Overview

Casbin是一个强大且高效的开源访问控制库，支持各种访问控制模型，用于在全局范围内执行授权。

执行一组规则就像在***策略***文件中列出主题、对象和期望的允许操作（或根据您的需要的任何其他格式）一样简单。 这在所有使用Casbin的流程中都是同义的。 开发者/管理员对布局、执行和授权条件的控制是完全的，这些都是通过***模型***文件设置的。 Casbin提供了一个*Enforcer*，用于根据提供给Enforcer的策略和模型文件验证传入的请求。

Casbin支持的语言

Casbin为各种编程语言提供支持，准备在任何项目和工作流中集成：

Casbin	jCasbin
生产就绪	生产就绪

PyCasbin	Casbin.NET
生产就绪	生产就绪

不同语言的功能集

我们一直在尽我们最大的努力，使Casbin在所有语言中都有相同的功能集。 然而，现实并不那么美好。

特性	Go	Java	Node.js	PHP	Python	C#	Delphi	Rust	C++	Lua	Dart	Elixir
Enforcement	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅
RBAC	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅
ABAC	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅
Scaling ABAC (`eval()`)	✅	✅	✅	✅	✅	✅	❌	✅	✅	✅	✅	✅
Adapter	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	❌
Management API	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅	✅
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	✅	✅	✅	❌	✅	❌	❌	✅	❌	❌	❌	❌
'in' of matcher	✅	✅	✅	✅	✅	❌	✅	❌	❌	❌	✅	✅

注意 - ✅ 对于观察者或角色管理器，只意味着在核心库中有接口。 这并不能表明是否有观察者或角色管理器的实现可用。

什么是Casbin?

Casbin是一个授权库，可以在我们希望某个`对象`或实体被特定用户或`主体`访问的流程中使用。 访问类型，即`动作`，可以是_读取_，*写入*，*删除*，或者由开发者设置的任何其他动作。 这就是Casbin最广泛使用的方式，它被称为“标准”或经典的`{ 主体, 对象, 动作 }`流程。

Casbin能够处理许多复杂的授权场景，而不仅仅是标准流程。 可以添加角色（RBAC），属性（ABAC）等。

Casbin做什么

1. 在经典的`{ 主体, 对象, 动作 }`形式或者你定义的自定义形式中执行策略。 支持允许和拒绝授权。
2. 处理访问控制模型及其策略的存储。
3. 管理角色-用户映射和角色-角色映射（也称为RBAC中的角色层次）。
4. 支持内置的超级用户，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. 提供多个内置操作符以支持规则匹配。 例如，`keyMatch`可以将资源键`/foo/bar`映射到模式`/foo*`。

Casbin 不做什么

1. 身份验证（也就是在用户登录时验证`用户名`和`密码`）
2. 管理用户或角色列表。

对于项目来说，管理他们的用户、角色或密码列表更方便。 用户通常有他们的密码，而Casbin并未设计为密码容器。 然而，Casbin存储了RBAC场景下的用户-角色映射。

Get Started

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
```

在您的项目的`composer.json`中需要这个包以下载该包:

```
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
```

Casbin4D以包的形式提供（目前适用于Delphi 10.3 Rio），您可以在IDE中安装它。 然而，没有可视化组件，这意味着您可以独立于包使用单元。 只需在您的项目中导入单元（假设您不介意它们的数量）。

```
luarocks install casbin
```

如果您收到错误消息：“您的用户在/usr/local/lib/luarocks/rocks中没有写权限”，您可能需要以特权用户身份运行命令，或者使用`--local`的本地树。 要修复错误，您可以在命令后面添加`--local`，如下所示:

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

新建一个Casbin执行器

Casbin使用配置文件来定义访问控制模型。

有两个配置文件：`model.conf`和`policy.csv`。 `model.conf`存储访问模型，而`policy.csv`存储具体的用户权限配置。 Casbin的使用非常直接。 我们只需要创建一个主要结构：**执行器**。 在构建这个结构时，`model.conf`和`policy.csv`将被加载。

换句话说，要创建一个Casbin执行器，您需要提供一个模型和一个适配器。

Casbin提供了一个您可以使用的文件适配器。 查看适配器以获取更多信息。

- 使用模型文件和默认的文件适配器的示例：

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

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

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

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

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

```
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() {
    // Create an Enforcer
    casbin::Enforcer e("path/to/model.conf", "path/to/
policy.csv");

    // your code ..
}
```

```
var
  casbin: ICasbin;
```

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

// If you use async_td as async executor
#[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(())
}

// If you use tokio as async executor
#[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(())
}
```

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

- 使用模型文本和其他适配器:

Go Python

```go
import (
    "log"

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

```python
import casbin
import casbin_sqlalchemy_adapter


# Use SQLAlchemy Casbin adapter with SQLLite DB
adapter = casbin_sqlalchemy_adapter.Adapter('sqlite:///test.db')

# Create a config model policy
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
    """)

# Create enforcer from adapter and config policy
e = casbin.Enforcer('rbac_example_model.conf', adapter)
```

检查权限

在访问发生之前，在您的代码中添加一个执行钩子：

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

Rust Lua

```go
sub := "alice" // the user that wants to access a resource.
obj := "data1" // the resource that is going to be accessed.
act := "read" // the operation that the user performs on the
resource.

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

if err != nil {
    // handle err
}

if ok == true {
    // permit alice to read data1
} else {
    // deny the request, show an error
}

// You could use BatchEnforce() to enforce some requests in
batches.
// This method returns a bool slice, and this slice's index
corresponds to the row index of the two-dimensional array.
// e.g. results[0] is the result of {"alice", "data1", "read"}
results, err := e.BatchEnforce([][]interface{}{{"alice",
"data1", "read"}, {"bob", "data2", "write"}, {"jack", "data3",
"read"}})
```

```java
String sub = "alice"; // the user that wants to access a
resource.
String obj = "data1"; // the resource that is going to be
accessed.
```

```
const sub = 'alice'; // the user that wants to access a
resource.
const obj = 'data1'; // the resource that is going to be
accessed.
const act = 'read'; // the operation that the user performs on
the resource.

if ((await e.enforce(sub, obj, act)) === true) {
    // permit alice to read data1
} else {
    // deny the request, show an error
}
```

```
$sub = "alice"; // the user that wants to access a resource.
$obj = "data1"; // the resource that is going to be accessed.
$act = "read"; // the operation that the user performs on the
resource.

if ($e->enforce($sub, $obj, $act) === true) {
    // permit alice to read data1
} else {
    // deny the request, show an error
}
```

```
sub = "alice"  # the user that wants to access a resource.
obj = "data1"  # the resource that is going to be accessed.
act = "read"  # the operation that the user performs on the
resource.

if e.enforce(sub, obj, act):
    # permit alice to read data1
    pass
else:
    # deny the request, show an error
    pass
```

```
var sub = "alice";  # the user that wants to access a resource.
var obj = "data1";  # the resource that is going to be accessed.
var act = "read";  # the operation that the user performs on
the resource.

if (await e.EnforceAsync(sub, obj, act))
{
    // permit alice to read data1
}
else
{
    // deny the request, show an error
}
```

```
    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 is super happy as she can read data1
else
  // Alice is sad
```

```
  let sub = "alice"; // the user that wants to access a
resource.
  let obj = "data1"; // the resource that is going to be
accessed.
  let act = "read"; // the operation that the user performs on
the resource.

  if e.enforce((sub, obj, act)).await? {
      // permit alice to read data1
  } else {
      // error occurs
  }
```

```
if e:enforce("alice", "data1", "read") then
   -- permit alice to read data1
else
   -- deny the request, show an error
end
```

Casbin还提供了运行时权限管理的API。 例如，您可以如下获取分配给用户的所有角色：

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")
```

查看管理API和RBAC API以获取更多用法。

请参考测试用例以获取更多用法。

How It Works

在Casbin中，访问控制模型被抽象为基于PERM**元模型（策略，效果，请求，匹配器）**的CONF文件。 为项目切换或升级授权机制就像修改配置一样简单。 您可以通过组合可用模型来定制自己的访问控制模型。 例如，您可以在一个模型内部将RBAC角色和ABAC属性结合在一起，并共享一套策略规则。

PERM模型由四个基础部分组成：策略，效果，请求和匹配器。 这些基础部分描述了资源和用户之间的关系。

Request

定义请求参数。 基本请求是一个元组对象，至少需要一个主体（被访问实体），对象（被访问资源）和动作（访问方法）。

例如，请求定义可能看起来像这样：`r={sub,obj,act}`

此定义指定了访问控制匹配函数所需的参数名称和顺序。

Policy

定义访问策略的模型。 它指定了策略规则文档中字段的名称和顺序。

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

注意：如果未定义eft（策略结果），则不会读取策略文件中的结果字段，匹配策略结果将默认允许。

Matcher

定义请求和策略的匹配规则。

例如：`m = r.sub == p.sub && r.act == p.act && r.obj == p.obj` 这个简单而常见的匹配规则意味着，如果请求的参数（实体，资源和方法）等于策略中找到的那些，那么返回策略结果（`p.eft`）。 策略的结果将保存在`p.eft`中。

Effect

对匹配器的匹配结果进行逻辑组合判断。

例如：`e = some(where(p.eft == allow))`

这个语句意味着，如果匹配策略结果`p.eft`有（一些）允许的结果，那么最终结果为真。

让我们看另一个例子：

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

这个例子组合的逻辑意义是：如果有一个策略匹配到允许的结果，并且没有策略匹配到拒绝的结果，结果为真。 换句话说，当匹配策略都是允许时，结果为真。 如果有任何拒绝，两者都为假（更简单地说，当允许和拒绝同时存在时，拒绝优先）。

Casbin中最基本和最简单的模型是ACL。 ACL的模型CONF如下：

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

# Policy definition
```

ACL模型的一个示例策略为：

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

这意味着：

- alice可以读取data1
- bob可以写入data2

我们还支持通过在末尾添加"来进行多行模式：

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

此外，如果您正在使用ABAC，您可以尝试如下例子中的'in'操作符，该例子是Casbin golang版本的（jCasbin和Node-Casbin还不支持）：

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

但是你**必须**确保数组的长度**超过** 1，否则会引发难以恢复的错误（panic）。

对于更多的操作符，你可以看一下govaluate。

Tutorials

在阅读之前，请注意，一些教程是针对Casbin的模型的，并适用于不同语言的所有Casbin实现。 其他一些教程是特定于语言的。

我们的论文

- PML: 基于解释器的Web服务访问控制策略语言

本文深入探讨了Casbin的设计细节。 如果您在论文中引用Casbin/PML，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}
}
```

- 基于元模型的访问控制策略规范语言（中文）

这是另一篇发表在《软件学报》上的长篇论文。 不同格式（Refworks，EndNote等）的引文 可以在以下位置找到：(另一版本) 基于元模型的访问控制策略规范语言（中文）

视频

- 一个安全的保险库 - 使用Casbin实现授权中间件 - JuniorDevSG
- 基于Casbin的微服务架构中的用户权限共享（俄语）

- Nest.js - Casbin RESTful RBAC 授权中间件
- Gin Tutorial Chapter 10: Learn Casbin basic models in 30 minutes (in Chinese)
- Gin Tutorial Chapter 11: Coding, API and custom function in Casbin (in Chinese)
- Gin + Casbin：动手学习权限（中文）
- jCasbin基础：一个简单的RBAC示例（中文）
- 基于Casbin的Golang的RBAC（中文）
- 学习Gin + Casbin（1）：开场和概述（中文）
- ThinkPHP 5.1 + Casbin：介绍（中文）
- ThinkPHP 5.1 + Casbin：RBAC授权（中文）
- ThinkPHP 5.1 + Casbin：RESTful和中间件（中文）
- PHP-Casbin快速入门（中文）
- ThinkPHP 5.1 + Casbin：如何使用自定义匹配函数（中文）
- Webman + Casbin：如何使用Webman Casbin插件（中文）

PERM元模型（策略，效果，请求，匹配器）

- 理解不同访问控制模型配置的Casbin
- 在Casbin中使用PERM建模授权
- 使用Casbin设计灵活的权限系统
- 使用访问控制列表进行授权
- 使用PERM和Casbin进行访问控制（波斯语）
- RBAC? ABAC? .. PERM! New Way of Authorization for Cloud-Based Web Services and Apps (in Russian)](https://habr.com/ru/articles/539778/)
- Practice & Examples of Flexible Authorization Using Casbin & PERM (in Russian)
- 使用Casbin进行权限管理（中文）
- Casbin分析（中文）

- 系统权限的设计（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）
- 使用Casbin进行权限评估（中文）
- Casbin：Go的每日库（中文）

Go Java Node.js PHP .NET Rust Lua

HTTP和RESTful

- 使用Casbin在Go中进行基于角色的HTTP授权 (或 中文翻译)

观察者

- 通过Casbin Watcher进行RBAC分布式同步（中文）

Beego

- 使用Casbin与Beego：1. 开始并测试（中文）
- 使用Casbin与Beego：2. 策略存储（中文）
- 使用Casbin与Beego：3. 策略查询（中文）
- 使用Casbin与Beego：4. 策略更新（中文）
- 使用Casbin与Beego：5. 策略更新（续）（中文）

Gin

- 在Golang项目中使用Casbin进行授权
- 教程：将Gin与Casbin集成
- 通过 Pipeline 在 K8s 上实施政策
- 在Gin应用中使用JWT和Casbin进行身份验证和授权
- 使用Go的后端API：1. 基于JWT的身份验证（中文）

- 使用Go的后端API：2. 基于Casbin的授权（中文）
- 使用Gin和GORM的Go的授权库Casbin（日语）

Echo

- 使用Casbin进行Web授权

Iris

- Iris + Casbin: 权限管理实践 (中文)
- 从零开始学习iris + Casbin

Argo CD

- 在Argo CD中使用Casbin进行组织RBAC

GShark

- GShark：轻松有效地在Github中扫描敏感信息（中文）

SpringBoot

- jCasbin：一个更轻量级的权限管理解决方案（中文）
- 将jCasbin与JFinal集成（中文）

Express

- 如何在AWS上为您的无服务器HTTP API添加基于角色的访问控制

Koa

- 使用Casbin和Koa进行授权第1部分
- 使用Casbin和Koa进行授权第2部分

Nest

- 如何使用 Casbin 和 Nest.js 创建基于角色的授权中间件
- nest.js：Casbin RESTful RBAC授权中间件（视频）

- 基于Casbin的Node.js中的属性访问控制的演示应用
- 使用cqrs graphql微服务架构的多租户SaaS启动套件

Fastify

- 在Node.js中使用Fastify和Casbin进行访问控制
- Casbin，为您的项目提供强大且高效的ACL
- 在dotnet中使用Casbin进行授权
- 在Rust中使用Casbin进行基于角色的HTTP授权
- [如何在你的rust web-app中使用casbin授权 第1部分]
- [如何在你的rust web-app中使用casbin授权 第2部分]

APISIX

- 在APISIX中使用Casbin进行授权

Understanding How Casbin Matching Works in Detail

在这篇文章中，我将解释如何使用Casbin库设计和实现RBAC。 对于处理多个资源层次结构和从高级别继承权限的SaaS平台，Casbin提供了一个性能良好的替代方案。

RBAC简介

RBAC是一种基于个人角色限制对资源访问的方法。 为了更好地理解分层RBAC是如何工作的，让我们在下一节看一下Azure的RBAC系统，然后尝试实现一个类似的系统。

理解Azure的分层RBAC

Azure的所有资源都有一个名为Owner的角色。 假设如果我在订阅级别被分配了Owner

角色，那就意味着我是该订阅下所有资源组和资源的Owner。 如果我在资源组级别拥有Owner，那么我就是该资源组下所有资源的Owner。

这张图片显示我在订阅级别拥有Owner访问权限。

当我检查这个订阅下的一个资源组的IAM时，你可以看到我从订阅中继承了Owner访问权限。

所以，这就是Azure的RBAC是如何分层的。 大多数企业软件都使用分层RBAC，因为资源级别的分层性质。 在这个教程中，我们将尝试使用Casbin实现一个类似的系统。

Casbin是如何工作的?

在深入实现之前，理解Casbin是什么以及它在高层次上如何运作是很重要的。 这种理解是必要的，因为每个基于角色的访问控制（RBAC）系统可能会根据特定的需求有所不同。 通过掌握Casbin的工作原理，我们可以有效地调整模型。

什么是ACL?

ACL代表访问控制列表。 这是一种将用户映射到操作和操作映射到资源的方法。

模型定义

让我们考虑一个简单的ACL模型示例。

```
[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. **request_definition**是系统的查询模板。 例如，一个请求`alice, write, data1`可以被解释为"主体Alice能否对对象'data1'执行'write'操作？"。

2. `policy_definition`是系统的分配模板。 例如，通过创建一个策略`alice, write, data1`，你就赋予了主体Alice在对象'data1'上执行'write'操作的权限。

3. `policy_effect`定义了策略的效果。

4. 在`matchers`部分，请求使用条件`r.sub == p.sub && r.obj == p.obj && r.act == p.act`与策略进行匹配。

现在让我们在Casbin编辑器上测试模型

打开编辑器并将上述模型粘贴到模型编辑器中。

在策略编辑器中粘贴以下内容：

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

在请求编辑器中粘贴以下内容：

```
alice, read, data1
```

结果将是:

```
true
```

ACL模型、策略和请求匹配的视觉表示

什么是RBAC?

RBAC代表基于角色的访问控制。 在RBAC中，用户被分配一个资源的角色，一个角色可以包含任意的操作。 然后请求检查用户是否有权限在资源上执行操作。

模型定义

让我们考虑一个简单的RBAC模型:

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

[policy_definition]
p = sub, act, obj

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

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

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

1. `role_definition`是一个图关系构建器，它使用图来比较请求对象和策略对象。

现在让我们在Casbin编辑器上测试模型

打开编辑器并将上述模型粘贴到模型编辑器中。

在策略编辑器中粘贴以下内容：

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

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

在请求编辑器中粘贴以下内容：

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

结果将是:

```
true
false
true
true
false
```

RBAC模型、策略和请求匹配的视觉表示

`g - Role to action mapping`表有一个图映射角色到操作。 这个图可以被编码为一系列的边，如在策略中所示，这是表示图的一种常见方式:

```
g, reader, read
```

> **信息**
>
> `p`表示一个可以使用`==`操作符进行比较的普通策略。 `g`是一个基于图的比较函数。 你可以通过添加数字后缀如`g, g2, g3, ...`等来定义多个图比较器。

什么是分层RBAC?

在分层RBAC中，有多种类型的资源，并且资源类型之间存在继承关系。 例如，“订阅”是一种类型，“资源组”是另一种类型。 类型为**订阅**的sub1可以包含多个类型为**资源组**的资源组（rg1，rg2）。

与资源层次结构类似，将有两种类型的角色和操作：订阅角色和操作，以及资源组角色和操作。 订阅角色和资源组角色之间存在任意关系。 例如，考虑一个订阅角色sub-owner。 这个角色被一个资源组角色rg-owner继承，这意味着如果我在订阅sub1上被分配了sub-owner角色，那么我自动也获得了rg1和rg2上的rg-owner角色。

模型定义

让我们以**分层**RBAC模型的一个简单例子来说明：

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

[policy_definition]
p = sub, act, obj

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

[policy_effect]
```

1. role_definition是一个图关系构建器，它使用图来比较请求对象和策略对象。

现在让我们在Casbin编辑器上测试这个模型

打开编辑器并将上述模型粘贴到模型编辑器中。

在策略编辑器中粘贴以下内容：

```
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
```

并在请求编辑器中粘贴以下内容：

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

结果将是：

```
true
```

RBAC模型、策略和请求匹配的视觉表示

g - 角色到（操作，角色）映射表有一个图映射角色到操作，角色映射。 这个图可以被编码为一系列的边，如策略中所示，这是表示图的常见方式：

```
// 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
```

g2 - 订阅到资源组映射表有一个图映射订阅到资源组：

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

主题匹配视觉表示

操作匹配视觉表示

对象匹配视觉表示

> **信息**
>
> 当一个请求提交给Casbin时，所有的策略都会进行这种匹配。 如果至少有一个策略匹配，那么请求的结果为真。 如果没有策略匹配请求，那么结果为假。

结论

在本教程中，我们学习了不同的授权模型是如何工作的，以及如何使用Casbin对其进行建模。 在本教程的第二部分，我们将在一个演示的Spring Boot应用程序中实现这一点，并使用Casbin保护API。

访问控制模型

Supported Models

Casbin支持的模型

Syntax for Models

模型的语法

Effector

Casbin中的Effector接口

Functions

使用内置函数或指定自定义函数

RBAC

Casbin RBAC 使用

RBAC with Pattern

带模式的RBAC

RBAC with Domains

RBAC与域的使用

RBAC with Conditions

使用带有条件的RBAC

Casbin RBAC vs. RBAC96

Casbin RBAC和RBAC96之间的区别

ReBAC

ReBAC based on Casbin

ABAC

基于Casbin的ABAC

BLP

Bell-LaPadula model in Casbin

Biba

Biba Integrity Model in Casbin

LBAC

Lattice-Based Access Control model in Casbin

📄 Priority Model

Casbin的优先级模型用于管理具有不同优先级的策略

📄 Usage Control (UCON)

Usage Control Model in Casbin

📄 Super Admin

超级管理员是整个系统的管理员。 它可以用于RBAC、ABAC和带有域的RBAC等模型中。

Supported Models

1. ACL (访问控制列表)
2. **带有超级用户的ACL**
3. **无用户的ACL**：这对于没有身份验证或用户登录的系统特别有用。
4. **无资源的ACL**：在某些情况下，目标是一种资源类型，而不是单个资源。 可以使用像"write-article"和"read-log"这样的权限。 这并不控制对特定文章或日志的访问。
5. RBAC (基于角色的访问控制)
6. **带有资源角色的RBAC**：用户和资源同时可以拥有角色（或组）。
7. **带有域/租户的RBAC**：用户可以为不同的域/租户拥有不同的角色集。
8. ABAC (基于属性的访问控制)：可以使用类似"resource.Owner"的语法糖来获取资源的属性。
9. BLP (Bell-LaPadula): A formal state transition model of computer security policy that describes a set of access control rules which use security labels on objects and clearances for subjects.
10. Biba (Biba Integrity Model): A computer security model that restricts information flow in a system to prevent unauthorized disclosure of classified information.
11. LBAC (Lattice-Based Access Control): A formal access control model that combines confidentiality and integrity controls in a unified framework, implementing a lattice structure for granular access control decisions.
12. UCON (Usage Control): A next-generation access control model that emphasizes continuous authorization, attribute mutability, and a unified framework of authorizations, obligations, and conditions.
13. RESTful：支持像"/res/*"，"/res/:id"这样的路径，以及像"GET"，"POST"，"PUT"，"DELETE"这样的HTTP方法。
14. IP Match: Supports IP address matching for network-based access control.
15. **拒绝优先**：同时支持允许和拒绝授权，其中拒绝优先于允许。
16. **优先级**：策略规则可以设置优先级，类似于防火墙规则。

