<type>(<scope>): <subject>`. The `<scope>` is optional. For more detailed usage, please refer to [Conventional Commits](#).

许可协议

By contributing to Casbin, you agree that your contributions will be licensed under the Apache License.

隐私政策

您的隐私对我们很重要。 It is Casbin's policy to respect your privacy regarding any information we may collect from you across our [docs website](#), as well as other sites we own and operate.

我们只是在我们真正需要你的信息向你提供服务时才要求你提供您的个人信息。我们在你知情和同意的情况下，以公正和合法的方式加以收集。 We also let you know why we are collecting it and how it will be used.

我们只在必要的时候保留您的信息来为您提供您请求的服务 The data we store will be protected within commercially acceptable means to prevent loss and theft, as well as unauthorized access, disclosure, copying, use, or modification.

We do not share any personally identifying information publicly or with third-parties, except when required to by law.

我们的网站可能会链接到不属于我们管理的外部网站。 Please be aware that we have no control over the content and practices of these sites and cannot accept responsibility or liability for their respective privacy policies.

您可以自由拒绝我们的个人信息请求。 我的理解是，我们可能无法向你提供你所需要的一些服务。

Your continued use of our website will be regarded as acceptance of our practices regarding privacy and personal information. 如果您有任何关于我们如何处理用户数据和个人信息的问题，请随时联系我们。

This policy is effective as of 29th June 2020.

服务条款

1. Terms

By accessing the website at <https://casbin.org>, you are agreeing to be bound by these terms of service, all applicable laws and regulations, and agree that you are responsible for compliance with any applicable local laws. If you do not agree with any of these terms, you are prohibited from using or accessing this site. The materials contained in this website are protected by applicable copyright and trademark law.

2. Use License

- a. Permission is granted to temporarily download one copy of the materials (information or software) on Casbin's website for personal, non-commercial transitory viewing only. This is the grant of a license, not a transfer of title, and under this license you may not:
 - i. 修改或复制此资料
 - ii. 将此资料商用，或者用于公开展示（无论是否商用）；
 - iii. 尝试反编译，或反工程化 Casbin 网站上的任何软件。
 - iv. 从资料中移除任何版权或所有权标记
 - v. transfer the materials to another person or "mirror" the materials on any other server.
- b. This license shall automatically terminate if you violate any of these restrictions and may be terminated by Casbin at any time. Upon terminating your viewing of these materials or upon the termination of this license, you must destroy any downloaded materials in your possession whether in

electronic or printed format.

3. Disclaimer

- a. The materials on Casbin's website are provided on an 'as is' basis. Casbin makes no warranties, expressed or implied, and hereby disclaims and negates all other warranties including, without limitation, implied warranties or conditions of merchantability, fitness for a particular purpose, or non-infringement of intellectual property or other violation of rights.
- b. Further, Casbin does not warrant or make any representations concerning the accuracy, likely results, or reliability of the use of the materials on its website or otherwise relating to such materials or on any sites linked to this site.

4. Limitations

In no event shall Casbin or its suppliers be liable for any damages (including, without limitation, damages for loss of data or profit, or due to business interruption) arising out of the use or inability to use the materials on Casbin's website, even if Casbin or a Casbin authorized representative has been notified orally or in writing of the possibility of such damage. Because some jurisdictions do not allow limitations on implied warranties, or limitations of liability for consequential or incidental damages, these limitations may not apply to you.

5. Accuracy of materials

The materials appearing on Casbin's website could include technical, typographical, or photographic errors. Casbin does not warrant that any of the materials on its website are accurate, complete or current. Casbin may make changes to the materials contained on its website at any time without notice. However Casbin does not make any commitment to update the

materials.

6. Links

Casbin has not reviewed all of the sites linked to its website and is not responsible for the contents of any such linked site. The inclusion of any link does not imply endorsement by Casbin of the site. Use of any such linked website is at the user's own risk.

7. Modifications

Casbin may revise these terms of service for its website at any time without notice. By using this website you are agreeing to be bound by the then current version of these terms of service.

8. Governing Law

These terms and conditions are governed by and construed in accordance with the laws of San Francisco, CA and you irrevocably submit to the exclusive jurisdiction of the courts in that State or location.

Refund Policy

In most cases, payments for Casbin subscriptions are not refundable.

If you have an issue with your account or think there has been an error in billing, please [contact support](#) for assistance.