示例

模型	模型文件	策略文件
ACL	basic_model.conf	basic_policy.csv
带有超级用户的ACL	basic_with_root_model.conf	basic_policy.csv
无用户的ACL	basic_without_users_model.conf	basic_without_users_policy.csv
无资源的ACL	basic_without_resources_model.conf	basic_without_resources_policy.csv
RBAC	rbac_model.conf	rbac_policy.csv
带资源角色的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
ReBAC	rebac_model.conf	rebac_policy.csv
ABAC	abac_model.conf	N/A
BLP	blp_model.conf	N/A
Biba	biba_model.conf	N/A

模型	模型文件	策略文件
LBAC	lbac_model.conf	N/A
IP Match	ipmatch_model.conf	ipmatch_policy.csv
RESTful	keymatch_model.conf	keymatch_policy.csv
拒绝覆盖	rbac_with_not_deny_model.conf	rbac_with_deny_policy.csv
允许和拒绝	rbac_with_deny_model.conf	rbac_with_deny_policy.csv
优先级	priority_model.conf	priority_policy.csv
明确的优先级	priority_model_explicit	priority_policy_explicit.csv
主题优先级	subject_priority_model.conf	subject_priority_policyl.csv

Syntax for Models

- 模型配置（CONF）应至少有四个部分：`[request_definition]`，`[policy_definition]`，`[policy_effect]`和`[matchers]`。
- 如果模型使用基于角色的访问控制（RBAC），它还应包括`[role_definition]`部分。
- 模型配置（CONF）可以包含注释。 注释以`#`符号开始，`#`符号后的所有内容都将被注释掉。

请求定义

`[request_definition]`部分定义了`e.Enforce(...)`函数中的参数。

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

在这个例子中，`sub`，`obj`和`act`代表了经典的访问三元组：主体（访问实体），对象（被访问资源）和动作（访问方法）。 然而，你可以自定义你自己的请求格式。 例如，如果你不需要指定特定的资源，你可以使用`sub, act`，或者如果你有两个访问实体，你可以使用`sub, sub2, obj, act`。

策略定义

`[policy_definition]`是策略的定义。 它定义了策略的含义。 例如，我们有以下模型：

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

我们有以下策略（如果在策略文件中）：

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

策略中的每一行都被称为策略规则。 每个策略规则都以`策略类型`开始，如`p`或`p2`。 如果有多个定义，它用于匹配策略定义。 上述策略显示了以下绑定。 绑定可以在匹配器中使用。

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

> **提示**
>
> 策略规则中的元素始终被视为`字符串`。 如果你对此有任何疑问，请参考以下讨论：https://github.com/casbin/casbin/issues/113

策略效果

`[policy_effect]`是策略效果的定义。 如果多个策略规则匹配请求，它决定是否应批准访问请求。 例如，一条规则允许，另一条规则拒绝。

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

上述策略效果意味着，如果有任何匹配的`allow`策略规则，最终效果是`allow`（也称为允许覆盖）。 `p.eft`是策略的效果，它可以是`allow`或`deny`。 这是可选的，其默认值是`allow`。 由于我们在上面没有指定它，所以它使用默认值。

策略效果的另一个例子是：

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

这意味着，如果没有匹配的`deny`策略规则，最终的效果是`allow`（也称为deny-override）。 `some`意味着存在一个匹配的策略规则。 `any`意味着所有匹配的策略规则（这里未使用）。 策略效果甚至可以与逻辑表达式连接：

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

这意味着必须至少有一个匹配的`allow`策略规则，并且不能有任何匹配的`deny`策略规则。 因此，以这种方式，支持允许和拒绝授权，并且拒绝优先。

> **备注**
>
> 尽管我们设计了如上的策略效果语法，但当前的实现只使用硬编码的策略效果。 这是因为我们发现没有那么大的需要那种级别的灵活性。 所以现在，你必须使用其中一个内置的策略效果，而不是定制你自己的。

支持的内置策略效果有：

策略效果	含义	例子
some(where (p.eft == allow))	allow-	ACL, RBAC,

策略效果	含义	例子
	override	etc.
!some(where (p.eft == deny))	deny-override	Deny-override
some(where (p.eft == allow)) && !some(where (p.eft == deny))	allow-and-deny	Allow-and-deny
priority(p.eft) || deny	priority	Priority
subjectPriority(p.eft)	基于角色的优先级	主题-优先级

匹配器

[匹配器]是策略匹配器的定义。 匹配器是定义如何根据请求评估策略规则的表达式。

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

上述匹配器是最简单的，意味着请求中的主题、对象和动作应与策略规则中的相匹配。

匹配器中可以使用算术运算符如`+, -, *, /`和逻辑运算符如`&&, ||, !`。

匹配器中表达式的顺序

表达式的顺序可以极大地影响性能。 看看下面的例子以获取更多详细信息：

```go
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
`

func TestManyRoles(t *testing.T) {

    m, _ := model.NewModelFromString(rbac_models)
    e, _ := NewEnforcer(m, false)

    roles := []string{"admin", "manager", "developer", "tester"}

    // 2500 projects
    for nbPrj := 1; nbPrj < 2500; nbPrj++ {
        // 4 objects and 1 role per object (so 4 roles)
        for _, role := range roles {
            roleDB := fmt.Sprintf("%s_project:%d", role, nbPrj)
            objectDB := fmt.Sprintf("/projects/%d", nbPrj)
            e.AddPolicy(roleDB, objectDB, "GET")
        }
        jasmineRole := fmt.Sprintf("%s_project:%d", roles[1],
nbPrj)
        e.AddGroupingPolicy("jasmine", jasmineRole)
    }
```

执行时间可能会很长，长达6秒。

```
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
```

然而，如果我们调整匹配器中表达式的顺序，并将更耗时的表达式如函数放在后面，执行时间将会非常短。

将上述示例中匹配器中的表达式顺序更改为:

```
[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
    rbac_api_test.go:599: RESPONSE abu
```

多个部分类型

如果你需要多个策略定义或多个匹配器，你可以使用`p2`或`m2`作为例子。 实际上，上述的四个部分都可以使用多种类型，语法是`r`后面跟一个数字，如`r2`或`e2`。 默认情况下，这四个部分应一一对应。 例如，你的`r2`部分只会使用`m2`匹配器来匹配`p2`策略。

你可以将`EnforceContext`作为`enforce`方法的第一个参数来指定类型。`EnforceContext`的定义如下:

Go　Node.js　Java

```go
EnforceContext{"r2","p2","e2","m2"}
type EnforceContext struct {
    RType string
    PType string
    EType string
    MType string
}
```

```node.js
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;
  }
}
```

```java
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;
    }
}
```

这是一个使用示例。 请参考model和policy。 请求如下:

Go Node.js Java

```go
// 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
```

```
// 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.seteType("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
```

特殊语法

你也可以使用“in”运算符，这是唯一一个带有文本名称的运算符。 此运算符检查右侧的数组，看是否包含等于左侧值的值。 等式是通过使用==运算符确定的，这个库不检查值之间的类型。 只要两个值可以被转换为接口{}并且仍然可以用==检查等式，它们就会按预期工作。 注意，你可以为数组使用一个参数，但它必须是一个[]接口{}。

也可以参考rbac_model_matcher_using_in_op，keyget2_model和keyget_model。

示例：

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

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

表达式求值器

Casbin中的匹配器评估是由每种语言的表达式求值器实现的。 Casbin整合了他们的力量，提供了统一的PERM语言。 除了这里提供的模型语法外，这些表达式求值器可能提供额外的功能，这些功能可能不被其他语言或实现支持。 在使用这个功能时，请谨慎。

每个Casbin实现使用的表达式求值器如下：

实现	语言	表达式求值器
Casbin	Golang	https://github.com/Knetic/govaluate
jCasbin	Java	https://github.com/killme2008/aviator
Node-Casbin	Node.js	https://github.com/donmccurdy/expression-eval
PHP-Casbin	PHP	https://github.com/symfony/expression-language
PyCasbin	Python	https://github.com/danthedeckie/simpleeval
Casbin.NET	C#	https://github.com/davideicardi/DynamicExpresso
Casbin4D	Delphi	https://github.com/casbin4d/Casbin4D/tree/master/SourceCode/Common/Third%20Party/

实现	语言	表达式求值器
		TExpressionParser
casbin-rs	Rust	https://github.com/jonathandturner/rhai
casbin-cpp	C++	https://github.com/ArashPartow/exprtk

> **备注**
>
> 如果您遇到Casbin的性能问题，可能是由于表达式评估器的低效率引起的。 您可以直接向Casbin或表达式评估器提出问题，寻求提高性能的建议。 有关更多详细信息，请参阅Benchmarks部分。

Effector

'Effect'代表政策规则的结果，而'Effector'是用于处理Casbin中效果的接口。

`MergeEffects()`

'MergeEffects()'函数用于将执行器收集的所有匹配结果合并为一个决定。

例如：

Go

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

在这个例子中：

- 'Effect'是由这个函数合并的最终决定（初始化为'Indeterminate'）。
- 'explainIndex'是'eft'（'Allow'或'Deny'）的索引，它被初始化为'-1'。
- 'err'用于检查是否支持效果。
- 'expr'是政策效果的字符串表示。
- `effects`是一个效果数组，可以是`Allow`、`Indeterminate`或`Deny`。
- `matches`是一个数组，用于指示结果是否与策略匹配。
- `policyIndex`是策略在模型中的索引。
- `policyLength`是策略的长度。

上面的代码说明了如何将参数传递给`MergeEffects()`函数，该函数将根据`expr`处理效果和匹配。

要使用`Effector`，请按照以下步骤操作：

Go

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

`MergeEffects()`的基本思想是，如果`expr`可以匹配结果，表明`p_eft`是`allow`，那么所有效果都可以合并。 如果没有匹配到拒绝规则，那么决定是允许。

> **备注**
>
> 如果`expr`不匹配条件`"priority(p_eft) || deny"`，并且`policyIndex`小于`policyLength-1`，它将**短路**中间的一些效果。

Functions

匹配器中的函数

你甚至可以在匹配器中指定函数，使其更强大。 你可以使用内置函数或指定你自己的函数。 内置的键匹配函数采用以下格式：

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

它们返回一个布尔值，表示`url`是否匹配`pattern`。

支持的内置函数有：

函数	url	模式	例子
keyMatch	像`/alice_data/resource1`这样的URL路径	像`/alice_data/*`这样的URL路径或`*`模式	keymatch_model.conf/keymatch_policy.csv
keyMatch2	像`/alice_data/resource1`这样的URL路径	像`/alice_data/:resource`这样的URL路径或`:`模式	keymatch2_model.conf/keymatch2_policy.csv
keyMatch3	像`/alice_data/resource1`这样的URL路径	像`/alice_data/{resource}`这样的URL路径或`{}`模式	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L171-L196
keyMatch4	像`/alice_data/123/book/123`这样的URL路径	像`/alice_data/{id}/book/{id}`这样的URL路径或`{}`模式	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L208-L222
keyMatch5	像`/alice_data/123/?status=1`这样的URL路径	像`/alice_data/{id}/*`这样的URL路径，`{}`或`*`模式	https://github.com/casbin/casbin/blob/1cde2646d10ad1190c0d784c3a1c0e1ace1b5bc9/util/builtin_operators_test.go#L485-L526
regexMatch	任何字符串	一个正则表达式模式	keymatch_model.conf/keymatch_policy.csv
ipMatch	像`192.168.2.123`这样的IP地址	像`192.168.2.0/24`这样的IP地址或CIDR	ipmatch_model.conf/ipmatch_policy.csv
globMatch	像`/alice_data/resource1`这样的路径样式路径	像`/alice_data/*`这样的glob模式	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L426-L466

对于获取键的函数，它们通常接受三个参数（除了`keyGet`）：

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

如果键`key_name`匹配模式，它们将返回键的值，如果没有匹配项，将返回`""`。

例如，`KeyGet2("/resource1/action", "/:res/action", "res")`将返回`"resource1"`，`KeyGet3("/resource1_admin/action", "/{res}_admin/*", "res")`将返回`"resource1"`。 至于`KeyGet`，它接受两个参数，`KeyGet("/resource1/action", "/*)`将返回`"resource1/action"`。

函数	url	模式	key_name	例子
keyGet	像`/proj/resource1`这样的URL路径	像`/proj/*`这样的URL路径或`*`模式	\	keyget_model.conf/keymatch_policy.csv
keyGet2	像`/proj/resource1`这样的URL路径	像`/prooj/:resource`这样的URL路径或`:`模式	在模式中指定的键名	keyget2_model.conf/keymatch2_policy.csv
keyGet3	像`/proj/res3_admin/`这样的URL路径	像`/proj/{resource}_admin/*`这样的URL路径或`{}`模式	在模式中指定的键名	https://github.com/casbin/casbin/blob/7bd496f94f5a2739a392d333a9aaaa10ae397673/util/builtin_operators_test.go#L209-L247

查看上述函数的详细信息：https://github.com/casbin/casbin/blob/master/util/builtin_operators_test.go

如何添加自定义函数

首先，准备你的函数。 它接受几个参数并返回一个布尔值：

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

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

然后，用`interface{}`类型包装它：

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

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

最后，将函数注册到Casbin执行器：

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

现在，你可以在你的模型CONF中像这样使用函数：

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

RBAC

角色定义

`[role_definition]` 用于定义 RBAC 角色继承关系。 Casbin 支持多个实例的 RBAC 系统，其中用户可以拥有角色及其继承关系，资源也可以拥有角色及其继承关系。 这两个 RBAC 系统不会相互干扰。

此部分为可选。 如果你在模型中不使用 RBAC 角色，那么省略此部分。

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

上述角色定义显示 `g` 是一个 RBAC 系统，而 `g2` 是另一个 RBAC 系统。 ,表示在继承关系中涉及两个方。 在最常见的情况下，如果你只需要用户的角色，通常只使用g。 当你需要用户和资源的角色（或组）时，也可以同时使用 g和g2`。 请参阅 rbac_model.conf 和 rbac_model_with_resource_roles.conf 以获取示例。

Casbin 在策略中存储实际的用户-角色映射（如果你在资源上使用角色，则为资源-角色映射）。 例如：

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

这意味着 `alice` 继承/是角色 `data2_admin` 的成员。 在这里， `alice` 可以是用户，资源，或者角色。 Casbin 只将其识别为字符串。

然后，在匹配器中，你应该如下所示检查角色：

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

这意味着请求中的 `sub` 应该在策略中拥有角色 `sub`。

> **备注**
>
> 1. Casbin 只存储用户-角色映射。
> 2. Casbin 不验证用户是否是有效用户或角色是否是有效角色。 这应由身份验证来处理。
> 3. 不要在 RBAC 系统内为用户和角色使用相同的名称，因为 Casbin 将用户和角色识别为字符串，Casbin 无法知道你是指定用户 `alice` 还是角色 `alice`。 你可以简单地通过使用 `role_alice` 来解决这个问题。
> 4. 如果 `A` 有角色 `B`，并且 `B` 有角色 `C`，那么 `A` 有角色 `C`。 这种传递性目前是无限的。

> **令牌名称约定**
>
> 按照惯例，策略定义中的主题令牌名称为 `sub` 并放在开头。 现在，Golang Casbin 支持自定义令牌名称和位置。 如果主题令牌名称是 `sub`，则主题令牌可以放在任意位置，无需任何额外操作。 如果主题令牌名称不是 `sub`，无论其位置如何，都应在初始化执行器后调用 `e.SetFieldIndex()` 为 `constant.SubjectIndex`。
>
> ```
> # `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
```

角色层次结构

Casbin 的 RBAC 支持 RBAC1 的角色层次结构特性，这意味着如果 `alice` 有 `role1`，并且 `role1` 有 `role2`，那么 `alice` 也将拥有 `role2` 并继承其权限。

在这里，我们有一个叫做层次级别的概念。 所以，在这个例子中，层次级别是 2。 对于 Casbin 中的内置角色管理器，你可以指定最大层次级别。 默认值是 10。 这意味着像 `alice` 这样的最终用户只能继承 10 个级别的角色。

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

    return &rm
}
```

如何区分角色和用户?

Casbin 在其 RBAC 中并不区分角色和用户。 它们都被视为字符串。 如果你只使用单级

RBAC（其中角色永远不会是另一个角色的成员），你可以使用 `e.GetAllSubjects()` 获取所有用户和 `e.GetAllRoles()` 获取所有角色。 它们将分别列出所有 `g, u, r` 规则中的所有 `u` 和所有 `r`。

但是，如果你正在使用多级 RBAC（具有角色层次结构）并且你的应用程序不记录名称（字符串）是用户还是角色，或者你有一个用户和一个角色具有相同的名称，你可以在将其传递给 Casbin 之前为角色添加前缀，如 `role::admin`。 这样，你可以通过检查这个前缀来知道它是否是一个角色。

如何查询隐式角色或权限?

当用户通过 RBAC 层次结构继承角色或权限，而不是在策略规则中直接分配它们时，我们称这种类型的分配为“隐式”。 要查询此类隐式关系，你需要使用这两个 API：`GetImplicitRolesForUser()` 和 `GetImplicitPermissionsForUser()`，而不是 `GetRolesForUser()` 和 `GetPermissionsForUser()`。 有关更多详细信息，请参阅此 GitHub 问题。

在 RBAC 中使用模式匹配

参见 RBAC with Pattern

角色管理器

详见 Role Managers 部分。

RBAC with Pattern

快速开始

- 在 `g(_, _)` 中使用模式。

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

- 使用带有域的模式。

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

- 使用所有模式。

  只需结合使用两个API。

如上所示，创建 `enforcer` 实例后，您需要通过 `AddNamedMatchingFunc` 和 `AddNamedDomainMatchingFunc` API激活模式匹配，这些API决定了模式如何匹配。

> **备注**
>
> If you use the online editor, You can add a pattern matching function by clicking the "Add Role Matching" button in the lower left corner.

在RBAC中使用模式匹配

有时，您希望具有特定模式的某些主题、对象或域/租户自动被授予角色。 RBAC中的模式匹配函数可以帮助您做到这一点。 模式匹配函数与之前的 匹配函数 具有相同的参数和返回值。

模式匹配函数支持 `g` 的每个参数。

我们知道，通常RBAC在匹配器中表示为 `g(r.sub, p.sub)`。 然后我们可以使用像这样的策略：

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

所以 `alice` 可以阅读所有书籍，包括 `book 1` 和 `book 2`。 但可能有数千本书，将每本书都添加到书籍角色（或组）中是非常繁琐的，每个 `g` 策略规则都需要添加一本书。

但是，使用模式匹配函数，您只需要用一行代码就可以编写策略：

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

Casbin 将自动为您将 `/book/1` 和 `/book/2` 匹配到模式 `/book/:id` 中。 您只需要像这样向执行器注册函数：

Go　Node.js

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

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

在域/租户中使用模式匹配函数时，您需要向执行器和模型注册函数。

Go　Node.js

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

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

如果你不明白 `g(r.sub, p.sub, r.dom)` 是什么意思，请阅读 带域的rbac。 简单来说，`g(r.sub, p.sub, r.dom)` 将检查用户 `r.sub` 是否在域 `r.dom` 中具有角色 `p.sub`。 所以这就是匹配器的工作方式。 你可以在这里看到完整的示例。

除了上面的模式匹配语法，我们还可以使用纯域模式。

例如，如果我们希望 `sub` 在不同的域，`domain1` 和 `domain2` 中都有访问权限，我们可以使用纯域模式：

```
p, admin, domain1, data1, read
p, admin, domain1, data1, write
p, admin, domain2, data2, read
p, admin, domain2, data2, write

g, alice, admin, *
g, bob, admin, domain2
```

在这个例子中，我们希望`alice`能够在domain1和domain2中读取和写入`data`。 `g`中的模式匹配`*`使`alice`能够访问两个域。

通过使用模式匹配，特别是在更复杂的场景中，需要考虑很多域或对象，我们可以以更优雅和有效的方式实现`policy_definition`。

RBAC with Domains

与域租户的角色定义

Casbin中的RBAC角色可以是全局的或特定于域的。 特定于域的角色意味着当用户在不同的域/租户中时，用户的角色可以不同。 对于像云这样的大型系统，这非常有用，因为用户通常在不同的租户中。

与域/租户的角色定义应该如下所示：

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

第三个`_`代表域/租户的名称，这部分不应该改变。 然后策略可以是：

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

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

这意味着`tenant1`中的`admin`角色可以读取`data1`。 并且`alice`在`tenant1`中拥有`admin`角色，在`tenant2`中拥有`user`角色。 因此，她可以读取`data1`。 然而，由于`alice`在`tenant2`中不是`admin`，她无法读取`data2`。

然后，在匹配器中，你应该按照以下方式检查角色：

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

请参考rbac_with_domains_model.conf以获取示例。

> **令牌名称约定**
>
> 注意：按照惯例，策略定义中的域令牌名称是`dom`，并作为第二个令牌（`sub, dom, obj, act`）放置。 现在，Golang Casbin支持自定义令牌名称和放置。 如果域令牌名称是`dom`，则无需任何额外操作，域令牌可以放置在任意位置。 如果域令牌名称不是`dom`，则无论其位置如何，都应在初始化执行器后调用`e.SetFieldIndex()`以获取`constant.DomainIndex`。
>
> ```
> # `domain` here for `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

条件角色管理器

`ConditionalRoleManager`支持在策略级别自定义条件函数。

例如，当我们需要一个临时角色策略时，我们可以按照以下方法进行：

`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 = _, _, (_, _)` 使用 `(_, _)` 来包含传递给条件函数的参数列表，`_` 作为参数占位符

`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, _
```

基本使用

通过 `AddNamedLinkConditionFunc` 为角色策略(`g`类型策略) 添加一个条件函数，当执行强制时，将自动获取相应的参数并传递给条件函数进行检查。 如果检查通过，那么相应的角色策略(`g`类型策略)就是有效的，否则就是无效的

```
e.AddNamedLinkConditionFunc("g", "alice", "data2_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data3_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data4_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data5_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data6_admin",
util.TimeMatchFunc)
e.AddNamedLinkConditionFunc("g", "alice", "data7_admin",
```

自定义条件函数

自定义条件函数需要符合以下函数类型

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

例如:

```
// 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",
```

带有域的条件角色管理器

`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
```

基本使用

通过 `AddNamedDomainLinkConditionFunc` 为角色策略(`g`类型策略) 添加一个条件函数，当执行强制时，将自动获取相应的参数并传递给条件函数进行检查。 如果检查通过，那么相应的角色策略(`g`类型策略)就是有效的，否则就是无效的

```
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
e.enforce("alice", "domain6", "data6", "write")             //
except: false
```

自定义条件函数

像基本的`Conditional RoleManager`一样，支持自定义函数，使用上没有区别。

注意，`DomainMatchingFunc`，`MatchingFunc`和`LinkConditionFunc`处于不同的级别，并在不同的情况下使用。

Casbin RBAC vs. RBAC96

Casbin RBAC和RBAC96

在这份文档中，我们将比较Casbin RBAC和RBAC96。

Casbin RBAC支持RBAC96的几乎所有功能，并在此基础上增加了新的功能。

RBAC版本	支持级别	描述
RBAC0	完全支持	RBAC0是RBAC96的基础版本。 它明确了用户、角色和权限之间的关系。
RBAC1	完全支持	RBAC1在RBAC0的基础上增加了角色层次结构。 这意味着，如果`alice`有`role1`， `role1`有`role2`，那么`alice`也将拥有`role2`并继承其权限。
RBAC2	支持互斥处理（像这样）	RBAC2在RBAC0上增加了约束。 这使得RBAC2能够处理互斥的策略。 然而，不支持数量限制。
RBAC3	支持互斥处理（像这样）	RBAC3是RBAC1和RBAC2的结合。 它支持RBAC1和RBAC2中的角色层次结构和约束。 然而，不支持数量限制。

Casbin RBAC和RBAC96之间的区别

1. 在Casbin中，用户和角色之间的区别并不像在RBAC96中那么明显。

   在Casbin中，用户和角色都被视为字符串。 例如，考虑以下策略文件：

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

   如果你使用Casbin Enforcer的实例调用`GetAllSubjects()`方法：

   ```
   e.GetAllSubjects()
   ```

   返回值将是：

   ```
   [admin alice]
   ```

   这是因为在Casbin中，主题包括用户和角色。

   然而，如果你调用`GetAllRoles()`方法：

   ```
   e.GetAllRoles()
   ```

   返回值将是：

   ```
   [admin]
   ```

从这里，你可以看出在Casbin中，用户和角色是有区别的，但它并不像在RBAC96中那么明显。 当然，你可以在你的策略中添加前缀，如`user::alice`和`role::admin`来明确它们的关系。

2. Casbin RBAC提供的权限比RBAC96更多。

   RBAC96只定义了7种权限：读、写、追加、执行、信用、借记和查询。

   然而，在Casbin中，我们将权限视为字符串。 这使你能够创建更适合你需求的权限。

3. Casbin RBAC支持域。

   在Casbin中，你可以基于域进行授权。 这个功能使你的访问控制模型更加灵活。

ReBAC

What is the ReBAC Model?

ReBAC (Relationship-Based Access Control) is a modern access control model that focuses on relationships between entities for permission management. Compared to traditional RBAC (Role-Based Access Control) or ABAC (Attribute-Based Access Control), ReBAC is better suited for systems with complex relationship networks, such as social networks, collaboration platforms, and multi-tenant systems.

In ReBAC, authorization decisions are based on relationships between entities, such as:

- Is the user the "owner" of the resource?
- Is the user a "friend" of the resource's "creator"?
- Does the user belong to an organization associated with the resource?
- Is the user an "admin" of a certain "project"?

These relationships are typically modeled as graph structures or paths.

ReBAC Support in Casbin

Casbin provides the following mechanisms to implement ReBAC:

- User-Resource-Role Relationships
- Resource-Type Relationships

A single policy rule can cover multiple users and multiple resources of the same type, enabling flexible and scalable permission control through relationship combinations.

Casbin uses `.conf` files to define access control models. Below is an official ReBAC model example:

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

[policy_definition]
p = role, obj_type, act

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

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

[matchers]
m = g(r.sub, r.obj, p.role) && g2(r.obj, p.obj_type) && r.act
== p.act
```

```
# Permission definition: The "collaborator" role can read files
of type "doc"
p, collaborator, doc, read

# User-Resource-Role Relationship: alice is a collaborator of
doc1
g, alice, doc1, collaborator

# Resource-Type Relationship: doc1 is of type "doc"
g2, doc1, doc
```

By checking whether a user has a specific role for a given resource and whether the resource belongs to a specified type, permissions are automatically derived through **role relationships + type relationships + permission definitions**.

ABAC

什么是ABAC模型?

ABAC代表基于属性的访问控制。 它允许你通过使用主体、对象或行为的属性（属性）来控制访问，而不是使用字符串值本身。 你可能听说过一种复杂的ABAC访问控制语言，叫做XACML。 另一方面，Casbin的ABAC要简单得多。 在Casbin的ABAC中，你可以使用结构体或类实例代替模型元素的字符串。

让我们来看一下官方的ABAC示例:

```
[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
```

在匹配器中，我们使用`r.obj.Owner`代替`r.obj`。 传递给`Enforce()`函数的`r.obj`将是一个结构体或类实例，而不是一个字符串。 Casbin将使用反射来为你检索该结构体或类中的`obj`成员变量。

这是`r.obj`结构体或类的定义:

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

如果您想通过JSON向enforcer传递参数，则需要使用`e.EnableAcceptJsonRequest(true)`启用该功能。

例如：

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

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

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

> **备注**
>
> 启用接受JSON参数的功能可能会导致性能下降1.1到1.5倍。

如何使用ABAC?

要使用ABAC，你需要做两件事：

1. 在模型匹配器中指定属性。
2. 将结构体或类实例作为参数传递给Casbin的`Enforce()`函数。

> **危险**
>
> 目前，只有像`r.sub`、`r.obj`、`r.act`等请求元素支持ABAC。 你不能在像

`p.sub`这样的策略元素上使用它，因为在Casbin的策略中没有办法定义一个结构体或类。

提示

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

提示

如果你需要在与CSV分隔符冲突的策略中使用逗号，你可以通过用引号包围语句来转义它。 例如，`"keyMatch("bob", r.sub.Role)"`将不会被分割。

扩展模型以适应复杂和大量的ABAC规则

上述ABAC模型的实现在其核心上是简单的。 然而，在许多情况下，授权系统需要复杂和大量的ABAC规则。 为了满足这个需求，建议在策略中添加规则，而不是在模型中。 这可以通过引入`eval()`功能构造来实现。 这是一个例子：

这是用于定义ABAC模型的`CONF`文件的定义。

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

[policy_definition]
p = sub_rule, obj, act

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

在这个例子中，`p.sub_rule`是一个结构体或类（用户定义的类型），包含了在策略中使用的必要属性。

这是用于`Enforcement`的模型对应的策略。 现在，你可以使用传递给`eval()`的对象实例作为参数来定义某些ABAC约束。

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

BLP

Overview

The Bell-LaPadula (BLP) model is a formal state transition model of computer security policy that describes a set of access control rules which use security labels on objects and clearances for subjects. It was developed by David Elliott Bell and Leonard J. LaPadula in 1973.

Model

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

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

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

[matchers]
m = (r.act == "read" && r.sub_level >= r.obj_level) || (r.act
== "write" && r.sub_level <= r.obj_level)
```

Policy

BLP model typically doesn't require explicit policy rules as the access control is determined by the security levels of subjects and objects. The matcher function implements the BLP rules:

- **No Read Up**: A subject cannot read an object with a higher security level
- **No Write Down**: A subject cannot write to an object with a lower security level

Examples

Request Examples

```
alice, 3, data1, 1, read    # alice (level 3) reads data1
(level 1) - ALLOWED
bob, 2, data2, 2, read      # bob (level 2) reads data2 (level
2) - ALLOWED
charlie, 1, data1, 1, read  # charlie (level 1) reads data1
(level 1) - ALLOWED
bob, 2, data3, 3, read      # bob (level 2) reads data3 (level
3) - DENIED (No Read Up)
charlie, 1, data2, 2, read  # charlie (level 1) reads data2
(level 2) - DENIED (No Read Up)

alice, 3, data3, 3, write   # alice (level 3) writes data3
(level 3) - ALLOWED
bob, 2, data3, 3, write     # bob (level 2) writes data3 (level
3) - ALLOWED
charlie, 1, data2, 2, write # charlie (level 1) writes data2
(level 2) - ALLOWED
```

Security Levels

In the BLP model, security levels are typically represented as integers where higher numbers indicate higher security levels:

- Level 1: Public/Unclassified
- Level 2: Confidential
- Level 3: Secret
- Level 4: Top Secret

Use Cases

BLP model is commonly used in:

- Military and government systems
- Financial institutions
- Healthcare systems
- Any environment requiring strict information flow control

Implementation Notes

- The model enforces mandatory access control (MAC)
- Security levels are assigned by system administrators
- Access decisions are based purely on security levels, not user identity
- The model prevents information leakage through read/write operations

Biba

Overview

The Biba Model (also known as Biba Integrity Model) is a formal state transition model of computer security policy developed by Kenneth J. Biba in 1975. It describes a set of access control rules designed to ensure data integrity. Unlike the Bell-LaPadula model which focuses on confidentiality, the Biba model is specifically designed to protect data integrity and prevent unauthorized modification of data.

Model

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

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

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

[matchers]
m = (r.act == "read" && r.sub_level <= r.obj_level) || (r.act
== "write" && r.sub_level >= r.obj_level)
```

Policy

The Biba model typically doesn't require explicit policy rules as the access control is determined by the integrity levels of subjects and objects. The matcher function implements the Biba integrity rules:

- **No Read Down (Simple Integrity Property)**: A subject cannot read an object with a lower integrity level
- **No Write Up (Star Integrity Property)**: A subject cannot write to an object with a higher integrity level

Core Principles

The Biba model is characterized by the phrase "read up, write down", which is the inverse of the Bell-LaPadula model's "read down, write up". This approach ensures:

1. **Data Integrity Protection**: Prevents corruption of high-integrity data by lower-integrity sources
2. **Controlled Information Flow**: Ensures that information flows only from higher to lower integrity levels for writes
3. **Trust Preservation**: Maintains the trustworthiness of data at each integrity level

Examples

Request Examples

```
alice, 3, data1, 1, read    # alice (level 3) reads data1
(level 1) - DENIED (No Read Down)
bob, 2, data2, 2, read      # bob (level 2) reads data2 (level
2) - ALLOWED
charlie, 1, data1, 1, read  # charlie (level 1) reads data1
(level 1) - ALLOWED
bob, 2, data3, 3, read      # bob (level 2) reads data3 (level
3) - ALLOWED
charlie, 1, data2, 2, read  # charlie (level 1) reads data2
(level 2) - ALLOWED

alice, 3, data3, 3, write   # alice (level 3) writes data3
(level 3) - ALLOWED
bob, 2, data3, 3, write     # bob (level 2) writes data3 (level
3) - DENIED (No Write Up)
charlie, 1, data2, 2, write # charlie (level 1) writes data2
(level 2) - DENIED (No Write Up)
alice, 3, data1, 1, write   # alice (level 3) writes data1
(level 1) - ALLOWED
bob, 2, data1, 1, write     # bob (level 2) writes data1 (level
1) - ALLOWED
```

Integrity Levels

In the Biba model, integrity levels are typically represented as integers where higher numbers indicate higher integrity levels:

- Level 1: Low integrity (e.g., public data, user-generated content)
- Level 2: Medium integrity (e.g., verified data, trusted sources)
- Level 3: High integrity (e.g., system data, administrative content)
- Level 4: Critical integrity (e.g., security policies, system configuration)

Use Cases

The Biba model is commonly used in:

- Financial systems where data accuracy is paramount
- Healthcare records management
- Database systems requiring data integrity
- Any environment where preventing data corruption is more important than preventing data disclosure
- Systems where the accuracy and reliability of information is critical

Implementation Notes

- The model enforces mandatory access control (MAC) focused on integrity
- Integrity levels are assigned by system administrators based on data trustworthiness
- Access decisions are based on integrity levels rather than user identity
- The model prevents data corruption through controlled read/write operations
- Unlike Bell-LaPadula, most Biba applications use only a small number of integrity levels

Comparison with Bell-LaPadula

Aspect	Bell-LaPadula	Biba
Primary Focus	Confidentiality	Integrity
Read Rule	No Read Up	No Read Down
Write Rule	No Write Down	No Write Up
Phrase	"Read down, write up"	"Read up, write down"

LBAC

What is the LBAC model?

LBAC stands for Lattice-Based Access Control. It is a formal access control model that can be implemented in various ways. The example shown below demonstrates one possible implementation in Casbin that combines confidentiality and integrity controls.

Model Definition

Here is an example LBAC model implementation in Casbin:

```
[request_definition]
r = sub, subject_confidentiality, subject_integrity, obj,
object_confidentiality, object_integrity, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

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

[matchers]
m = (r.act == "read" && r.subject_confidentiality >=
r.object_confidentiality && r.subject_integrity >=
```

How it works

In this example implementation, the matcher function implements access control rules based on two security properties:

- **Confidentiality Control**: Prevents unauthorized disclosure of information
- **Integrity Control**: Prevents unauthorized modification of information

The access control decisions are made by comparing the security levels of subjects and objects in both dimensions.

Examples

Request Examples

The following are example requests to demonstrate how this LBAC implementation works:

```
# Normal read operations (ALLOWED)
admin, 5, 5, file_topsecret, 3, 3, read      # admin (conf:5,
int:5) reads file_topsecret (conf:3, int:3) - ALLOWED
manager, 4, 4, file_secret, 4, 2, read       # manager (conf:4,
int:4) reads file_secret (conf:4, int:2) - ALLOWED
staff, 3, 3, file_internal, 2, 3, read       # staff (conf:3,
int:3) reads file_internal (conf:2, int:3) - ALLOWED
guest, 2, 2, file_public, 2, 2, read         # guest (conf:2,
int:2) reads file_public (conf:2, int:2) - ALLOWED

# Read operation violations (DENIED)
staff, 3, 3, file_secret, 4, 2, read         # staff (conf:3,
```

Security Levels

In this example implementation, both confidentiality and integrity levels are represented as integers where higher numbers indicate higher security levels:

Confidentiality Levels

- Level 1: Public/Unclassified
- Level 2: Confidential
- Level 3: Secret
- Level 4: Top Secret

Integrity Levels

- Level 1: Low integrity (e.g., public data, user-generated content)
- Level 2: Medium integrity (e.g., verified data, trusted sources)
- Level 3: High integrity (e.g., system data, administrative content)
- Level 4: Critical integrity (e.g., security policies, system configuration)

Use Cases

This LBAC implementation example is suitable for:

- Multi-level security environments
- Applications where data protection and data accuracy are both important

Implementation Notes

- The model enforces mandatory access control (MAC) with dual security properties
- Security levels are assigned by system administrators
- Access decisions are based on both confidentiality and integrity levels
- The model prevents both information leakage and data corruption

Priority Model

Casbin支持加载具有优先级的策略。

以隐式优先级加载策略

这非常简单：顺序决定了优先级；出现得越早的策略优先级越高。

model.conf:

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

以显式优先级加载策略

> 另见：casbin#550

优先级值越小，优先级越高。 如果优先级中有非数字字符，它将被放在最后，而不是抛出错误。

> **令牌名称约定**
>
> 在策略定义中通常使用的优先级令牌名称是“priority”。 要使用自定义的，你需要调用`e.SetFieldIndex()`并重新加载策略（参见TestCustomizedFieldIndex的完整示例）。

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`
```

目前，显式优先级仅支持`AddPolicy`和`AddPolicies`。 如果已经调用了`UpdatePolicy`，你不应该更改优先级属性。

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
```

根据角色和用户层次结构加载具有优先级的策略

角色和用户的继承结构只能是多棵树，不能是图。 如果一个用户有多个角色，你必须确保用户在不同的树中有相同的级别。 如果两个角色有相同的级别，那么先出现的策略（与角色关联）具有更高的优先级。 更多详情，另见casbin#833和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
```

角色层次结构看起来像这样:

```
role: root
 └─ role: admin
    ├─ role editor
    │  └─ user: jane
    │
    └─ role: subscriber
       └─ user: alice
```

优先级自动看起来像这样:

```
role: root                  # auto priority: 30
 └─ role: admin             # auto priority: 20
     ├─ role: editor        # auto priority: 10
     └─ role: subscriber    # auto priority: 10
```

Usage Control (UCON)

Overview

The Usage Control (UCON) model is an extension of traditional access control models that integrates authorizations, obligations, and conditions. Developed by Sandhu and Park, UCON provides a comprehensive framework for controlling access to and usage of digital resources in distributed systems. Unlike traditional access control models that focus only on authorization decisions at access time, UCON introduces continuous enforcement and attribute mutability.

Model & Policy

> **Note**: Casbin does not directly support UCON in its core library. Instead, UCON functionality is provided through the extension library casbin-ucon, which adds session-based access control with conditions, obligations, and continuous monitoring capabilities. It can use the same models and policies supported by the Casbin core.

model

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

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

policy

```
p, alice, document1, read
```

> **Note**: In UCON, Conditions and Obligations are not written in policy.csv; they are added and managed in code through the casbin-ucon API.

Core Components

The UCON model consists of three core components:

1. **Authorizations (A)**: Traditional access control rules based on subject and object attributes
2. **oBligations (B)**: Requirements that must be fulfilled by subjects before or during resource usage
3. **Conditions (C)**: Environmental or system factors independent of subjects and objects

Key Features

1. **Continuity**: Unlike traditional models that check permissions only at access time, UCON enables ongoing enforcement during the entire usage session
2. **Mutability**: Subject and object attributes can be updated as a consequence of usage
3. **Pre-decisions**: Authorizations, obligations, and conditions checked before access
4. **Ongoing-decisions**: Continuous checking during resource usage
5. **Post-decisions**: Updates to attributes after usage completion

Use Cases

The UCON model is particularly useful in:

- Digital Rights Management (DRM) systems
- Healthcare information systems
- Cloud computing environments
- IoT device access control
- Data sharing in collaborative environments
- Financial systems with complex compliance requirements

Implementation with casbin-ucon

To implement UCON in your Casbin-based application, you need to use the casbin-ucon extension library:

```
go get github.com/casbin/casbin-ucon
```

```go
// Import the required packages
import (
    "github.com/casbin/casbin/v2"
    "github.com/casbin/casbin-ucon"
    "fmt"
    "time"
)

func main() {
    // Create standard Casbin enforcer
```

Implementation Notes

- UCON encompasses traditional access control models like MAC, DAC, and RBAC
- Implementation requires a reference monitor capable of continuous monitoring
- Attribute updates (mutability) should be handled atomically to maintain consistency
- Obligations may require integration with external monitoring systems
- Conditions evaluation may depend on environmental factors outside the access control system
- When using casbin-ucon, you need to properly manage session lifecycle and monitoring

Comparison with Traditional Models

Aspect	Traditional Access Control	UCON
Decision Time	Pre-access only	Pre, ongoing, and post access
Attribute Mutability	Static	Dynamic (can change during usage)
Decision Factors	Authorizations only	Authorizations, obligations, and conditions

Aspect	Traditional Access Control	UCON
Enforcement	One-time check	Continuous monitoring
Revocation	Explicit	Can be automatic based on attribute changes

References

For complete API documentation, detailed usage, and latest updates about casbin-ucon, please refer to casbin-ucon .

Super Admin

超级管理员是整个系统的管理员。 它可以用于RBAC、ABAC和带有域的RBAC等模型中。 详细的例子如下：

```
[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"
```

这个例子说明，有了定义的`request_definition`、`policy_definition`、`policy_effect`和`matchers`，Casbin可以确定请求是否能匹配策略。 一个重要的方面是检查`sub`是否为root。 如果判断正确，授权将被授予，用户有权限执行所有操作。

类似于Linux系统中的root用户，被授权为root可以访问所有文件和设置。 如果我们想让一个`sub`对整个系统有完全的访问权限，我们可以将其指定为超级管理员，授予`sub`执行所有操作的权限。

存储

Model Storage

模型存储

Policy Storage

策略存储

Policy Subset Loading

正在加载过滤后的策略

Model Storage

与策略不同，模型只能被加载，不能被保存。 我们认为模型不是一个动态组件，不应该在运行时被修改，所以我们没有实现一个API来将模型保存到存储中。

然而，有个好消息。 我们提供了三种等效的方式来加载模型，无论是静态还是动态：

从.CONF文件加载模型

这是使用Casbin的最常见方式。 这对于初学者来说很容易理解，当你需要Casbin团队的帮助时，分享起来也很方便。

`.CONF` 文件 examples/rbac_model.conf 的内容如下：

```
[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
```

然后你可以按照以下方式加载模型文件：

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

从代码加载模型

模型可以从代码动态初始化，而不是使用`.CONF`文件。 这是一个RBAC模型的例子：

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

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

// Load the policy rules from the .CSV file adapter.
// Replace it with your adapter to avoid using files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")

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

从字符串加载模型

或者，你可以从多行字符串加载整个模型文本。 这种方法的优点是你不需要维护一个模型文件。

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

// Initialize the model from a string.
text :=
`
[request_definition]
r = sub, obj, act

[policy_definition]
p = sub, obj, act

[role_definition]
g = _, _

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

[matchers]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
`
m, _ := model.NewModelFromString(text)

// Load the policy rules from the .CSV file adapter.
// Replace it with your adapter to avoid using files.
a := fileadapter.NewAdapter("examples/rbac_policy.csv")
```

Policy Storage

在Casbin中，策略存储是作为一个适配器来实现的。

从.CSV文件加载策略

这是使用Casbin的最常见方式。 这对初学者来说很容易理解，当你向Casbin团队寻求帮助时，分享起来也很方便。

`.CSV`文件examples/rbac_policy.csv的内容如下：

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

> **备注**
>
> 如果你的文件包含逗号，你应该用双引号包裹它们。 例如：
>
> ```
> p, alice, "data1,data2", read    --correct
> p, alice, data1,data2, read        --incorrect (the whole
> phrase "data1,data2" should be wrapped in double quotes)
> ```
>
> 如果你的文件包含逗号和双引号，你应该用双引号包裹字段，并将嵌入的双引号加倍。

```
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：casbin#886

适配器API

方法	类型	描述
LoadPolicy()	基础	从存储中加载所有策略规则
SavePolicy()	基础	将所有策略规则保存到存储中
AddPolicy()	可选	向存储中添加策略规则
RemovePolicy()	可选	从存储中移除策略规则
RemoveFilteredPolicy()	可选	从存储中移除与过滤器匹配的策略规则

数据库存储格式

你的策略文件

```
p, data2_admin, data2, read
```

对应的数据库结构（如MySQL）

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

每列的含义

- `id`：数据库中的主键。 它并不存在于`casbin策略`的一部分。 它的生成方式取决于具体的适配器。
- `ptype`：它对应于`p`，`g`，`g2`等。
- `v0-v5`：列名没有具体的含义，对应于`策略csv`中从左到右的值。 列的数量取决于你自己定义的数量。 理论上，可以有无限多的列，但适配器中通常只实现了6列。 如果这对你来说还不够，请向相应的适配器仓库提交问题。

适配器详情

关于适配器API的使用和数据库表结构设计的更多细节，请访问：/docs/adapters

Policy Subset Loading

一些适配器支持过滤策略管理。 这意味着Casbin加载的策略是基于给定过滤器在数据库中存储的策略的子集。 这允许在大型、多租户环境中有效地执行策略，其中解析整个策略成为性能瓶颈。

要使用支持的适配器进行过滤策略，只需调用 `LoadFilteredPolicy` 方法。 过滤参数的有效格式取决于使用的适配器。 为了防止意外数据丢失，当加载过滤策略时，`SavePolicy` 方法被禁用。

例如，以下代码片段使用内置的过滤文件适配器和带有域的RBAC模型。 在这种情况下，过滤器将策略限制在一个域内。 除了`"domain1"`的域外，任何策略行都从加载的策略中省略：

```
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"},
}
```

还有另一种方法支持子集加载功能：`LoadIncrementalFilteredPolicy`。`LoadIncrementalFilteredPolicy` 类似于 `LoadFilteredPolicy`，但它不会清除先前加载的策略。 它只是将过滤后的策略添加到现有策略中。

Scenarios

Data Permissions

数据权限的解决方案

Menu Permissions

菜单权限示例

Data Permissions

我们有两种数据权限（过滤）的解决方案：使用隐式分配API或使用`BatchEnforce()` API。

1. 查询隐式角色或权限

当用户通过RBAC层次结构继承角色或权限，而不是在策略规则中直接分配给他们时，我们将这种分配称为“隐式”。 要查询此类隐式关系，您需要使用以下两个API：`GetImplicitRolesForUser()`和`GetImplicitPermissionsForUser()`，而不是`GetRolesForUser()`和`GetPermissionsForUser()`。 有关更多详细信息，请参阅此GitHub问题。

2. 使用`BatchEnforce()`

`BatchEnforce()`执行每个请求并在布尔数组中返回结果。

例如：

Go Node.js Java

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

```
const boolArray = await e.batchEnforce(requests);
```

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

Menu Permissions

我们首先介绍一个使用Spring Boot的菜单系统示例。 这个示例利用jCasbin来管理菜单权限。 最终，它的目标是抽象出一个专门用于菜单权限的中间件，可以扩展到Casbin支持的其他语言，如Go和Python。

1. 配置文件

你需要在`policy.csv`文件中设置角色和权限管理，以及菜单项之间的父子关系。 更多详情，请参考这个GitHub仓库。

1.1 概述

使用`policy.csv`，你可以灵活地配置角色权限和菜单结构，以实现细粒度的访问控制。 这个配置文件定义了不同角色对各种菜单项的访问权限，用户和角色之间的关联，以及菜单项之间的层次关系。

1.2 权限定义（策略）

- **策略规则**：策略以`p`为前缀定义，指定角色（`sub`）和他们在菜单项（`obj`）上的权限（`act`），以及规则的效果（`eft`），其中`allow`表示权限被授予，`deny`表示权限被拒绝。

示例：

- `p, ROLE_ROOT, SystemMenu, read, allow`表示`ROLE_ROOT`角色有权读取`SystemMenu`菜单项。
- `p, ROLE_ROOT, UserMenu, read, deny`表示`ROLE_ROOT`角色被拒绝读取`UserMenu`菜单项。

1.3 角色和用户关联

- **角色继承**：用户-角色关系和角色层次结构以`g`为前缀定义。 这允许用户从一个或多个角色继承权限。

示例：

- `g, user, ROLE_USER`表示用户`user`被分配了`ROLE_USER`角色。
- `g, ROLE_ADMIN, ROLE_USER`表示`ROLE_ADMIN`从`ROLE_USER`继承权限。

1.4 菜单项层次结构

- **菜单关系**：菜单项之间的父子关系以`g2`为前缀定义，有助于构建菜单的结构。

示例：

- `g2, UserSubMenu_allow, UserMenu`表示`UserSubMenu_allow`是`UserMenu`的子菜单。
- `g2, (NULL), SystemMenu`表示`SystemMenu`没有子菜单项，意味着它是顶级菜单项。

1.5 菜单权限继承和默认规则

使用jCasbin管理菜单权限时，父菜单和子菜单之间的权限关系遵循特定的继承规则，有两个重要的默认规则：

父菜单权限的继承：

如果一个父菜单明确地被授予`allow`权限，那么它的所有子菜单也默认为`allow`权限，除非特别标记为`deny`。 这意味着一旦一个父菜单是可访问的，它的子菜单默认也是可访问的。

处理没有直接权限设置的父菜单：

如果一个父菜单没有直接的权限设置（既没有明确允许也没有明确拒绝），但至少有一个子菜单明确被授予`allow`权限，那么父菜单被隐式地认为具有`allow`权限。 这确保了用户可以导航到这些子菜单。

1.6 特殊权限继承规则

关于角色之间权限的继承，特别是在涉及`deny`权限的场景中，必须遵循以下规则，以确保系统安全和权限的精确控制：

明确拒绝和默认拒绝的区别：

如果一个角色，如`ROLE_ADMIN`，被明确拒绝访问一个菜单项，如`AdminSubMenu_deny`（标记为`deny`），那么即使这个角色被另一个角色（例如，`ROLE_ROOT`）继承，继承的角色也不被允许访问被拒绝的菜单项。 这确保了明确的安全策略不会因为角色继承而被绕过。

默认拒绝权限的继承：

相反，如果一个角色对一个菜单项（例如，`UserSubMenu_deny`）的访问拒绝是默认的（没有明确标记为`deny`，但因为它没有被明确授予`allow`），那么当这个角色被另一个角色（例如，`ROLE_ADMIN`）继承时，继承的角色可以覆盖默认的`deny`状态，允许访问这些菜单项。

1.7 示例描述

策略：

```
p, ROLE_ROOT, SystemMenu, read, allow
p, ROLE_ROOT, AdminMenu, read, allow
p, ROLE_ROOT, UserMenu, read, deny
p, ROLE_ADMIN, UserMenu, read, allow
p, ROLE_ADMIN, AdminMenu, read, allow
p, ROLE_ADMIN, AdminSubMenu_deny, read, deny
```

菜单名称	ROLE_ROOT	ROLE_ADMIN	ROLE_USER
SystemMenu	✅	❌	❌
UserMenu	❌	✅	❌
UserSubMenu_allow	❌	✅	✅
UserSubSubMenu	❌	✅	✅
UserSubMenu_deny	❌	✅	❌
AdminMenu	✅	✅	❌
AdminSubMenu_allow	✅	✅	❌
AdminSubMenu_deny	✅	❌	❌

2. 菜单权限控制

通过MenuService中的`findAccessibleMenus()`函数，可以确定给定用户名可以访问的所有菜单项的列表。 要检查特定用户是否有权访问指定的菜单项，可以使用`checkMenuAccess()`方法。 这种方法确保了菜单权限的有效控制，利用jCasbin的能力高效地管理访问权限。

扩展功能

Enforcers

执行器是Casbin中的主要结构，它作为一个接口供用户对策略规则和模型进行操作。

Adapters

支持的适配器和使用方法

Watchers

维持多个Casbin执行器实例之间的一致性

Dispatchers

调度器提供了一种同步策略增量更改的方式。

📄 Role Managers

角色管理器用于管理Casbin中的RBAC角色层次结构。

📄 Middlewares

Casbin中间件

📄 GraphQL Middlewares

GraphQL端点的授权

📄 Cloud Native Middlewares

云原生中间件

Enforcers

`Enforcer`是Casbin中的主要结构。 它作为一个接口供用户对策略规则和模型进行操作。

支持的执行器

下面提供了Casbin执行器的完整列表。 欢迎对新执行器的任何第三方贡献。 请通知我们，我们会将其添加到此列表中:)

Go Python

执行器	作者	描述
执行器	Casbin	`Enforcer`是用户与Casbin策略和模型交互的基本结构。 你可以在这里找到有关`Enforcer` API的更多详细信息。
CachedEnforcer	Casbin	`CachedEnforcer`基于`Enforcer`，支持使用映射在内存中缓存请求的评估结果。 它提供了在指定过期时间内清除缓存的能力。 此外，它通过读写锁保证了线程安全。 你可以使用`EnableCache`来启用评估结果的缓存（默认为启用）。 `CachedEnforcer`的其他API方法与`Enforcer`相同。
DistributedEnforcer	Casbin	`DistributedEnforcer`支持分布式集群中

执行器	作者	描述
		的多个实例。 它为调度器包装了 `SyncedEnforcer`。 你可以在这里找到有关调度器的更多详细信息。
SyncedEnforcer	Casbin	`SyncedEnforcer`基于`Enforcer`，提供同步访问。 它是线程安全的。
SyncedCachedEnforcer	Casbin	`SyncedCachedEnforcer`包装了`Enforcer`，提供决策同步缓存。

执行器	作者	描述
执行器	Casbin	`Enforcer`是用户与Casbin策略和模型交互的基本结构。 你可以在这里找到有关`Enforcer` API的更多详细信息。
DistributedEnforcer	Casbin	`DistributedEnforcer`支持分布式集群中的多个实例。 它为调度器包装了`SyncedEnforcer`。 你可以在这里找到有关调度器的更多详细信息。
SyncedEnforcer	Casbin	`SyncedEnforcer`基于`Enforcer`，提供同步访问。 它是线程安全的。
AsyncEnforcer	Casbin	`AsyncEnforcer`提供异步API。
FastEnforcer	Casbin	`FastEnforcer`使用一个比普通模型快50倍的新模型。 你可以在这里找到更多信息

Adapters

在Casbin中，策略存储是作为一个适配器（也称为Casbin的中间件）来实现的。 Casbin用户可以使用适配器从存储中加载策略规则（也称为`LoadPolicy()`），或者将策略规则保存到存储中（也称为`SavePolicy()`）。 为了保持轻量级，我们没有将适配器代码放在主库中。

支持的适配器

下面提供了Casbin适配器的完整列表。 欢迎任何第三方对新适配器的贡献，请通知我们，我们会将其放入此列表：

Go Java Node.js PHP Python .NET Rust Ruby Swift Lua

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Filtered File Adapter (built-in)	File	@faceless-saint	❌	For .CSV (Comma-Separated Values) files with policy subset loading support
SQL Adapter	SQL	@Blank-Xu	✅	MySQL, PostgreSQL, SQL Server, SQLite3 are supported in `master` branch and Oracle is supported in `oracle` branch by `database/sql`
Xorm Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, TiDB, SQLite, SQL Server, Oracle are supported by Xorm
GORM	ORM	Casbin	✅	MySQL, PostgreSQL, Sqlite3, SQL

Adapter	Type	Author	AutoSave	Description
Adapter				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, Gremlin-based graph databases are supported by ent ORM
Beego ORM Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, Sqlite3 are supported by Beego ORM
SQLX Adapter	ORM	@memwey	✅	MySQL, PostgreSQL, SQLite, Oracle are supported by SQLX
Sqlx Adapter	ORM	@Blank-Xu	✅	MySQL, PostgreSQL, SQL Server, SQLite3 are supported in `master` branch and Oracle is supported in `oracle` branch by sqlx
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

Adapter	Type	Author	AutoSave	Description
				GoFrame ORM
GoFrame V2 Adapter	ORM	@hailaz	✅	MySQL, SQLite, PostgreSQL, Oracle, SQL Server are supported by GoFrame ORM
Bun Adapter	ORM	@JunNishimura	✅	MySQL, SQLite, PostgreSQL, SQL Server are supported by Bun ORM
Filtered PostgreSQL Adapter	SQL	Casbin	✅	For PostgreSQL
Filtered pgx Adapter	SQL	@pckhoi	✅	PostgreSQL is supported by pgx
Pgx Adapter	SQL	@gtoxlili	✅	PostgreSQL is supported by pgx, supports customizable column count
PostgreSQL Adapter	SQL	@cychiuae	✅	For PostgreSQL
RQLite Adapter	SQL	EDOMO Systems	✅	For RQLite
MongoDB Adapter	NoSQL	Casbin	✅	For MongoDB based on MongoDB Go Driver
RethinkDB Adapter	NoSQL	@adityapandey9	✅	For RethinkDB
Cassandra Adapter	NoSQL	Casbin	❌	For Apache Cassandra DB
DynamoDB Adapter	NoSQL	HOOQ	❌	For Amazon DynamoDB

Adapter	Type	Author	AutoSave	Description
Dynacasbin	NoSQL	NewbMiao	✅	For Amazon DynamoDB
ArangoDB Adapter	NoSQL	@adamwasila	✅	For ArangoDB
Amazon S3 Adapter	Cloud	Soluto	❌	For Minio and Amazon S3
Go CDK Adapter	Cloud	@bartventer	✅	Adapter based on Go Cloud Dev Kit that supports: Amazon DynamoDB, Azure CosmosDB, GCP Firestore, MongoDB, In-Memory
Azure Cosmos DB Adapter	Cloud	@spacycoder	✅	For Microsoft Azure Cosmos DB
GCP Firestore Adapter	Cloud	@reedom	❌	For Google Cloud Platform Firestore
GCP Cloud Storage Adapter	Cloud	qurami	❌	For Google Cloud Platform Cloud Storage
GCP Cloud Spanner Adapter	Cloud	@flowerinthenight	✅	For Google Cloud Platform Cloud Spanner
Consul Adapter	KV store	@ankitm123	❌	For HashiCorp Consul
Redis Adapter (Redigo)	KV store	Casbin	✅	For Redis

Adapter	Type	Author	AutoSave	Description
Redis Adapter (go-redis)	KV store	@mlsen	✅	For Redis
Etcd Adapter	KV store	@sebastianliu	❌	For etcd
BoltDB Adapter	KV store	@speza	✅	For Bolt
Bolt Adapter	KV store	@wirepair	❌	For Bolt
BadgerDB Adapter	KV store	@inits	✅	For BadgerDB
Protobuf Adapter	Stream	Casbin	❌	For Google Protocol Buffers
JSON Adapter	String	Casbin	❌	For JSON
String Adapter	String	@qiangmzsx	❌	For String
HTTP File Adapter	HTTP	@h4ckedneko	❌	For http.FileSystem
FileSystem Adapter	File	@naucon	❌	For fs.FS and embed.FS

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files

Adapter	Type	Author	AutoSave	Description
JDBC Adapter	JDBC	Casbin	✅	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server are supported by JDBC
Hibernate Adapter	ORM	Casbin	✅	Oracle, DB2, SQL Server, Sybase, MySQL, PostgreSQL are supported by Hibernate
MyBatis Adapter	ORM	Casbin	✅	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server (the same as JDBC) are supported by MyBatis 3
Hutool Adapter	ORM	@mapleafgo	✅	MySQL, Oracle, PostgreSQL, SQLite are supported by Hutool
MongoDB Adapter	NoSQL	Casbin	✅	MongoDB is supported by mongodb-driver-sync
DynamoDB Adapter	NoSQL	Casbin	❌	For Amazon DynamoDB
Redis Adapter	KV store	Casbin	✅	For Redis

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Filtered File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files with policy subset loading support
String Adapter (built-in)	String	@calebfaruki	❌	For String
Basic	Native	Casbin	✅	pg, mysql, mysql2, sqlite3,

Adapter	Type	Author	AutoSave	Description
Adapter	ORM			oracledb, mssql are supported by the adapter itself
Sequelize Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite, Microsoft SQL Server are supported by Sequelize
TypeORM Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, Oracle, WebSQL, MongoDB are supported by TypeORM
Prisma Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, MariaDB, SQLite, MS SQL Server, AWS Aurora, Azure SQL are supported by Prisma
Knex Adapter	ORM	knex	✅	MSSQL, MySQL, PostgreSQL, SQLite3, Oracle are supported by Knex.js
Objection.js Adapter	ORM	@willsoto	✅	MSSQL, MySQL, PostgreSQL, SQLite3, Oracle are supported by Objection.js
MikroORM Adapter	ORM	@baisheng	✅	MongoDB, MySQL, MariaDB, PostgreSQL, SQLite are supported by MikroORM
Node PostgreSQL	SQL	@touchifyapp	✅	PostgreSQL adapter with advanced policy subset

Adapter	Type	Author	AutoSave	Description
Native Adapter				loading support and improved performances built with node-postgres.
Mongoose Adapter	NoSQL	elastic.io and Casbin	✅	MongoDB is supported by Mongoose
Mongoose Adapter (No-Transaction)	NoSQL	minhducck	✅	MongoDB is supported by Mongoose
Node MongoDB Native Adapter	NoSQL	NathanBhanji	✅	For Node MongoDB Native
Node MongoDB Native Adapter	NoSQL	@juicycleff	✅	For Node MongoDB Native
DynamoDB Adapter	NoSQL	@fospitia	✅	For Amazon DynamoDB
Couchbase Adapter	NoSQL	@MarkMYoung	✅	For Couchbase
Redis Adapter	KV store	Casbin	❌	For Redis
Redis Adapter	KV store	@NandaKishorJeripothula	❌	For Redis

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Database Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite, Microsoft SQL Server are supported by techone/database
Zend Db Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite, Oracle, IBM DB2, Microsoft SQL Server, Other PDO Driver are supported by zend-db
Doctrine DBAL Adapter (Recommend)	ORM	Casbin	✅	Powerful PHP database abstraction layer (DBAL) with many features for database schema introspection and management.
Medoo Adapter	ORM	Casbin	✅	Medoo is a lightweight PHP Database Framework to Accelerate Development, supports all SQL databases, including `MySQL`, `MSSQL`, `SQLite`, `MariaDB`, `PostgreSQL`, `Sybase`, `Oracle` and more.
Laminas-db Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, Oracle, IBM DB2, Microsoft SQL Server, PDO, etc. are supported by laminas-db
Zend-db Adapter	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	❌	For Redis

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Django ORM Adapter	ORM	Casbin	✅	PostgreSQL, MariaDB, MySQL, Oracle, SQLite, IBM DB2, Microsoft SQL Server, Firebird, ODBC are supported by Django ORM
SQLObject Adapter	ORM	Casbin	✅	PostgreSQL, MySQL, SQLite, Microsoft SQL Server, Firebird, Sybase, MAX DB, pyfirebirdsql are supported by SQLObject
SQLAlchemy Adapter	ORM	Casbin	✅	PostgreSQL, MySQL, SQLite, Oracle,

Adapter	Type	Author	AutoSave	Description
				Microsoft SQL Server, Firebird, Sybase are supported by SQLAlchemy
Async SQLAlchemy Adapter	ORM	Casbin	✅	PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server, Firebird, Sybase are supported by SQLAlchemy
Async Databases Adapter	ORM	Casbin	✅	PostgreSQL, MySQL, SQLite, Oracle, Microsoft SQL Server, Firebird, Sybase are supported by Databases
Peewee Adapter	ORM	@shblhy	✅	PostgreSQL, MySQL, SQLite are supported by Peewee

Adapter	Type	Author	AutoSave	Description
MongoEngine Adapter	ORM	@zhangbailong945	❌	MongoDB is supported by MongoEngine
Pony ORM Adapter	ORM	@drorvinkler	✅	MySQL, PostgreSQL, SQLite, Oracle, CockroachDB are supported by Pony ORM
Tortoise ORM Adapter	ORM	@thearchitector	✅	PostgreSQL (>=9.4), MySQL, MariaDB, and SQLite are supported by Tortoise ORM
Async Ormar Adapter	ORM	@shepilov-vladislav	✅	PostgreSQL, MySQL, SQLite are supported by Ormar
SQLModel Adapter	ORM	@shepilov-vladislav	✅	PostgreSQL, MySQL, SQLite are supported by SQLModel
Couchbase Adapter	NoSQL	ScienceLogic	✅ (without `remove_filtered_policy()`)	For Couchbase

Adapter	Type	Author	AutoSave	Description
DynamoDB Adapter	NoSQL	@abqadeer	✅	For DynamoDB
Pymongo Adapter	NoSQL	Casbin	❌	MongoDB is supported by Pymongo
Redis Adapter	KV store	Casbin	✅	For Redis
GCP Firebase Adapter	Cloud	@devrushi41	✅	For Google Cloud Platform Firebase

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
EF Adapter	ORM	Casbin	❌	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, DB2, etc. are supported by Entity Framework 6
EFCore Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, DB2, etc. are supported by Entity Framework Core
Linq2DB Adapter	ORM	@Tirael	✅	MySQL, PostgreSQL, SQLite, Microsoft SQL Server, Oracle, Access, Firebird, Sybase, etc. are supported by linq2db
Azure	Cloud	@sagarkhandelwal	✅	For Microsoft Azure Cosmos DB

Adapter	Type	Author	AutoSave	Description
Cosmos DB Adapter				

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Diesel Adapter	ORM	Casbin	✅	SQLite, PostgreSQL, MySQL are supported by Diesel
Sqlx Adapter	ORM	Casbin	✅	PostgreSQL, MySQL are supported by Sqlx with fully asynchronous operation
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	✅	For Amazon DynamoDB
MongoDB Adapter	MongoDB	@wangjun861205	✅	For MongoDB

Adapter	Type	Author	AutoSave	Description
JSON Adapter	String	Casbin	✅	For JSON
YAML Adapter	String	Casbin	✅	For YAML
String Adapter	String	Casbin	❌	For String

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Sequel Adapter	ORM	CasbinRuby	✅	ADO, Amalgalite, IBM_DB, JDBC, MySQL, Mysql2, ODBC, Oracle, PostgreSQL, SQLAnywhere, SQLite3, and TinyTDS are supported by Sequel

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files
Memory Adapter (built-in)	Memory	Casbin	❌	For memory
Fluent Adapter	ORM	Casbin	✅	PostgreSQL, SQLite, MySQL, MongoDB are supported by Fluent

Adapter	Type	Author	AutoSave	Description
File Adapter	File	Casbin	❌	For .CSV (Comma-Separated Values)

Adapter	Type	Author	AutoSave	Description
(built-in)				files
Filtered File Adapter (built-in)	File	Casbin	❌	For .CSV (Comma-Separated Values) files with policy subset loading support
LuaSQL Adapter	ORM	Casbin	✅	MySQL, PostgreSQL, SQLite3 are supported by LuaSQL
4DaysORM Adapter	ORM	Casbin	✅	MySQL, SQLite3 are supported by 4DaysORM
OpenResty Adapter	ORM	@tom2nonames	✅	MySQL, PostgreSQL are supported by it

> **备注**
>
> 1. 如果使用显式或隐式适配器调用`casbin.NewEnforcer()`，策略将自动加载。
> 2. 您可以调用`e.LoadPolicy()`从存储中重新加载策略规则。
> 3. 如果适配器不支持`Auto-Save`功能，当您添加或删除策略时，策略规则不能自动保存回存储。您必须手动调用`SavePolicy()`来保存所有策略规则。

示例

这里我们提供了几个示例：

文件适配器（内置）

下面展示了如何从内置文件适配器初始化一个执行器：

Go PHP Rust

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

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

```php
use Casbin\Enforcer;

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

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

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

这与下面的内容相同:

Go PHP Rust

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

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

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

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

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

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

MySQL适配器

下面展示了如何从MySQL数据库初始化一个执行器。 它连接到127.0.0.1:3306上的MySQL DB，使用root和空密码。

Go Rust PHP

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

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

```rust
// https://github.com/casbin-rs/diesel-adapter
// make sure you activate feature `mysql`

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

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

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

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

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

$config = [
    // Either 'driver' with one of the following values:
    // pdo_mysql,pdo_sqlite,pdo_pgsql,pdo_oci (unstable),pdo_sqlsrv,pdo_sqlsrv,
    // mysqli,sqlanywhere,sqlsrv,ibm_db2 (unstable),drizzle_pdo_mysql
```

使用您自己的存储适配器

您可以像下面这样使用您自己的适配器:

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

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

在不同适配器之间迁移/转换

如果你想从`A`转换适配器到`B`，你可以这样做:

1.从A加载策略到内存

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

或者

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

2.将你的适配器从A转换到B

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

3.将策略从内存保存到B

```
e.SavePolicy()
```

运行时加载/保存

您可能还希望在初始化后重新加载模型，重新加载策略或保存策略：

```
// Reload the model from the model CONF file.
e.LoadModel()

// Reload the policy from file/database.
e.LoadPolicy()

// Save the current policy (usually after changed with Casbin API) back to file/
database.
e.SavePolicy()
```

AutoSave

适配器有一个叫做`Auto-Save`的功能。 当一个适配器支持`Auto-Save`时，意味着它可以支持将单个策略规则添加到存储中，或者从存储中删除单个策略规则。 这与`SavePolicy()`不同，因为后者会删除存储中的所有策略规则，并将所有Casbin执行器的策略规则保存到存储中。 因此，当策略规则的数量较大时，可能会遇到性能问题。

当适配器支持`Auto-Save`时，您可以通过`Enforcer.EnableAutoSave()`函数切换此选项。 该选项默认启用（如果适配器支持）。

> **备注**
>
> 1. `Auto-Save`功能是可选的。 适配器可以选择实现它或者不实现。
> 2. `Auto-Save`只在Casbin执行器使用的适配器支持它时才起作用。
> 3. 查看上面的适配器列表中的`AutoSave`列，以查看适配器是否支持`Auto-Save`。

这是一个如何使用`Auto-Save`的示例：

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

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

如何编写适配器

所有适配器都应实现Adapter接口，至少提供两个必需的方法：`LoadPolicy(model model.Model) error` 和 `SavePolicy(model model.Model) error`。

其他三个函数是可选的。 如果适配器支持`Auto-Save`功能，它们应该被实现。

方法	类型	描述
LoadPolicy()	必需	从存储中加载所有策略规则
SavePolicy()	必需	将所有策略规则保存到存储中
AddPolicy()	可选	将策略规则添加到存储中
RemovePolicy()	可选	从存储中删除策略规则
RemoveFilteredPolicy()	可选	从存储中删除与过滤器匹配的策略规则

> **备注**
>
> 如果适配器不支持`Auto-Save`，它应该为这三个可选函数提供一个空的实现。 这是一个Golang的示例：

```go
// AddPolicy adds a policy rule to the storage.
func (a *Adapter) AddPolicy(sec string, ptype string, rule []string) error {
    return errors.New("not implemented")
}

// RemovePolicy removes a policy rule from the storage.
func (a *Adapter) RemovePolicy(sec string, ptype string, rule []string) error {
    return errors.New("not implemented")
}

// RemoveFilteredPolicy removes policy rules that match the filter from the
storage.
func (a *Adapter) RemoveFilteredPolicy(sec string, ptype string, fieldIndex
```

Casbin执行器在调用这三个可选函数时会忽略`not implemented`错误。

关于如何编写适配器的详细信息。

- 数据结构。 适配器应支持至少读取六列。
- 数据库名称。 默认的数据库名称应该是`casbin`。
- 表名。 默认的表名应该是`casbin_rule`。
- Ptype列。 此列的名称应为`ptype`，而不是`p_type`或`Ptype`。
- 表定义应为`(id int primary key, ptype varchar, v0 varchar, v1 varchar, v2 varchar, v3 varchar, v4 varchar, v5 varchar)`。
- 唯一键索引应在`ptype,v0,v1,v2,v3,v4,v5`列上建立。
- `LoadFilteredPolicy`需要一个`filter`作为参数。 过滤器应该像这样。

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

谁负责创建数据库?

按照惯例，如果数据库不存在，适配器应能够自动创建名为`casbin`的数据库，并用它来存储策略。 请参考Xorm适配器作为参考实现：https://github.com/casbin/xorm-adapter

上下文适配器

ContextAdapter为Casbin适配器提供了一个上下文感知的接口。

通过上下文，您可以为适配器API实现诸如超时控制等功能

示例

gormadapter支持带有上下文的适配器，以下是使用上下文实现的超时控制

```
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()
```

如何编写一个上下文适配器

`ContextAdapter` API只比普通的`Adapter` API多了一层上下文处理，在实现普通Adapter API的基础上，你可以为上下文封装你自己的处理逻辑

简单引用 `gormadapter`： adapter.go

Watchers

我们支持使用分布式消息系统，如etcd，来维持多个Casbin执行器实例之间的一致性。 这使我们的用户能够并发使用多个Casbin执行器来处理大量的权限检查请求。

与策略存储适配器类似，我们并未在主库中包含观察者代码。 对新的消息系统的任何支持都应作为观察者来实现。 下面提供了Casbin观察者的完整列表。 我们欢迎任何第三方为新的观察者做出贡献，请通知我们，我们会将其添加到此列表中：

Go Java Node.js Python .NET Ruby PHP

Watcher	Type	Author	Description
PostgreSQL WatcherEx	Database	@IguteChung	WatcherEx for PostgreSQL
Redis WatcherEx	KV store	Casbin	WatcherEx for Redis
Redis Watcher	KV store	@billcobbler	Watcher for Redis
Etcd Watcher	KV store	Casbin	Watcher for etcd
TiKV Watcher	KV store	Casbin	Watcher for TiKV
Kafka Watcher	Messaging system	@wgarunap	Watcher for Apache Kafka
NATS Watcher	Messaging system	Soluto	Watcher for NATS
ZooKeeper Watcher	Messaging	Grepsr	Watcher for Apache ZooKeeper

Watcher	Type	Author	Description
	system		
NATS, RabbitMQ, GCP Pub/Sub, AWS SNS & SQS, Kafka, InMemory	Messaging System	@rusenask	Watcher based on Go Cloud Dev Kit that works with leading cloud providers and self-hosted infrastructure
NATS, RabbitMQ, GCP Pub/Sub, AWS SNS & SQS, Kafka, InMemory	Messaging System	@bartventer	WatcherEx based on Go Cloud Dev Kit that works with leading cloud providers and self-hosted infrastructure
RocketMQ Watcher	Messaging system	@fmyxyz	Watcher for Apache RocketMQ

Watcher	Type	Author	Description
Etcd Adapter	KV store	@mapleafgo	Watcher for etcd
Redis Watcher	KV store	Casbin	Watcher for Redis
Redis WatcherEx	KV store	Casbin	WatcherEx for Redis
Lettuce-Based Redis Watcher	KV store	Casbin	Watcher for Redis based on Lettuce)
PostgreSQL Watcher	Database	Casbin	Watcher for PostgreSQL
Kafka Watcher	Messaging system	Casbin	Watcher for Apache Kafka

Watcher	Type	Author	Description
Etcd Watcher	KV store	Casbin	Watcher for etcd
Redis Watcher	KV store	Casbin	Watcher for Redis
Pub/Sub Watcher	Messaging system	Casbin	Watcher for Google Cloud Pub/Sub
MongoDB Change Streams Watcher	Database	Casbin	Watcher for MongoDB Change Streams
Postgres Watcher	Database	@mcollina	Watcher for PostgreSQL

Watcher	Type	Author	Description
Etcd Watcher	KV store	Casbin	Watcher for etcd
Redis Watcher	KV store	Casbin	Watcher for Redis
Redis Watcher	KV store	ScienceLogic	Watcher for Redis
Redis Async Watcher	KV store	@kevinkelin	Watcher for Redis
PostgreSQL Watcher	Database	Casbin	Watcher for PostgreSQL
RabbitMQ Watcher	Messaging system	Casbin	Watcher for RabbitMQ

Watcher	Type	Author	Description
Redis Watcher	KV store	@Sbou	Watcher for Redis

Watcher	Type	Author	Description
Redis Watcher	KV store	CasbinRuby	Watcher for Redis
RabbitMQ Watcher	Messaging system	CasbinRuby	Watcher for RabbitMQ

Watcher	Type	Author	Description
Redis Watcher	KV store	@Tinywan	Watcher for Redis

WatcherEx

为了支持多个实例之间的增量同步，我们提供了`WatcherEx`接口。 我们希望它能在策略发生变化时通知其他实例，但目前还没有`WatcherEx`的实现。 我们建议您使用调度器来实现这一点。

与`Watcher`接口相比，`WatcherEx`可以区分接收到的更新操作的类型，例如，`AddPolicy`和`RemovePolicy`。

WatcherEx Apis：

API	描述
SetUpdateCallback(func(string)) error	SetUpdateCallback设置观察者在策略在数据库中被其他实例更改时将调用的回调函数。 一个典型的回调是Enforcer.LoadPolicy()。
Update() error	Update调用其他实例的更新回调以同步其策略。 通常在数据库中更改策略后调用，如Enforcer.SavePolicy()，Enforcer.AddPolicy()，Enforcer.RemovePolicy()等。
Close()	Close停止并释放观察者，回调函数将不再被调用。

API	描述
UpdateForAddPolicy(sec, ptype string, params ...string) error	UpdateForAddPolicy调用其他实例的更新回调以同步其策略。 在通过Enforcer.AddPolicy(), Enforcer.AddNamedPolicy(), Enforcer.AddGroupingPolicy()和Enforcer.AddNamedGroupingPolicy()添加策略后调用。
UpdateForRemovePolicy(sec, ptype string, params ...string) error	UpdateForRemovePolicy调用其他实例的更新回调以同步其策略。 在通过Enforcer.RemovePolicy(), Enforcer.RemoveNamedPolicy(), Enforcer.RemoveGroupingPolicy()和Enforcer.RemoveNamedGroupingPolicy()删除策略后调用。
UpdateForRemoveFilteredPolicy(sec, ptype string, fieldIndex int, fieldValues ...string) error	UpdateForRemoveFilteredPolicy调用其他实例的更新回调以同步其策略。 在Enforcer.RemoveFilteredPolicy(), Enforcer.RemoveFilteredNamedPolicy(), Enforcer.RemoveFilteredGroupingPolicy()和Enforcer.RemoveFilteredNamedGroupingPolicy()之后调用。
UpdateForSavePolicy(model model.Model) error	UpdateForSavePolicy调用其他实例的更新回调以同步其策略。 在Enforcer.SavePolicy()之后调用。
UpdateForAddPolicies(sec string, ptype string, rules ...[]string) error	UpdateForAddPolicies调用其他实例的更新回调以同步其策略。 在Enforcer.AddPolicies(), Enforcer.AddNamedPolicies(), Enforcer.AddGroupingPolicies()和Enforcer.AddNamedGroupingPolicies()之后调用。

API	描述
UpdateForRemovePolicies(sec string, ptype string, rules ...[]string) error	UpdateForRemovePolicies调用其他实例的更新回调以同步其策略。 在Enforcer.RemovePolicies(), Enforcer.RemoveNamedPolicies(), Enforcer.RemoveGroupingPolicies()和Enforcer.RemoveNamedGroupingPolicies()之后调用。

Dispatchers

调度器提供了一种同步策略增量更改的方式。 它们应基于一致性算法，如Raft，以确保所有执行器实例的一致性。 通过调度器，用户可以轻松建立分布式集群。

调度器的方法分为两部分。 第一部分是与Casbin结合的方法。 这些方法应在Casbin内部调用。 用户可以使用Casbin本身提供的更完整的API。

另一部分是调度器本身定义的方法，包括调度器初始化方法，以及不同算法提供的不同功能，如动态成员和配置更改。

> **备注**
>
> 我们希望调度器只确保运行时Casbin执行器的一致性。 所以，如果在初始化期间策略不一致，调度器将无法正常工作。 用户需要在使用调度器之前确保所有实例的状态一致。

下面提供了Casbin调度器的完整列表。 欢迎对新的调度器进行任何第三方贡献。 请通知我们，我们会将其添加到此列表中。

Go

Adapter	Type	Author	Description
Hashicorp Raft Dispatcher	Raft	Casbin	A dispatcher based on Hashicorp Raft
KDKYG/casbin-	Raft	@KDKYG	A dispatcher based on

Adapter	Type	Author	Description
dispatcher			Hashicorp Raft

DistributedEnforcer

DistributedEnforcer为调度器包装了SyncedEnforcer。

Go

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

Role Managers

角色管理器用于管理Casbin中的RBAC角色层次结构（用户-角色映射）。 角色管理器可以从Casbin策略规则或外部源（如LDAP，Okta，Auth0，Azure AD等）检索角色数据。 我们支持不同的角色管理器实现。 为了保持轻量级，我们没有在主库中包含角色管理器代码（除默认角色管理器外）。 下面提供了Casbin角色管理器的完整列表。 欢迎任何第三方为新的角色管理器做出贡献。 请通知我们，我们会将其添加到此列表中：)

Go Java Node.js PHP Python

Role manager	Author	Description
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	Supports role hierarchy stored in Okta
Auth0 Role Manager	Casbin	Supports role hierarchy stored in Auth0's Authorization Extension

对于开发者：所有角色管理器必须实现RoleManager接口。 Session Role Manager可以作为参考实现。

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

对于开发者：所有角色管理器必须实现RoleManager接口。 Default Role Manager可以作为参考实现。

Role manager	Author	Description
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

对于开发者：所有角色管理器必须实现RoleManager接口。 Default Role Manager可以作为参考实现。

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

对于开发者：所有角色管理器必须实现RoleManager接口。 Default Role Manager可以作为参考实现。

Role manager	Author	Description
Default Role Manager (built-in)	Casbin	Supports role hierarchy stored in the Casbin policy

对于开发者：所有角色管理器必须实现RoleManager接口。 Default Role Manager可以作为参考实现。

API

请参阅API部分以获取详细信息。

Middlewares

Web框架

Name	Description
Gin	A HTTP web framework featuring a Martini-like API with much better performance, via plugin: authz or gin-casbin
Beego	An open-source, high-performance web framework for Go, via built-in plugin: plugins/authz
Caddy	Fast, cross-platform HTTP/2 web server with automatic HTTPS, via plugin: caddy-authz
Traefik	The cloud native application proxy, via plugin: traefik-auth-plugin
Kratos	Your ultimate Go microservices framework for the cloud-native era, via plugin: tx7do/kratos-casbin or overstarry/kratos-casbin
Go kit	A toolkit for microservices, via built-in plugin: plugins/authz

Name	Description
Fiber	An Express inspired web framework written in Go, via middleware: casbin in gofiber/contrib or fiber-casbinrest or fiber-boilerplate or gofiber-casbin
FastHTTP	Fast HTTP package for Go. Tuned for high performance. Zero memory allocations in hot paths. Up to 10x faster than net/http, via plugin: fasthttp-auth
Revel	A high productivity, full-stack web framework for the Go language, via plugin: auth/casbin
Echo	High performance, minimalist Go web framework, via plugin: echo-authz or echo-casbin or casbinrest or echo-boilerplate
Iris	The fastest web framework for Go in (THIS) Earth. HTTP/2 Ready-To-GO, via plugin: casbin or iris-middleware-casbin
GoFrame	A modular, powerful, high-performance and enterprise-class application development framework of Golang, via plugin: gf-casbin
Negroni	Idiomatic HTTP Middleware for Golang, via plugin: negroni-authz
Chi	A lightweight, idiomatic and composable router for building HTTP services, via plugin: chi-authz
Buffalo	A Go web development eco-system, designed to make your life easier, via plugin: buffalo-mw-rbac

Name	Description
Macaron	A high productive and modular web framework in Go, via plugin: authz
DotWeb	Simple and easy go web micro framework, via plugin: authz
Tango	Micro & pluggable web framework for Go, via plugin: authz
Baa	An express Go web framework with routing, middleware, dependency injection and http context, via plugin: authz
Tyk	An open source Enterprise API Gateway, supporting REST, GraphQL, TCP and gRPC protocols, via plugin: tyk-authz
Hertz	Go HTTP framework with high-performance and strong-extensibility for building micro-services, via plugin: casbin

Name	Description
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	A powerful and easy-to-use Java security framework that performs authentication, authorization, cryptography, and session management, via plugin: shiro-casbin or shiro-jcasbin-spring-boot-starter
JFinal	A simple, light, rapid, independent and extensible Java WEB + ORM

Name	Description
	framework, via plugin: jfinal-authz
Nutz	Web framework (MVC/IOC/AOP/DAO/JSON) for all Java developers, via plugin: nutz-authz
mangoo I/O	An intuitive, lightweight, high performance full stack Java web framework, via built-in plugin: AuthorizationService.java

Name	Description
Shield	An authZ server and authZ aware reverse-proxy built on top of casbin.
Express	Fast, unopinionated, minimalist web framework for node, via plugin: express-authz
Koa	Expressive middleware for node.js using ES2017 async functions, via plugin: koa-authz or koajs-starter or koa-casbin
LoopBack 4	A highly extensible Node.js and TypeScript framework for building APIs and microservices, via plugin: loopback4-authorization
Nest	Progressive Node.js framework for building efficient and scalable server-side applications on top of TypeScript & JavaScript. via plugin: nest-authz or nest-casbin or NestJS Casbin Module or nestjs-casbin or acl-nest or nestjs-casbin-typeorm

Name	Description
Fastify	Fast and low overhead web framework, for Node.js. via plugin: fastify-casbin or fastify-casbin-rest
Egg	Born to build better enterprise frameworks and apps with Node.js & Koa, via plugin: egg-authz or egg-zrole
hapi	The Simple, Secure Framework Developers Trust. via plugin: hapi-authz
Casbin JWT Express	Authorization middleware that uses stateless JWT token to validate ACL rules using Casbin
Hono	Fast, lightweight, built on Web Standards. via plugin: @hono/casbin

Name	Description
Laravel	The PHP framework for web artisans, via plugin: laravel-authz
Yii PHP Framework	A fast, secure, and efficient PHP framework, via plugin: yii-permission or yii-casbin
CakePHP	Build fast, grow solid PHP Framework, via plugin: cake-permission
CodeIgniter	Associate users with roles and permissions in CodeIgniter4 Web Framework, via plugin: CodeIgniter Permission

Name	Description
ThinkPHP 5.1	The ThinkPHP 5.1 framework, via plugin: think-casbin
ThinkPHP 6.0	The ThinkPHP 6.0 framework, via plugin: think-authz
Symfony	The Symfony PHP framework, via plugin: symfony-permission or 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
EasySwoole	A distributed, persistent memory PHP framework based on the Swoole extension, via plugin: easyswoole-permission or easyswoole-hyperfOrm-permission
Slim	A PHP micro framework that helps you quickly write simple yet powerful web applications and APIs, via plugin: casbin-with-slim
Phalcon	A full-stack PHP framework delivered as a C-extension, via plugin: phalcon-permission
Webman	High performance HTTP Service Framework for PHP based on Workerman, via plugin: webman-permission or webman-casbin

Name	Description
Django	A high-level Python Web framework, via plugin: django-casbin or django-casbin-auth
Flask	A microframework for Python based on Werkzeug, Jinja 2 and good intentions, via plugin: flask-authz or Flask-Casbin (3rd-party, but maybe more friendly) or rbac-flask
FastAPI	A modern, fast (high-performance), web framework for building APIs with Python 3.6+ based on standard Python type hints, via plugin: fastapi-casbin-auth or Fastapi-app
OpenStack	The most widely deployed open source cloud software in the world, via plugin: openstack-patron
Tornado	Tornado is a Python web framework and asynchronous networking library, via plugin: tornado-authz

Name	Description
Nginx	A HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server, via plugin: nginx-casbin-module

Name	Description
ASP.NET Core	An open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends, via plugin: Casbin.AspNetCore

Name	Description
ASP.NET Core	A simple demo of using Casbin at ASP.NET Core framework, via plugin: CasbinACL-aspNetCore

Name	Description
Actix	A Rust actors framework, via plugin: actix-casbin
Actix web	A small, pragmatic, and extremely fast rust web framework, via plugin: actix-casbin-auth
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 web	A ergonomic and modular rust web framework, via plugin: axum-casbin-auth
Poem web	A full-featured and easy to use web framework with the Rust programming language, via plugin: poem-casbin

Name	Description
OpenResty	A dynamic web platform based on NGINX and LuaJIT, via plugin: lua-resty-casbin and casbin-openresty-example
Kong	A cloud-native, platform-agnostic, scalable API Gateway distinguished for its high performance and extensibility via plugins, via plugin: kong-authz

Name	Description
APISIX	A dynamic, real-time, high-performance API gateway, via plugin: authz-casbin

Name	Description
Vapor	A server-side Swift web framework, via plugin: vapor-authz

云服务提供商

Node.js

名称	描述
Okta	一个可信赖的平台，用于通过插件保护每一个身份：casbin-spring-boot-demo
Auth0	一个易于实施，可适应的身份验证和授权平台，通过插件：casbin-auth0-rbac

GraphQL Middlewares

Casbin遵循官方建议的方式为GraphQL端点提供授权，将授权的唯一真实来源设定为：https://graphql.org/learn/authorization/。 换句话说，Casbin应该位于GraphQL层和您的业务逻辑之间。

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

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

支持的GraphQL中间件

下面提供了Casbin GraphQL中间件的完整列表。 欢迎任何第三方对新的GraphQL中间件的贡献。 请通知我们，我们会将其添加到此列表中：）

Go　Node.js　Python

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	Graphql和Casbin一起使用的实现
gqlgen_casbin_RBAC_example	gqlgen	@WenyXu	(空)

中间件	GraphQL实现	作者	描述
graphql-authz	GraphQL.js	Casbin	用于GraphQL.js的Casbin授权中间件

中间件	GraphQL实现	作者	描述
graphql-authz	GraphQL-core 3	@Checho3388	用于GraphQL-core 3的Casbin授权中间件

Cloud Native Middlewares

云原生项目

Go Node.js

项目	作者	描述
k8s-authz	Casbin	用于Kubernetes的授权中间件
envoy-authz	Casbin	用于Istio和Envoy的授权中间件
kubesphere-authz	Casbin	用于kubeSphere的授权中间件

项目	作者	描述
ODPF Shield	Open Data Platform	ODPF Shield是一个云原生的基于角色的授权感知反向代理服务。

API

API Overview

Casbin API 使用

Management API

提供对Casbin策略管理的全面支持的原始API

RBAC API

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

RBAC with Domains API

一个更用户友好的RBAC与域的API。 此API是管理API的一个子集。 RBAC用户可以使用此API简化他们的代码。

RBAC with Conditions API

一个更加用户友好的带有条件的RBAC API。

RoleManager API

RoleManager API提供了一个定义管理角色操作的接口。 向RoleManager添加匹配函数后，可以在角色名称和域中使用通配符。

API Overview

此概述仅向您展示如何使用 Casbin API，并不解释如何安装 Casbin 或其工作原理。 您可以在这里找到这些教程：Casbin 的安装 和 Casbin 的工作原理。 所以，当您开始阅读这个教程时，我们假设您已经完全安装并导入了 Casbin 到您的代码中。

Enforce API

让我们从 Casbin 的 Enforce API 开始。 我们将从 `model.conf` 加载一个 RBAC 模型，并从 `policy.csv` 加载策略。 您可以在这里了解模型语法，我们在这个教程中不会讨论它。 我们假设您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]
m = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act
```

`policy.csv`

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

阅读配置文件后，请阅读以下代码。

```go
// 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")
```

此代码从本地文件加载访问控制模型和策略。 `casbin.NewEnforcer()` 函数将返回一个执行器。 它会将其两个参数识别为文件路径，并从那里加载文件。 过程中发生的错误存储在变量 `err` 中。 此代码使用默认适配器加载模型和策略，当然，您也可以使用第三方适配器达到相同的效果。

`ok, err := enforcer.Enforce("alice", "data1", "read")` 代码用于确认访问权限。 如果 Alice 可以使用读操作访问 data1， `ok` 的返回值将为 `true`；否则，将为 `false`。 在此示例中， `ok` 的值为 `true`。

EnforceEx API

有时您可能想知道是哪个策略允许了请求，所以我们准备了 `EnforceEx()` 函数。 您可以这样使用它:

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

`EnforceEx()` 函数将在返回值 `reason` 中返回确切的策略字符串。 在此示例中，`amber` 是一个 `admin` 角色，所以策略 `p, admin, data1, read` 允许此请求为 `true`。 此代码的输出在注释中。

Casbin 提供了许多类似于此的 API。 这些 API 在基本功能上增加了一些额外的功能。 它们包括:

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

  此函数使用一个匹配器。

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

  这是`EnforceWithMatcher()`和`EnforceEx()`的组合。

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

  此函数允许一系列的任务，并返回一个数组。

这是Casbin的一个简单用例。 你可以使用Casbin使用这些API启动一个授权服务器。 我们将在接下来的段落中向你展示一些其他类型的API。

管理API

获取API

这些API用于在策略中检索特定对象。 在这个例子中，我们正在加载一个执行器并从中检索一些东西。

请看下面的代码:

```
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)
```

与前面的例子类似，前四行用于从本地文件加载必要的信息。 我们不会在这里进一步讨论。

代码`allSubjects := enforcer.GetAllSubjects()`检索策略文件中的所有主题，并将它们作为数组返回。 然后我们打印那个数组。

通常，代码的输出应该是:

```
[admin alice bob]
```

你也可以将函数`GetAllSubjects()`更改为`GetAllNamedSubjects()`，以获取当前命名策略中出现的主题列表。

同样，我们为`Objects, Actions, Roles`准备了`GetAll`函数。 要访问这些函数，你只需要在函数名中将`Subject`替换为所需的类别。

此外，还有更多的策略获取器可用。 调用方法和返回值与上述类似。

- `policy = e.GetPolicy()`检索策略中的所有授权规则。
- `filteredPolicy := e.GetFilteredPolicy(0, "alice")`检索策略中具有指定字段过滤器的所有授权规则。
- `namedPolicy := e.GetNamedPolicy("p")`检索命名策略中的所有授权规则。
- `filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "bob")`检索具有指定字段过滤器的命名策略中的所有授权规则。
- `groupingPolicy := e.GetGroupingPolicy()`检索策略中的所有角色继承规则。
- `filteredGroupingPolicy := e.GetFilteredGroupingPolicy(0, "alice")`检索具有指定字段过滤器的策略中的所有角色继承规则。
- `namedGroupingPolicy := e.GetNamedGroupingPolicy("g")`检索策略中的所有角色继承规则。
- `namedGroupingPolicy := e.GetFilteredNamedGroupingPolicy("g", 0, "alice")`检索具有指定字段过滤器的策略中的所有角色继承规则。

添加，删除，更新 API

Casbin提供了各种API，用于在运行时动态添加，删除或修改策略。

以下代码演示了如何添加，删除和更新策略，以及如何检查策略是否存在：

```
// load information from files
enforcer, err := casbin.NewEnforcer("./example/model.conf",
"./example/policy.csv")
if err != nil {
```

通过使用这些API，您可以动态编辑您的策略。 同样，我们也为`FilteredPolicy, NamedPolicy, FilteredNamedPolicy, GroupingPolicy, NamedGroupingPolicy, FilteredGroupingPolicy, FilteredNamedGroupingPolicy`提供了类似的API。 要使用它们，只需将函数名称中的`Policy`替换为适当的类别。

此外，通过将参数更改为数组，您可以批量编辑您的策略。

例如，考虑像这样的函数：

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

如果我们将`Policy`更改为`Policies`并按如下方式修改参数：

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

那么我们可以批量编辑这些策略。

同样的操作也可以应用于`GroupingPolicy, NamedGroupingPolicy`。

AddEx API

Casbin提供了AddEx系列API，以帮助用户批量添加规则。

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

这些方法与没有Ex后缀的方法的区别在于，如果其中一个规则已经存在，它们将继续检查下一个规则，而不是立即返回false。

例如，让我们比较`AddPolicies`和`AddPoliciesEx`。

您可以通过将以下代码复制到casbin下的测试中来运行并观察。

```
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
    ok, _ = e.AddPoliciesEx([][]string{{"user1", "data1",
"read"}, {"user2", "data2", "read"}})
    fmt.Println(e.GetPolicy())
```

RBAC API

Casbin为您提供了一些API，以便修改RBAC模型和策略。 如果您熟悉RBAC，您可以轻松使用这些API。

在这里，我们只向您展示如何使用Casbin的RBAC API，而不讨论RBAC本身。 您可以在这里获取更多详细信息。

我们使用以下代码来加载模型和策略，就像以前一样。

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

然后，我们可以使用Enforcer `enforcer`的实例来访问这些API。

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

`GetRolesForUser()`返回一个包含amber所有角色的数组。 在这个例子中，amber只有一个角色，那就是admin，所以数组`roles`是`[admin]`。 同样，您可以使用`GetUsersForRole()`来获取属于某个角色的用户。 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admin的成员，所以函数的返回值是 `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()` 来删除一个权限。

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

并使用 `DeletePermissionForUser()` 来为用户删除一个权限。

Casbin有许多像这样的API。 他们的调用方法和返回值与上述API的风格相同。 您可以在下一个文档中找到这些API。

Management API

提供对Casbin策略管理的全面支持的原始API

Filtered API

几乎所有的过滤api都有相同的参数 `(fieldIndex int, fieldValues ...string)`。 `fieldIndex`是匹配开始的索引， `fieldValues`表示结果应具有的值。 请注意，fieldValues中的空字符串可以是任何单词。

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") // will return: [["alice", "book", "read"], ["bob", "book", "read"], ["bob", "book", "write"]]

e.GetFilteredPolicy(1, "book", "read") // will return: [["alice", "book", "read"], ["bob", "book", "read"]]

e.GetFilteredPolicy(0, "alice", "", "read") // will return: [[alice book read]]

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

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决定一个“主体”是否可以用“操作”访问一个“对象”，输入参数通常是：(sub, obj, act)。

例如：

Go Node.js PHP Python Java

```
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一个“主体”是否可以用“操作”访问一个“对象”，输入参数通常是：(matcher, sub, obj, act)，当matcher为空字符串时，默认使用模型匹配器。

例如：

Go PHP Python Java

```
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

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

```node
const ok = await e.enforceEx(request);
```

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

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

`EnforceExWithMatcher()`

EnforceExWithMatcher使用自定义匹配器并通过通知匹配的规则来解释执行。

例如:

Go

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

`BatchEnforce()`

BatchEnforce执行每个请求并在布尔数组中返回结果

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获取出现在当前策略中的主题列表。

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出现的主题列表。

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获取当前策略中出现的对象列表。

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出现的对象列表。

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获取当前策略中出现的操作列表。

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出现的操作列表。

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获取当前策略中出现的角色列表。

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出现的角色列表。

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，可以指定字段过滤器。

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，可以指定字段过滤器。

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确定是否存在命名的授权规则。

例如:

Go Node.js PHP Python .NET Rust Java

```
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将授权规则添加到当前策略。 如果规则已经存在，函数返回false，规则不会被添加。否则，函数通过添加新规则返回true。

例如：

Go Node.js PHP Python .NET Rust Java

```
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()`

AddPolicies将授权规则添加到当前策略。 该操作本质上是原子性的。 因此，如果授权规则包含与当前策略不一致的规则，函数返回false，没有策略规则被添加到当前策略。 如果所有的授权规则与策略规则一致，函数返回true，并将每个策略规则添加到当前策略。

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将授权规则添加到当前策略。 如果规则已经存在，规则将不会被添加。 但与AddPolicies不同，其他不存在的规则会被添加，而不是直接返回false

例如：

Go

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

`AddNamedPolicy()`

AddNamedPolicy将授权规则添加到当前命名策略。 如果规则已经存在，函数将返回false并且规则不会被添加。 否则，函数通过添加新规则返回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将授权规则添加到当前命名策略。 该操作本质上是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false并且不会有策略规则添加到当前策略。 如果所有的授权规则都与策略规则一致，函数将返回true并且每个策略规则都会被添加到当前策略。

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'],
```

```
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将授权规则添加到当前命名策略。 如果规则已经存在，规则将不会被添加。 但与AddNamedPolicies不同，其他不存在的规则会被添加，而不是直接返回false

例如：

Go

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

`SelfAddPoliciesEx()`

SelfAddPoliciesEx将授权规则添加到当前命名策略，且自动通知观察者功能被禁用。 如果规则已经存在，规则将不会被添加。 但与SelfAddPolicies不同，其他不存在的规则会被添加，而不是直接返回false

例如：

Go

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

`RemovePolicy()`

RemovePolicy从当前策略中移除一个授权规则。

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从当前策略中移除授权规则。 该操作的性质是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false，并且没有策略规则从当前策略中移除。 如果所有的授权规则与策略规则一致，函数将返回true，并且每个策略规则都会从当前策略中移除。

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'],
```

```
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从当前策略中移除一个授权规则，可以指定字段过滤器。 RemovePolicy从当前策略中移除一个授权规则。

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从当前命名策略中移除一个授权规则。

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从当前命名策略中移除授权规则。 该操作的性质是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false，并且没有策略规则从当前策略中移除。 如果所有的授权规则与策略规则一致，函数将返回true，并且每个策略规则都会从当前策略中移除。

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.RemoveNamedPolicies("p", rules)
```

```
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一个授权规则，可以指定字段过滤器。

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.RemoveFilteredNamedPolicyAync("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，函数将返回false，规则不会被添加。 否则，函数通过添加新规则返回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()`

AddGroupingPolicies将角色继承规则添加到当前策略。 该操作本质上是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false，且不会向当前策略添加策略规则。 如果所有授权规则与策略规则一致，函数将返回true，并将每个策略规则添加到当前策略。

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将角色继承规则添加到当前策略。 如果规则已经存在，规则不会被添加。 但与AddGroupingPolicies不同，其他不存在的规则会被添加，而不是直接返回false

例如：

Go

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

`AddNamedGroupingPolicy()`

AddNamedGroupingPolicy将命名的角色继承规则添加到当前策略。 如果规则已经存在，函数将返回false，规则不会被添加。 否则，函数通过添加新规则返回true。

例如：

Go Node.js PHP Python .NET Rust Java

```
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将命名的角色继承规则添加到当前策略。 该操作本质上是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false，且不会向当前策略添加策略规则。 如果所有授权规则与策略规则一致，函数将返回true，并将每个策略规则添加到当前策略。

例如：

Go Node.js Python Rust Java

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

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

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

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

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

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

```rust
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?;
```

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

`AddNamedGroupingPoliciesEx()`

AddNamedGroupingPoliciesEx将命名的角色继承规则添加到当前策略。 如果规则已经存在，规则不会被添加。 但与AddNamedGroupingPolicies不同，其他不存在的规则会被添加，而不是直接返回false

例如：

Go

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

`RemoveGroupingPolicy()`

RemoveGroupingPolicy从当前策略中删除一个角色继承规则。

例如：

Go Node.js PHP Python .NET Rust Java

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

```node.js
const removed = await e.removeGroupingPolicy('alice', 'data2_admin')
```

```php
$removed = $e->removeGroupingPolicy("alice", "data2_admin");
```

```python
removed = e.remove_grouping_policy("alice", "data2_admin")
```

```net
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()`

RemoveGroupingPolicies从当前策略中删除角色继承规则。 该操作的性质是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false，并且不会从当前策略中删除任何策略规则。 如果所有的授权规则都与策略规则一致，函数将返回true，并且每个策略规则都将从当前策略中删除。

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从当前策略中删除一个角色继承规则，可以指定字段过滤器。

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从当前命名策略中删除一个角色继承规则。

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从当前策略中删除命名的角色继承规则。 该操作的性质是原子性的。 因此，如果授权规则包含的规则与当前策略不一致，函数将返回false，并且不会从当前策略中删除任何策略规则。 如果所有的授权规则都与策略规则一致，函数将返回true，并且每个策略规则都将从当前策略中删除。

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从当前命名策略中删除一个角色继承规则，可以指定字段过滤器。

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将旧策略更新为新策略。

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将所有旧策略更新为新策略。

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)
```

`UpdateNamedPolicy()`

UpdateNamedPolicy update a old policy to new policy in the named policy.

`例如:`

Go Node.js Python Java

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

```
const update = await e.updateNamedPolicy('p', ["eve", "data3",
"read"], ["eve", "data3", "write"]);
```

```
updated = e.update_named_policy("p", ["eve", "data3", "read"], ["eve",
"data3", "write"])
```

```
boolean updated = e.updateNamedPolicy("p", Arrays.asList("eve",
"data3", "read"), Arrays.asList("eve", "data3", "write"));
```

`UpdateNamedPolicies()`

UpdateNamedPolicies updates all old policies to new policies in the named policy.

For example:

Go Python

```
updated, err := e.UpdateNamedPolicies("p", [][]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_named_policies("p", old_rules, new_rules)
```

`AddFunction()`

`AddFunction`添加一个自定义函数。

For example:

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) {
```

```
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: STring, 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/数据库加载过滤策略。

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()`

`LoadIncrementalFilteredPolicy`从文件/数据库追加一个过滤策略。

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`更新`g`部分的oldRule为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`更新名为`ptype`的oldRule为`g`部分的newRule。

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()`

`SetFieldIndex`支持`sub`、`obj`、`domain`和`priority`的常规名称和位置的自定义。

```
[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。 此API是管理API的一个子集。 RBAC用户可以使用此API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获取用户拥有的角色。

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); // No domain
```

```
List<String> res = e.getRolesForUser("alice");
```

`GetUsersForRole()`

GetUsersForRole获取拥有某个角色的用户。

例如:

Go Node.js PHP Python .NET Rust Java

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

```node
const res = await e.getUsersForRole('data1_admin')
```

```php
$res = $e->getUsersForRole("data1_admin");
```

```python
users = e.get_users_for_role("data1_admin")
```

```net
var res = e.GetUsersForRole("data1_admin");
```

```rust
let users = e.get_users_for_role("data1_admin", None); // No domain
```

```java
List<String> res = e.getUsersForRole("data1_admin");
```

`HasRoleForUser()`

HasRoleForUser确定用户是否拥有某个角色。

例如:

Go Node.js PHP Python .NET Rust Java

```go
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为用户添加一个角色。 如果用户已经拥有该角色（即没有影响），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。 如果用户已经拥有这些角色中的一个（即没有影响）， 则返回false。

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,
```

`DeleteRoleForUser()`

DeleteRoleForUser删除用户的一个角色。 如果用户没有该角色（即没有影响），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（即没有影响），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?; // No domain
```

```
boolean deletedAtLeastOne = e.deleteRolesForUser("alice");
```

`DeleteUser()`

DeleteUser删除一个用户。 如果用户不存在（即没有影响）， 则返回false。

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删除一个权限。 如果权限不存在（即未受影响），则返回false。

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为用户或角色添加一个权限。 如果用户或角色已经拥有该权限（即未受影响），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()`

AddPermissionsForUser为用户或角色添加多个权限。 如果用户或角色已经拥有其中一个权限（即未受影响），则返回false。

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",
```

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

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

`DeletePermissionForUser()`

DeletePermissionForUser删除用户或角色的一个权限。 如果用户或角色没有该权限（即未受影响），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（即未受影响），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");
```

`GetNamedPermissionsForUser()`

GetNamedPermissionsForUser gets permissions for a user or role by named policy.

例如:

```
p, alice, data1, read
p, bob, data2, write
p2, admin, create
g, alice, admin
```

GetNamedPermissionsForUser("p", "alice") will return `[["alice", "data1", "read"]]`. GetNamedPermissionsForUser("p2", "alice") will return `[["admin", "create"]]`.

Go Python

```go
permissions, err := e.GetNamedPermissionsForUser("p", "alice")
```

```python
permissions = e.get_named_permissions_for_user("p", "alice")
```

`HasPermissionForUser()`

HasPermissionForUser确定用户是否拥有一个权限。

例如:

Go Node.js PHP Python .NET Rust Java

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

```node.js
await e.hasPermissionForUser('alice', 'read')
```

```php
$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获取用户拥有的隐式角色。 与GetRolesForUser()相比，此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"]。\ 但是GetImplicitRolesForUser("alice")将获取：["role:admin", "role:user"]。

例如：

Go　Node.js　PHP　Python　.NET　Rust　Java

```
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");
```

`GetNamedImplicitRolesForUser()`

GetNamedImplicitRolesForUser gets implicit roles that a user has by named policy.

例如:

```
g, alice, admin
g, admin, super_admin
g2, alice, user
g2, user, guest
```

GetNamedImplicitRolesForUser("g", "alice") will return `["admin", "super_admin"]`. GetNamedImplicitRolesForUser("g2", "alice") will return `["user", "guest"]`.

Go Python

```
roles, err := e.GetNamedImplicitRolesForUser("g", "alice")
```

```
roles = e.get_named_implicit_roles_for_user("g", "alice")
```

`GetImplicitUsersForRole()`

GetImplicitUsersForRole获取继承该角色的所有用户。 与GetUsersForRole()相比，此函数检索间接用户。

For example:

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

GetUsersForRole("role:user")只能获取：["role:admin"].\ 但是GetImplicitUesrsForRole("role:user")将获取：["role:admin", "alice"].

For example:

Go Node.js Java

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

```
const users = e.getImplicitUsersForRole("role:user");
```

```
List<String> users = e.getImplicitUsersForRole("role:user");
```

`GetImplicitPermissionsForUser()`

GetImplicitPermissionsForUser获取用户或角色的隐式权限。\ 与GetPermissionsForUser()相比，此函数检索继承角色的权限。

例如：

```
p, admin, data1, read
p, alice, data2, read
g, alice, admin
```

GetPermissionsForUser("alice")只能获取：[["alice", "data2", "read"]].\ 但是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); // No domain
```

```
List<List<String>> implicitPermissions =
e.getImplicitPermissionsForUser("alice");
```

`GetNamedImplicitPermissionsForUser()`

GetNamedImplicitPermissionsForUser通过命名策略获取用户或角色的隐式权限。与GetImplicitPermissionsForUser()相比，此函数允许您指定策略名称。

例如：

```
p, admin, data1, read
p2, admin, create
g, alice, admin
```

GetImplicitPermissionsForUser("alice")只能获取：[["admin", "data1", "read"]],其策略默认为"p"

但是你可以通过GetNamedImplicitPermissionsForUser("p2","alice")指定策略为"p2"来获取：[["admin", "create"]]

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。

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

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

`GetImplicitResourcesForUser()`

GetImplicitResourcesForUser返回用户应为真的所有策略。

例如:

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

GetImplicitResourcesForUser("alice")将返回`[[alice data1 read] [alice data2 read] [alice data2 write]]`

Go

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

`GetImplicitUsersForPermission()`

GetImplicitUsersForPermission获取权限的隐式用户。

For example:

```
p, admin, data1, read
p, bob, data1, read
g, alice, admin
```

GetImplicitUsersForPermission("data1", "read")将返回：`["alice", "bob"]`.

注意：只会返回用户，角色（"g"中的第二个参数）将被排除。

Go

```
users, err := e.GetImplicitUsersForPermission("data1", "read")
```

`GetImplicitObjectPatternsForUser()`

GetImplicitObjectPatternsForUser returns all object patterns (with wildcards) that

a user has for a given domain and action.

For example:

```
p, admin, chronicle/123, location/*, read
p, user, chronicle/456, location/789, read
g, alice, admin
g, bob, user
```

GetImplicitObjectPatternsForUser("alice", "chronicle/123", "read") will return `["location/*"]`. GetImplicitObjectPatternsForUser("bob", "chronicle/456", "read") will return `["location/789"]`.

Go Node.js PHP Python .NET Rust Java

```go
patterns, err := e.GetImplicitObjectPatternsForUser("alice",
"chronicle/123", "read")
```

```node
const patterns = await
e.getImplicitObjectPatternsForUser("alice", "chronicle/123",
"read")
```

```php
$patterns = $e->getImplicitObjectPatternsForUser("alice",
"chronicle/123", "read");
```

```python
patterns = e.get_implicit_object_patterns_for_user("alice",
"chronicle/123", "read")
```

```dotnet
var patterns = e.GetImplicitObjectPatternsForUser("alice",
"chronicle/123", "read");
```

```
let patterns = e.get_implicit_object_patterns_for_user("alice",
"chronicle/123", "read").await?;
```

```
List<String> patterns =
e.getImplicitObjectPatternsForUser("alice", "chronicle/123",
"read");
```

`GetAllowedObjectConditions()`

GetAllowedObjectConditions返回用户可以访问的对象条件的字符串数组。

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.")` 将返回 `["price < 25", "category_id = 2"], nil`

注意：

0. 前缀：您可以自定义对象条件的前缀，通常使用"r.obj."作为前缀。 移除前缀后，剩下的部分就是对象的条件。 如果有一个obj策略不满足前缀要求，将返回 `errors.ERR_OBJ_CONDITION`。

1. 如果'objectConditions'数组为空，返回`errors.ERR_EMPTY_CONDITION`。这个错误是因为一些数据适配器的ORM在接收到空条件时默认返回全表数据，这往往与预期行为相反（例如GORM）。如果您使用的适配器不是这样的，您可以选择忽略这

个错误。

Go

```
conditions, err := e.GetAllowedObjectConditions("alice",
"read", "r.obj.")
```

`GetImplicitUsersForResource()`

GetImplicitUsersForResource基于资源返回隐式用户。

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")将返回`[["bob", "data2", "write"], ["alice", "data2", "read"] ["alice", "data2", "write"]], nil`。

GetImplicitUsersForResource("data1")将返回`[["alice", "data1", "read"]], nil`。

Go

```
ImplicitUsers, err := e.GetImplicitUsersForResource("data2")
```

> **备注**
>
> 只会返回用户，角色（"g"中的第二个参数）将被排除。

`GetNamedImplicitUsersForResource()`

GetNamedImplicitUsersForResource return implicit user based on resource with named policy support. This function handles resource role relationships through named policies (e.g., g2, g3, etc.).

For example:

```
p, admin_group, admin_data, *
g, admin, admin_group
g2, app, admin_data
```

GetNamedImplicitUsersForResource("g2", "app") will return users who have access to admin_data through g2 relationship.

Go

```go
ImplicitUsers, err := e.GetNamedImplicitUsersForResource("g2",
"app")
```

RBAC with Domains API

一个更用户友好的RBAC与域的API。 此API是管理API的一个子集。 RBAC用户可以使用此API简化他们的代码。

参考

全局变量`e`代表Enforcer实例。

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()

`GetUsersForRoleInDomain()`函数检索在域内具有角色的用户。

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()

`GetRolesForUserInDomain()`函数检索用户在域内拥有的角色。

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()

`GetPermissionsForUserInDomain()`函数检索用户或角色在域内的权限。

例如:

Go Java

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

```
List<List<String>> res =
e.getPermissionsForUserInDomain("alice", "domain1");
```

AddRoleForUserInDomain()

`AddRoleForUserInDomain()`函数为域内的用户添加角色。 如果用户已经拥有该角色（没有做出改变）， 它将返回`false`。

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()

`DeleteRoleForUserInDomain()`函数删除用户在域内的角色。 如果用户没有该角色（没有做出改变）， 它将返回`false`。

例如:

Go Java

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

```
boolean ok = e.deleteRoleForUserInDomain("alice", "admin",
"domain1");
```

DeleteRolesForUserInDomain()

`DeleteRolesForUserInDomain()`函数删除用户在域内的所有角色。 如果用户没有任何角色（没有做出改变），它将返回`false`。

例如:

Go

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

GetAllUsersByDomain()

`GetAllUsersByDomain()`函数检索与给定域关联的所有用户。 如果在模型中没有定义域，它将返回一个空的字符串数组。

例如:

Go

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

DeleteAllUsersByDomain()

`DeleteAllUsersByDomain()`函数删除与给定域关联的所有用户。 如果在模型中没有定义域，它将返回`false`。

例如:

Go

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

DeleteDomains()

DeleteDomains将删除所有关联的用户和角色。 如果没有提供参数，它将删除所有域。

例如:

Go

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

GetAllDomains()

GetAllDomains将获取所有域。

例如:

Go

```
res, _ := e.GetAllDomains()
```

> **备注**
>
> 如果你正在处理像`name::domain`这样的域，它可能会导致意外的行为。 在Casbin中，`::`是一个保留关键字，就像编程语言中的`for`，`if`，我们永远不应该在域中放置`::`。

GetAllRolesByDomain()

GetAllRolesByDomain将获取与域关联的所有角色。

例如：

Go

```go
res := e.GetAllRolesByDomain("domain1")
```

> **备注**
>
> 此方法不适用于具有继承关系的域，也称为隐式角色。

GetImplicitUsersForResourceByDomain()

GetImplicitUsersForResourceByDomain根据资源和域返回隐式用户。

例如：

```
p, admin, domain1, data1, read
p, admin, domain1, data1, write
p, admin, domain2, data2, read
```

GetImplicitUsersForResourceByDomain("data1", "domain1")将返回`[["alice", "domain1", "data1", "read"],["alice", "domain1", "data1", "write"]], nil`

Go

```
ImplicitUsers, err :=
e.GetImplicitUsersForResourceByDomain("data1", "domain1")
```

> **备注**
>
> 只会返回用户，角色（"g"中的第二个参数）将被排除。

RBAC with Conditions API

一个更加用户友好的带有条件的RBAC API。

参考

AddNamedLinkConditionFunc

`AddNamedLinkConditionFunc`为链接`userName->roleName`添加条件函数fn，当fn返回true时，链接有效，否则无效

Go

```
e.AddNamedLinkConditionFunc("g", "userName", "roleName",
YourLinkConditionFunc)
```

AddNamedDomainLinkConditionFunc

`AddNamedDomainLinkConditionFunc`为链接`userName-> {roleName, domain}`添加条件函数fn，当fn返回true时，链接有效，否则无效

Go

```
e.AddNamedDomainLinkConditionFunc("g", "userName", "roleName",
```

SetNamedLinkConditionFuncParams

`SetNamedLinkConditionFuncParams`为链接`userName->roleName`设置条件函数fn的参数

Go

```go
e.SetNamedLinkConditionFuncParams("g", "userName", "roleName",
"YourConditionFuncParam")
e.SetNamedLinkConditionFuncParams("g", "userName2",
"roleName2", "YourConditionFuncParam_1",
"YourConditionFuncParam_2")
```

SetNamedDomainLinkConditionFuncParams

`SetNamedDomainLinkConditionFuncParams`为链接`userName->{roleName, domain}`设置条件函数fn的参数

Go

```go
e.SetNamedDomainLinkConditionFuncParams("g", "userName",
"roleName", "domainName", "YourConditionFuncParam")
e.SetNamedDomainLinkConditionFuncParams("g", "userName2",
"roleName2", "domainName2", "YourConditionFuncParam_1",
"YourConditionFuncParam_2")
```

RoleManager API

RoleManager

RoleManager提供了一个定义管理角色操作的接口。 向RoleManager添加匹配函数后，可以在角色名称和域中使用通配符。

`AddNamedMatchingFunc()`

`AddNamedMatchingFunc`函数将`MatchingFunc`按Ptype添加到RoleManager。 执行角色匹配时将使用`MatchingFunc`。

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()`

`AddNamedDomainMatchingFunc`函数将`MatchingFunc`按Ptype添加到RoleManager。 `DomainMatchingFunc`与上述的`MatchingFunc`类似。

例如:

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()`

`GetRoleManager`函数获取`g`的当前角色管理器。

例如:

Go Node.js Python

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

```
const rm = await e.getRoleManager();
```

```
rm = e.get_role_manager()
```

`GetNamedRoleManager()`

`GetNamedRoleManager`函数通过命名的Ptype获取角色管理器。

例如:

Go Node.js Python

```
rm := e.GetNamedRoleManager("g2")
```

```
const rm = await e.getNamedRoleManager("g2");
```

```
rm = e.get_named_role_manager("g2")
```

`SetRoleManager()`

`SetRoleManager`函数为`g`设置当前的角色管理器。

例如:

Go Node.js Python

```
e.SetRoleManager(rm)
```

```
e.setRoleManager(rm);
```

```
rm = e.set_role_manager(rm)
```

`SetNamedRoleManager()`

`SetNamedRoleManager`函数通过命名的Ptype设置角色管理器。

例如:

Go Python

```
rm := e.SetNamedRoleManager("g2", rm)
```

```
rm = e.set_role_manager("g2", rm)
```

`Clear()`

`Clear`函数清除所有存储的数据，并将角色管理器重置为其初始状态。

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：name1和角色：name2。 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删除了两个角色之间的继承链接。 角色：name1和角色：name2。 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确定两个角色之间是否存在链接。 角色：name1继承角色：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获取用户继承的角色。 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将所有角色打印到日志。

例如:

Go Node.js Python

```
rm.PrintRoles()
```

```
await rm.printRoles();
```

```
rm.print_roles()
```

`SetLogger()`

SetLogger设置角色管理器的记录器。

例如:

Go

```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

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

高级用法

Multi-threading

在多线程环境中使用Casbin

Benchmarks

Casbin中策略执行的开销

Performance Optimization

Casbin性能优化

Authorization of Kubernetes

基于Casbin的Kubernetes（k8s）RBAC和ABAC授权中间件

📄 Admission Webhook for K8s

基于Casbin的Kubernetes（K8s）RBAC和ABAC授权中间件

📄 Authorization of Service Mesh through Envoy

通过Envoy对服务网格进行授权

Multi-threading

在多线程环境中使用Casbin时，您可以使用Casbin执行器的同步包装器：https://github.com/casbin/casbin/blob/master/enforcer_synced.go (GoLang) 和 https://github.com/casbin/casbin-cpp/blob/master/casbin/enforcer_synced.cpp (C++)。

此外，它还支持"AutoLoad"功能，如果数据库中的策略规则发生任何更改，Casbin执行器可以自动从数据库加载最新的策略规则。 要启动策略的定期自动加载，请调用"StartAutoLoadPolicy()"函数。 同样，要停止这种自动加载，请调用"StopAutoLoadPolicy()"函数。

Benchmarks

Go Python C++ Lua (JIT)

策略执行的开销已在model_b_test.go中进行了基准测试。 测试环境配置如下：

```
Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz, 2601 Mhz, 4 Core(s), 8 Logical Processor(s)
```

以下是通过运行`go test -bench=.`获得的基准测试结果 -benchmem（op = 一个`Enforce()`调用，ms = 毫秒，KB = 千字节）：

测试用例	规则大小	时间开销（ms/op）	内存开销（KB）
ACL	2条规则（2个用户）	0.015493	5.649
RBAC	5条规则（2个用户，1个角色）	0.021738	7.522
RBAC（小）	1100条规则（1000个用户，100个角色）	0.164309	80.620
RBAC（中）	11000条规则（10000个用户，1000个角色）	2.258262	765.152
RBAC（大）	110000条规则（100000个用户，10000个角色）	23.916776	7,606
带资源角色的RBAC	6条规则（2个用户，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条规则（0个用户）	0.007510	2.328
RESTful	5条规则（3个用户）	0.045398	91.774
否决覆盖	6条规则（2个用户，1个角色）	0.023281	8.370
优先级	9条规则（2个用户，2个角色）	0.016389	5.313

Pycasbin中策略执行的开销已在`tests/benchmarks`目录中进行了基准测试。 测试环境配置如下：

```
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
```

以下是执行`casbin_benchmark`（op = 一个`enforce()`调用，ms = 毫秒）获得的基准测试结果：

测试用例	规则大小	时间开销 (ms/op)
ACL	2条规则（2个用户）	0.067691
RBAC	5条规则（2个用户，1个角色）	0.080045

测试用例	规则大小	时间开销 (ms/op)
RBAC（小）	1100条规则（1000个用户，100角色）	0.853590
RBAC（中）	11000条规则（10000个用户，1000个角色）	6.986668
RBAC（大）	110000条规则（100000个用户，10000个角色）	77.922851
带有资源角色的RBAC	6条规则（2个用户，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条规则（0个用户）	0.053213
RESTful	5条规则（3个用户）	NA
否决覆盖	6条规则（2个用户，1个角色）	NA
优先级	9条规则（2个用户，2个角色）	0.084684

Casbin CPP中策略执行的开销已在`tests/benchmarks`目录中使用Google的基准测试工具进行了基准测试。 测试环境配置如下：

```
Intel(R) Core(TM) i5-6300HQ CPU @ 2.30GHz, 4 cores, 4 threads
```

以下是从执行`Release`配置中构建的`casbin_benchmark`目标获得的基准测试结果（op = 一个`enforce()`调用，ms = 毫秒）：

测试用例	规则大小	时间开销（ms/op）
ACL	2条规则（2个用户）	0.0195
RBAC	5条规则（2个用户，1个角色）	0.0288
RBAC（小）	1100条规则（1000个用户，100个角色）	0.300
RBAC（中）	11000条规则（10000个用户，1000个角色）	2.113
RBAC（大）	110000条规则（100000个用户，10000个角色）	21.450
带资源角色的RBAC	6条规则（2个用户，2个角色）	0.03
带域/租户的RBAC	6条规则（2个用户，1个角色，2个域）	0.041
ABAC	0条规则（0个用户）	NA
RESTful	5条规则（3个用户）	NA
拒绝-覆盖	6条规则（2个用户，1个角色）	0.0246
优先级	9条规则（2个用户，2个角色）	0.035

Lua Casbin中策略执行的开销已在bench.lua中进行了基准测试。 测试环境配置如下：

```
AMD Ryzen(TM) 5 4600H CPU @ 3.0GHz, 6 Cores, 12 Threads
```

以下是通过运行`luajit bench.lua`获得的基准测试结果（op = 一个`enforce()`调用，ms = 毫秒）：

测试用例	规则大小	时间开销（毫秒/操作）
ACL	2条规则（2个用户）	0.0533
RBAC	5条规则（2个用户，1个角色）	0.0972
RBAC（小）	1100条规则（1000个用户，100角色）	0.8598
RBAC（中）	11000条规则（10000个用户，1000个角色）	8.6848
RBAC（大）	110000条规则（100000个用户，10000个角色）	90.3217
带有资源角色的RBAC	6条规则（2个用户，2个角色）	0.1124
带有域/租户的RBAC	6条规则（2个用户，1个角色，2个域）	0.1978
ABAC	0条规则（0个用户）	0.0305
RESTful	5条规则（3个用户）	0.1085
拒绝覆盖	6条规则（2个用户，1个角色）	0.1934
优先级	9条规则（2个用户，2个角色）	0.1437

基准监控

在下面的嵌入式网页中，您可以看到Casbin对每个提交的性能变化。

您也可以直接在以下地址浏览：https://v1.casbin.org/casbin/benchmark-monitoring

Last Update:
Repository:

Download data as JSON

Powered by github-action-benchmark

Performance Optimization

当在拥有数百万用户或权限的生产环境中应用时，您可能会遇到Casbin执行性能下降的问题。 通常有两个原因：

高流量

每秒进入的请求数量过大，例如，单个Casbin实例每秒10000个请求。 在这种情况下，单个Casbin实例通常无法处理所有请求。 有两种可能的解决方案：

1. 使用多线程启用多个Casbin实例，这样您可以充分利用机器中的所有核心。 更多详情，请参见：多线程。
2. 将Casbin实例部署到集群（多台机器）并使用Watcher确保所有Casbin实例的一致性。 更多详情，请参见：观察者。

> **备注**
>
> 您可以同时使用上述两种方法，例如，将Casbin部署到一个10台机器的集群，每台机器有5个线程同时服务于Casbin执行请求。

策略规则数量高

在云环境或多租户环境中，可能需要数百万条策略规则。 每次执行调用甚至在初始时间加载策略规则都可能非常慢。 这种情况通常可以通过几种方式缓解：

1. 检查您的Casbin模型或策略是否设计得当。 一个编写良好的模型和策略可以将每个用户/租户的重复逻辑抽象出来，并将规则数量减少到非常小的级别（\< 100）。 例如，您可以在所有租户之间共享一些默认规则，并允许用户稍后自定义他们的规则。 定制的规则可以覆盖默认规则。 如果您有任何进一步的问题，请在Casbin仓库上开

一个GitHub问题。

2. 进行分片，让Casbin执行器只加载一小部分策略规则。 例如，执行器_0可以服务于租户_0到租户_99，而执行器_1可以服务于租户_100到租户_199。 要加载所有策略规则的子集，请参见：策略子集加载。

3. 将权限授予RBAC角色，而不是直接授予用户。 Casbin的RBAC是通过角色继承树（作为缓存）实现的。 所以，对于像Alice这样的用户，Casbin只需要O(1)的时间就可以查询RBAC树的角色-用户关系并执行。 如果你的g规则不经常改变，那么RBAC树就不需要不断更新。 查看此讨论的详细信息：https://github.com/casbin/casbin/issues/681#issuecomment-763801583

> **备注**
>
> 你可以同时尝试上述所有方法。

Authorization of Kubernetes

K8s-authz是一个基于Casbin的Kubernetes（k8s）授权中间件，利用RBAC（基于角色的访问控制）和ABAC（基于属性的访问控制）进行策略执行。 此中间件与K8s验证准入webhook集成，以验证Casbin为每个向K8s资源发出的请求定义的策略。 自定义准入控制器使用`ValidatingAdmissionWebhook`在Kubernetes中注册，以对API服务器转发的请求对象进行验证，并提供一个响应，指示是否应允许或拒绝请求。

为确定何时将传入请求发送到准入控制器，已实施了一个验证webhook。 此webhook代理对任何类型的K8s资源或子资源的请求，并执行策略验证。 只有在Casbin执行器授权的情况下，用户才被允许对这些资源执行操作。 执行器检查用户在策略中定义的角色。 K8s集群是此中间件的部署目标。

要求

在继续之前，请确保您有以下内容：

- 正在运行的Kubernetes集群。 您可以使用Docker设置本地集群，或在您的服务器上设置完整的Kubernetes生态系统。 有关在Windows上设置本地Kubernetes集群的详细说明，请参阅此指南，有关在Linux上设置集群的说明，请参阅此指南。
- Kubectl CLI。 有关在Windows上安装Kubectl的说明可以在这里找到，对于Linux，可以在这里找到。
- OpenSSL

使用

按照以下步骤使用K8s-authz：

1. 使用OpenSSL为每个用户生成证书和密钥。 运行下面的脚本：

```
./gen_cert.sh
```

2. 通过运行以下命令手动从Dockerfile构建Docker镜像。 记得在命令和部署文件中相应地更改构建版本。

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

3. 在model.conf和policy.csv文件中定义Casbin策略。 有关这些策略如何工作的更多信息，请参阅文档。

4. 在部署之前，您可以根据您的特定要求，在main.go文件以及验证webhook配置文件中修改端口。

5. 通过运行以下命令在Kubernetes集群上部署验证控制器和webhook：

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

6. 对于生产服务器，建议创建一个Kubernetes `secret`来保护证书：

```
kubectl create secret generic casbin -n default \
  --from-file=key.pem=certs/casbin-key.pem \
  --from-file=cert.pem=certs/casbin-crt.pem
```

7. 完成上述步骤后，您需要更新main.go和manifests中的证书目录，以及创建的`secret`的目录。

现在，服务器应该已经启动并运行，准备验证对K8s资源的请求并相应地执行策略。

Admission Webhook for K8s

1. Casbin K8s-Gatekeeper的概述和文档

Casbin K8s-GateKeeper是一个将Casbin集成为访问控制工具的Kubernetes准入webhook。 通过使用Casbin K8s-GateKeeper，您可以建立灵活的规则来授权或拦截对K8s资源的任何操作，无需编写任何代码，只需几行Casbin模型和策略的声明性配置，这些配置是Casbin ACL（访问控制列表）语言的一部分。

Casbin K8s-GateKeeper由Casbin社区开发和维护。 该项目的仓库可在此处获取：https://github.com/casbin/k8s-gatekeeper

0.1 一个简单的例子

例如，您不需要编写任何代码，只需使用以下几行配置就可以实现这个功能：“禁止在任何部署中使用带有某些指定标签的图像”：

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" && \
contain(split(accessWithWildcard(${OBJECT}.Spec.Template.Spec.Containers , "*",
"Image"),":",1) , p.obj)
```

And Policy:

```
p, "1.14.1",deny
```

这些都是普通的Casbin ACL语言。 假设你已经阅读了关于它们的章节，理解起来会非常容易。

Casbin K8s-Gatekeeper有以下优点：

- 易于使用。 编写几行ACL远比编写大量代码要好。

- 它允许热更新配置。 您不需要关闭整个插件来修改配置。
- 它是灵活的。 可以在任何K8s资源上制定任意规则，可以使用`kubectl gatekeeper`进行探索。
- 它简化了K8s准入webhook的实现，这是非常复杂的。 你不需要知道K8s准入webhook是什么，也不需要知道如何为它编写代码。 你需要做的只是知道你想要约束的资源，然后编写Casbin ACL。 大家都知道K8s是复杂的，但是通过使用Casbin K8s-Gatekeeper，你的时间可以得到节省。
- 它由Casbin社区维护。 如果你对这个插件有任何疑问，或者在尝试使用时遇到任何问题，随时联系我们。

1.1 Casbin K8s-Gatekeeper是如何工作的?

K8s-Gatekeeper是一个K8s的准入webhook，它使用Casbin应用任意用户定义的访问控制规则，以帮助防止管理员不希望的任何K8s操作。

Casbin是一个强大且高效的开源访问控制库。 它提供了基于各种访问控制模型进行授权的支持。 有关Casbin的更多详细信息，请参见概述。

K8s中的准入webhooks是接收'准入请求'并对其进行处理的HTTP回调。 特别地，K8s-Gatekeeper是一种特殊类型的准入webhook：'ValidatingAdmissionWebhook'，它可以决定是否接受或拒绝这个准入请求。 至于准入请求，它们是描述在K8s指定资源上的操作的HTTP请求（例如，创建/删除部署）。 关于准入webhooks的更多信息，请参阅K8s官方文档。

1.2 一个说明其工作方式的例子

例如，当有人想要创建一个包含运行nginx的pod的部署（使用kubectl或K8s客户端），K8s将生成一个准入请求，如果将其转换为YAML格式，可能会是这样的：

```yaml
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
        ports:
        - containerPort: 80
```

这个请求将经过图片中显示的所有中间件的处理过程，包括我们的K8s-Gatekeeper。 K8s-Gatekeeper可以检测到存储在K8s的etcd中的所有Casbin执行器，这些执行器是由用户创建和维护的（通过`kubectl`或我们提供的Go客户端）。 每个

执行器都包含一个Casbin模型和一个Casbin策略。 准入请求将由每个执行器逐一处理，只有通过所有执行器的请求才能被这个K8s-Gatekeeper接受。

（如果你不明白什么是Casbin执行器、模型或策略，请参阅这个文档：入门）。

例如，出于某种原因，管理员想要禁止'image:nginx:1.14.1'的出现，而允许'nginx:1.3.1'。 可以创建包含以下规则和策略的执行器（我们将在后面的章节中解释如何创建执行器，这些模型和策略是什么，以及如何编写它们）。

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
```

通过创建包含上述模型和策略的执行器，之前的准入请求将被这个执行器拒绝，这意味着K8s不会创建这个部署。

2 安装K8s-gatekeeper

有三种可用的安装K8s-gatekeeper的方法：外部webhook，内部webhook和Helm。

> **备注**
>
> 注意：这些方法仅供用户尝试K8s-gatekeeper使用，并不安全。 如果你希望在生产环境中使用它，请确保你阅读了第5章。 高级设置并在安装前做出任何必要的修改。

2.1 内部webhook

2.1.1 第一步：构建镜像

对于内部webhook方法，webhook本身将作为Kubernetes内的一个服务实现。 要创建必要的服务和部署，你需要构建一个K8s-gatekeeper的镜像。 你可以通过运行以下命令来构建你自己的镜像：

```
docker build --target webhook -t k8s-gatekeeper .
```

这个命令将创建一个名为'k8s-gatekeeper:latest'的本地镜像。

> **备注**
>
> 注意：如果你正在使用minikube，请在运行'docker build'之前执行`eval $(minikube -p minikube docker-env)`。

2.1.2 第二步：为K8s-gatekeeper设置服务和部署

运行以下命令：

```
kubectl apply -f config/rbac.yaml
kubectl apply -f config/webhook_deployment.yaml
kubectl apply -f config/webhook_internal.yaml
```

这将开始运行K8s-gatekeeper，你可以通过运行`kubectl get pods`来确认这一点。

2.1.3 第三步：为K8s-gatekeeper安装CRD资源

运行以下命令：

```
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
```

2.2 外部 webhook

对于外部 webhook 方法，K8s-gatekeeper 将在 Kubernetes 外部运行，而 Kubernetes 将像访问常规网站一样访问 K8s-gatekeeper。 Kubernetes 有一个强制要求，即 admission webhook 必须是 HTTPS。 为了尝试 K8s-gatekeeper，我们提供了一套证书和私钥（尽管这不安全）。 如果你更喜欢使用自己的证书，请参考第5章。 高级设置 以获取调整证书和私钥的说明。

我们提供的证书是为 'webhook.domain.local' 发布的。 因此，修改主机（例如，/etc/hosts）并将 'webhook.domain.local' 指向 K8s-gatekeeper 正在运行的 IP 地址。

然后执行以下命令：

```
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 通过 Helm 安装 K8s-gatekeeper

2.3.1 第一步：构建镜像

请参考第2.1.1章。

2.3.2 Helm 安装

运行命令 `helm install k8sgatekeeper ./k8sgatekeeper`。

3. 尝试 K8s-gatekeeper

3.1 创建 Casbin 模型和策略

你有两种方法来创建模型和策略：通过 kubectl 或通过我们提供的 go-client。

3.1.1 通过 kubectl 创建/更新 Casbin 模型和策略

在 K8s-gatekeeper 中，Casbin 模型存储在一个名为 'CasbinModel' 的 CRD 资源中。 其定义位于 `config/auth.casbin.org_casbinmodels.yaml` 中。

`example/allowed_repo/model.yaml` 中有示例。 注意以下字段：

- metadata.name：模型的名称。 此名称必须与与此模型相关的 CasbinPolicy 对象的名称相同，以便 K8s-gatekeeper 可以将它们配对并创建一个执行器。
- spec.enable：如果此字段设置为“false”，则将忽略此模型（以及与此模型相关的 CasbinPolicy 对象）。
- spec.modelText：包含 Casbin 模型的模型文本的字符串。

Casbin 策略存储在另一个名为 'CasbinPolicy' 的 CRD 资源中，其定义可以在 `config/auth.casbin.org_casbinpolicies.yaml` 中找到。

`example/allowed_repo/policy.yaml` 中有示例。 请注意以下字段：

- metadata.name：策略的名称。 此名称必须与与此策略相关的CasbinModel对象的名称相同，以便K8s-gatekeeper 可以将它们配对并创建一个执行器。

- spec.policyItem：包含Casbin模型的策略文本的字符串。

创建自己的CasbinModel和CasbinPolicy文件后，使用以下命令应用它们：

```
kubectl apply -f <filename>
```

一旦创建了一对CasbinModel和CasbinPolicy，K8s-gatekeeper将能够在5秒内检测到它。

3.1.2 通过我们提供的go-client创建/更新Casbin模型和策略

我们理解可能会有一些情况，不方便直接在K8s集群的节点上使用shell执行命令，比如当你正在为你的公司构建一个自动化的云平台。 因此，我们开发了一个go-client来创建和维护CasbinModel和CasbinPolicy。

go-client库位于`pkg/client`。

在`client.go`中，我们提供了一个创建客户端的函数。

```go
func NewK8sGateKeeperClient(externalClient bool) (*K8sGateKeeperClient, error)
```

`externalClient`参数决定了K8s-gatekeeper是在K8s集群内部运行还是在外部运行。

在`model.go`中，我们提供了各种函数来创建、删除和修改CasbinModel。 你可以在`model_test.go`中找到如何使用这些接口。

在`policy.go`中，我们提供了各种函数来创建、删除和修改CasbiPolicy。 你可以在`policy_test.go`中找到如何使用这些接口。

3.1.2 尝试K8s-gatekeeper是否工作

假设你已经在`example/allowed_repo`中创建了精确的模型和策略。 现在，尝试以下命令：

```
kubectl apply -f example/allowed_repo/testcase/reject_1.yaml
```

你应该发现K8s会拒绝这个请求，并提到webhook是拒绝这个请求的原因。 然而，当你尝试应用`example/allowed_repo/testcase/approve_2.yaml`时，它将被接受。

4. 如何使用K8s-gatekeeper编写模型和策略

首先，确保你熟悉Casbin模型和策略的基本语法。 如果你不熟悉，请先阅读入门部分。 在本章中，我们假设你已经理解了Casbin模型和策略是什么。

4.1 模型的请求定义

当K8s-gatekeeper正在授权一个请求时，输入总是一个对象：Admission Request的Go对象。 这意味着执行器总是这样使用的：

```
ok, err := enforcer.Enforce(admission)
```

其中`admission`是一个由K8s的官方go api`"k8s.io/api/admission/v1"`定义的`AdmissionReview`对象。 您可以在此存储库中找到此结构的定义：https://github.com/kubernetes/api/blob/master/admission/v1/types.go。 有关更多信息，您也可以参考 https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-request-and-response。

因此，对于K8s-gatekeeper使用的任何模型，`request_definition` 的定义应始终如此：

```
    [request_definition]
    r =  obj
```

'obj' 的名称并非必须的，只要名称与 `[matchers]` 部分中使用的名称一致即可。

4.2 模型的匹配器

您应该使用 Casbin 的 ABAC 功能来编写您的规则。 然而，集成在 Casbin 中的表达式求值器不支持在映射或数组（切片）中进行索引，也不支持数组的扩展。 因此，K8s-gatekeeper 提供了各种 'Casbin 函数' 作为扩展来实现这些功能。 如果您发现这些扩展仍无法满足您的需求，随时可以提出问题，或创建一个拉取请求。

如果您不熟悉 Casbin 函数，可以参考 Function 以获取更多信息。

以下是扩展函数：

4.2.1 扩展函数

4.2.1.1 access

Access 用于解决 Casbin 不支持在映射或数组中进行索引的问题。 示例 `example/allowed_repo/model.yaml` 展示了此函数的使用：

```
[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
```

在这个匹配器中，`access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image")`

等同于 `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Image`，其中 `r.obj.Request.Object.Object.Spec.Template.Spec.Containers` 是一个切片。

Access 还可以调用没有参数并返回单个值的简单函数。 示例 `example/container_resource_limit/model.yaml` 展示了这一点：

```
[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)
```

在这个匹配器中，`access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Resources","Limits","cpu","Value")` 等同于 `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Resources.Limits["cpu"].Value()`，其中 `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Resources.Limits` 是一个映射，`Value()` 是一个没有参数并返回单个值的简单函数。

4.2.1.2 accessWithWildcard

有时，您可能有这样的需求：数组中的所有元素都必须有一个前缀 "aaa"。 然而，Casbin 不支持 `for` 循环。 通过 `accessWithWildcard` 和 "map/slice expansion" 功能，您可以轻松实现这样的需求。

例如，假设 `a.b.c` 是一个数组 `[aaa,bbb,ccc,ddd,eee]`，那么 `accessWithWildcard(a,"b","c","*")` 的结果将是一个切片 `[aaa,bbb,ccc,ddd,eee]`。 通过使用通配符 `*`，切片被扩展。

同样，通配符可以使用多次。 例如，`accessWithWildcard(a,"b","c","*","*")` 的结果将是 `[a.b.c[0][0], a.b.c[0][1], ..., a.b.c[1][0], a.b.c[1][1], ...]`。

4.2.1.3 支持变长参数的函数

在 Casbin 的表达式求值器中，当一个参数是数组时，它将被自动扩展为一个变长参数。 利用这个特性来支持数组/切片/映射的扩展，我们还集成了几个接受数组/切片作为参数的函数：

- contain(): 接受多个参数并返回任何参数（除最后一个参数外）是否等于最后一个参数。
- split(a,b,c...,sep,index): 返回一个包含 `[splits(a,sep)[index], splits(b,sep)[index], splits(a,sep)[index], ...]` 的切片。
- len(): 返回可变长度参数的长度。
- matchRegex(a,b,c...,regex): 返回所有给定参数（`a`，`b`，`c`，...） 是否匹配给定的正则表达式。

这是 `example/disallowed_tag/model.yaml` 中的一个例子：

```
    [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)
```

假设 `accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , "*", "Image")` 返回 `["a:b", "c:d", "e:f", "g:h"]`，因为 splits 支持可变长度参数并对每个元素执行 splits 操作，所以将选择并返回索引 1 处的元素。 因此，`split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , "*", "Image"),":",1)` 返回 `["b","d","f","h"]`。 而 `contain(split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , "*", "Image"),":",1) , p.obj)` 返回 `p.obj` 是否包含在 `["b","d","f","h"]` 中。

4.2.1.2 类型转换函数

- ParseFloat(): 将整数解析为浮点数（这是必要的，因为在比较中使用的任何数字都必须转换为浮点数）。
- ToString(): 将对象转换为字符串。 此对象必须具有字符串的基本类型（例如，当有 `type XXX string` 的声明时，类型为 `XXX` 的对象）。
- IsNil(): 返回参数是否为 nil。

5. 高级设置

5.1 关于证书

在 Kubernetes（k8s）中，webhook 必须使用 HTTPS 是强制性的。 有两种方法可以实现这一点：

- 使用自签名证书（此存储库中的示例使用此方法）
- 使用普通证书

5.1.1 自签名证书

使用自签名证书意味着发行证书的证书颁发机构（CA）不是众所周知的 CA。 因此，你必须让 k8s 知道这个 CA。

目前，此存储库中的示例使用自制的 CA，其私钥和证书分别存储在 `config/certificate/ca.crt` 和 `config/certificate/ca.key` 中。 webhook 的证书是 `config/certificate/server.crt`，由自制的 CA 颁发。 此证书的域名是 "webhook.domain.local"（用于外部 webhook）和 "casbin-webhook-svc.default.svc"（用于内部 webhook）。

通过 webhook 配置文件将 CA 的信息传递给 k8s。 `config/webhook_external.yaml` 和 `config/webhook_internal.yaml` 都有一个名为 "CABundle" 的字段，其中包含 CA 证书的 base64 编码字符串。

如果你需要更改证书/域名（例如，如果你想将此 webhook 放入 k8s 的另一个命名空间，同时使用内部 webhook，或者

如果你想在使用外部 webhook 时更改域名），应遵循以下程序：

1. 生成新的CA：
   - 为伪造的CA生成私钥：

     ```
     openssl genrsa -des3 -out ca.key 2048
     ```

   - 移除私钥的密码保护：

     ```
     openssl rsa -in ca.key -out ca.key
     ```

2. 为webhook服务器生成私钥：

   ```
   openssl genrsa -des3 -out server.key 2048
   openssl rsa -in server.key -out server.key
   ```

3. 使用自生成的CA签署webhook的证书：
   - 临时使用您系统的openssl配置文件的副本。 您可以通过运行`openssl version -a`找出配置文件的位置，通常被称为`openssl.cnf`。
   - 在配置文件中：
     - 找到`[req]`段落并添加以下行：`req_extensions = v3_req`
     - 找到`[v3_req]`段落并添加以下行：`subjectAltName = @alt_names`
     - 将以下行追加到文件中：

       ```
       [alt_names]
       DNS.2=<The domain you want>
       ```

       注意：如果您决定修改服务名称，请用您自己服务的真实服务名称替换'casbin-webhook-svc.default.svc'。
   - 使用修改后的配置文件生成证书请求文件：

     ```
     openssl req -new -nodes -keyout server.key -out server.csr -config openssl.cnf
     ```

   - 使用自制的CA响应请求并签署证书：

     ```
     openssl x509 -req -days 3650 -in server.csr -out server.crt -CA ca.crt -CAkey
     ```

4. 替换'CABundle'字段：`config/webhook_external.yaml`和`config/webhook_internal.yaml`都有一个名为"CABundle"的字段，其中包含CA的证书的base64编码字符串。 用新证书更新此字段。

5. 如果您正在使用helm，需要对helm图表应用类似的更改。

5.1.2 合法证书

如果您使用合法证书，您不需要经历所有这些程序。 移除`config/webhook_external.yaml`和`config/webhook_internal.yaml`中的"CABundle"字段，并将这些文件中的域更改为您拥有的域。

Authorization of Service Mesh through Envoy

Envoy-authz 是一个Envoy的中间件，通过casbin执行外部RBAC和ABAC授权。 此中间件通过gRPC服务器使用Envoy的外部授权API。 此代理可以部署在任何类型的基于Envoy的服务网格上，如Istio。

需求

- Envoy 1.17+
- Istio或任何其他类型的服务网格
- grpc依赖

依赖项使用`go.mod`进行管理。

中间件的工作方式

- 客户端发出HTTP请求。
- Envoy代理将请求发送到gRPC服务器。
- gRPC服务器根据casbin策略授权请求。
- 如果授权，请求将被转发；否则，请求将被拒绝。

gRPC服务器基于Envoy中的external_auth.proto协议缓冲区。

```
// A generic interface for performing authorization checks on
```

从上述proto文件中，我们需要在授权服务器中使用`Check()`服务。

用法

- 按照此指南在配置文件中定义Casbin策略。

您可以使用在线casbin-editor验证/测试您的策略。

- 通过运行以下命令启动认证服务器：

```
go build .
./authz
```

- 加载Envoy配置：

```
envoy -c authz.yaml -l info
```

一旦Envoy启动，它将拦截请求进行授权处理。

与Istio集成

为了使这个中间件工作，你需要发送包含JWT令牌或头部的用户名的自定义头部。 您可以参考官方Istio文档以获取更多关于修改`请求头`的信息。

管理

Admin Portal

Casbin的管理员门户

Casbin Service

将Casbin用作服务

Command-line Tools

Command-line Tools

Log & Error Handling

在Casbin中的日志记录和错误处理

📄 Frontend Usage

Casbin.js是一个Casbin插件，它可以方便你在前端应用程序中管理访问控制

Admin Portal

我们提供一个名为Casdoor的基于网络的门户，用于模型管理和策略管理：

还有一些使用Casbin作为授权引擎的第三方管理员门户项目。 您可以基于这些项目开始构建自己的Casbin服务。

Go Java Node.js Python PHP

项目	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	基于Gin + Casbin + GORM的go-admin
gin-vue-admin	flipped-aurora	Vue + Element	Gin	基于Gin + GORM + Vue

项目	作者	前端	后端	描述
		UI		
gin-admin	@LyricTian	React + Ant Design	Gin	基于Gin + GORM + Casbin + Ant Design React的RBAC脚手架
go-admin	@hequan2017	无	Gin	基于Gin + GORM + JWT + RBAC (Casbin)的Go 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)的管理员门户
echo-admin (前端, 后端)	@RealLiuSha	Vue 2.x + Element UI	Echo	基于Echo + Gorm + Casbin + Uber-FX的管理员门户
Spec-Center	@atul-wankhade	无	Mux	基于Casbin + MongoDB的Golang RESTful平台

项目	作者	前端	后端	描述
spring-boot-web	@BazookaW	无	SpringBoot	基于SpringBoot 2.0 + MyBatisPlus + Casbin的管理员门户

项目	作者	前端	后端	描述
node-mysql-rest-api	@JoemaNequinto	无	Express	一个用于构建RESTful APIs微服务的样板应用，使用Node.js、Express、Sequelize、JWT和Casbin。
Casbin-Role-Mgt-Dashboard-RBAC	@alikhan866	React + Material UI	Express	初学者友好的RBAC管理，与Enforcer集成，可以即时检查执行结果

项目	作者	前端	后端	描述
fastapi-best-architecture	@WuClan	Vue + Arco-design	FastAPI	基于FastAPI, SQLAlchemy, JWT和RBAC的管理员门户
fastapi-mysql-generator	@CoderCharm	无	FastAPI	FastAPI + MySQL + JWT + Casbin

项目	作者	前端	后端	描述
FastAPI-MySQL-Tortoise-Casbin	@xingxingzaixian	无	FastAPI	FastAPI + MySQL + Tortoise + Casbin
openstack-policy-editor	Casbin	Bootstrap	Django	Casbin的Web UI

项目	作者	前端	后端	描述
Tadmin	@leeqvip	AmazeUI	ThinkPHP	基于ThinkPHP的非侵入式后端框架
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权限管理系统

项目	作者	前端	后端	描述
larke-admin (前端, 后端)	@deatil	Vue 2 + Element UI	Laravel 8	基于Laravel 8, JWT和RBAC的管理员门户
hyperf-vuetify-admin	@TragicMale	Vue + Vuetify 2.x	Hyperf	基于Hyperf, Vuetify和Casbin的管理员门户

Casbin Service

如何将Casbin用作服务?

名称	描述
Casbin服务器	基于gRPC的官方“Casbin作为服务”解决方案。 提供了管理API和RBAC API。
中间件-acl	基于Casbin的RESTful访问控制中间件。
auth-server	用于校对服务的Auth服务器。

Command-line Tools

Casbin CLIs are command-line tools that provide a command-line interface for Casbin, enabling you to use all of Casbin APIs in the shell. This documentation covers the usage of Casbin CLI for various languages including Rust, Java, Go, Python, .NET, and NodeJs.

Installation

Go (casbin-go-cli)

1. Clone project from repository

```
git clone https://github.com/casbin/casbin-go-cli.git
```

2. Build project

```
cd casbin-go-cli
go build -o casbin
```

Rust (casbin-rust-cli)

From crates.io

```
cargo install casbin-rust-cli
```

From source

1. Clone project from repository

```
git clone https://github.com/casbin-rs/casbin-rust-cli.git
```

2. Build project

```
cd casbin-rust-cli
cargo build --release
```

Java (casbin-java-cli)

1. Clone project from repository

```
git clone https://github.com/jcasbin/casbin-java-cli.git
```

2. Build project, the jar package will be generated in the target directory

```
cd casbin-java-cli
mvn clean install
```

Python (casbin-python-cli)

1. Clone project from repository

```
git clone https://github.com/casbin/casbin-python-cli.git
```

2. Install dependencies

```
cd casbin-python-cli
pip install -r requirements.txt
```

3. Run CLI

```
python -m casbin_cli.client [command] [options] [args]
```

.NET (casbin-dotnet-cli)

1. Clone project from repository

```
git clone https://github.com/casbin-net/casbin-dotnet-cli.git
```

2. Build project

```
cd casbin-dotnet-cli
dotnet build
```

Usage

Options

options	description	must	remark
`-m, --model`	The path of the model file or model text	y	Please wrap it with `""` and separate each line with `\\|`
`-p, --policy`	The path of the policy file or policy text	y	Please wrap it with `""` and separate each line with `\\|`
`-e, --enforce`	Check permissions	n	Please wrap it with `""`
`-ex, --enforceEx`	Check permissions and get which policy it is	n	Please wrap it with `""`
`-AF, --addFuntion`	Add custom funtion (casbin-java-cli only)	n	Please wrap it with `""` and separate each line with `\\|`
`-ap, --addPolicy`	Add a policy rule to the policy file (casbin-java-cli only)	n	Please wrap it with `""`

options	description	must	remark
`-rp, --removePolicy`	Remove a policy rule from the policy file (casbin-java-cli only)	n	Please wrap it with `""`

Get started

- Check whether Alice has read permission on data1

```
./casbin enforce -m "examples/rbac_model.conf" -p "examples/rbac_policy.csv" "alice" "data1" "read"
```

> `{"allow":true,"explain":null}`

```
./casbin enforce -m "[request_definition]\nr = sub, obj, act\n[policy_definition]\np = sub, obj, act\n[role_definition]\ng = _, _\n[policy_effect]\ne = some(where (p.eft == allow))\n[matchers]\nm = g(r.sub, p.sub) && r.obj == p.obj && r.act == p.act" -p "p, alice, data1, read\np, bob, data2, write\np, data2_admin, data2, read\np, data2_admin, data2, write\ng, alice, data2_admin" "alice" "data1" "read"
```

> `{"allow":true,"explain":null}`

- Check whether Alice has write permission for data2. If so, display the effective policy.

```
./casbin enforceEx -m "examples/rbac_model.conf" -p
"examples/rbac_policy.csv" "alice" "data2" "write"
```

> `{"allow":true,"explain":["data2_admin","data2","write"]}`

- Add a policy to the policy file (casbin-java-cli only)

```
./casbin addPolicy -m "examples/rbac_model.conf" -p
"examples/rbac_policy.csv" "alice" "data2" "write"
```

> `{"allow":true,"explain":null}`

- Delete a policy from the policy file (casbin-java-cli only)

```
./casbin removePolicy -m "examples/rbac_model.conf" -p
"examples/rbac_policy.csv" "alice" "data2" "write"
```

> `{"allow":true,"explain":null}`

Log & Error Handling

日志记录

Casbin默认使用内置的`log`将日志打印到控制台，如：

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

默认情况下不启用日志记录。 您可以通过`Enforcer.EnableLog()`或`NewEnforcer()`的最后一个参数来切换它。

> **备注**
>
> For Golang: We already support logging the model, enforce request, role, and policy in Golang. 您可以为Casbin定义自己的日志。

> **备注**
>
> For Python: PyCasbin leverages the default Python logging mechanism. PyCasbin makes a call to `logging.getLogger()` to set the logger. No special logging configuration is needed other than initializing the logger in the parent application. 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 logs in PyCasbin, you can specify the logging configuration through the parameter `logging_config`. If no configuration is specified, 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. 对于其他Python用户，您应参考Python日志文档来配置记录器。

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.

```
// Set a default logger as enforcer e1's logger.
// This operation can also be seen as changing the logger of e1
at runtime.
e1.SetLogger(&Log.DefaultLogger{})

// Set another logger as enforcer e2's logger.
e2.SetLogger(&YouOwnLogger)

// Set your logger when initializing enforcer e3.
e3, _ := casbin.NewEnforcer("examples/rbac_model.conf", a,
logger)
```

Supported loggers

We provide some loggers to help you log information.

Go PHP

Logger	Author	Description
Default	Casbin	The default logger using golang log.

Logger	Author	Description
logger (built-in)		
Zap logger	Casbin	Using zap, provide json encoded log and you can customize more with your own zap-logger.

Logger	Author	描述
psr3-bridge logger	Casbin	Provides a PSR-3 compliant bridge.

How to write a logger

Your logger should implement the Logger interface.

Method	Type	Description
EnableLog()	mandatory	Control whether to print the message.
IsEnabled()	mandatory	Show the current logger's enabled status.
LogModel()	mandatory	Log info related to the model.
LogEnforce()	mandatory	Log info related to enforcing.
LogRole()	mandatory	Log info related to the role.
LogPolicy()	mandatory	Log info related to the policy.

You can pass your custom `logger` to `Enforcer.SetLogger()`.

Here is an example of how to customize a logger for Golang:

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

// DefaultLogger is the implementation for a Logger using
golang log.
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())
}
```

Error handling

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's bug.

There are five main functions you may need to be aware of for errors or panics:

Function	Behavior on error
NewEnforcer()	Returns an error
LoadModel()	Returns an error
LoadPolicy()	Returns an error
SavePolicy()	Returns an error
Enforce()	Returns an error

> **备注**
>
> `NewEnforcer()` calls `LoadModel()` and `LoadPolicy()` internally. So you don't have to call the latter two when using `NewEnforcer()`.

Enable and disable

The enforcer can be disabled via the `Enforcer.EnableEnforce()` function. When it's disabled, `Enforcer.Enforce()` will always return `true`. Other operations like adding or removing policies are not affected. Here's an example:

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

// Will return false.
// By default, the enforcer is enabled.
e.Enforce("non-authorized-user", "data1", "read")

// Disable the enforcer at runtime.
e.EnableEnforce(false)

// Will return true for any request.
e.Enforce("non-authorized-user", "data1", "read")

// Enable the enforcer again.
e.EnableEnforce(true)

// Will return false.
e.Enforce("non-authorized-user", "data1", "read")
```

Frontend Usage

Casbin.js是一个Casbin插件，它可以方便你在前端应用程序中管理访问控制。

安装

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

或

```
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封装
rbac-react	React	@daobeng	在React中使用HOCs、CASL和Casbin.js实现基于角色的访问控制
vue-	Vue	Casbin	Casbin.js的Vue封装

中间件	类型	作者	描述
authz			
angular-authz	Angular	Casbin	Casbin.js的Angular封装

快速开始

你可以在你的前端应用程序中使用`manual`模式，并在你希望的时候设置权限。

```
const casbinjs = require("casbin.js");
// Set the user's permission:
// He/She can read `data1` and `data2` objects and can write
`data1` object
const permission = {
    "read": ["data1", "data2"],
    "write": ["data1"]
}

// Run casbin.js in manual mode, which requires you to set the
permission manually.
const authorizer = new casbinjs.Authorizer("manual");
```

现在我们有了一个授权器，`authorizer`。 我们可以通过使用`authorizer.can()`和`authorizer.cannot()`API从中获取权限规则。 这两个API的返回值都是JavaScript Promises（详情在此），所以我们应该像这样使用返回值的`then()`方法：

```
result = authorizer.can("write", "data1");
result.then((success, failed) => {
```

`cannot()` API的使用方式与此相同:

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

在上面的代码中，参数中的`success`变量意味着请求得到了结果而没有抛出错误，并不意味着权限规则是`true`。 `failed`变量也与权限规则无关。 只有在请求过程中出现问题时，它才有意义。

你可以参考我们的React示例来看看Casbin.js的实际用法。

权限对象

Casbin.js将接受一个JSON对象来操作访问者的相应权限。 例如:

```
{
    "read": ["data1", "data2"],
    "write": ["data1"]
}
```

上面的权限对象显示访问者可以`读取` `data1`和`data2`对象，而他们只能`写入` `data1`对象。

高级用法

Casbin.js为将你的前端访问控制管理与你的后端Casbin服务集成提供了完美的解决方案。

使用`auto`模式并在初始化Casbin.js `Authorizer`时指定你的端点，它将自动同步权限并操作前端状态。

```
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 will automatically sync the permission with your
backend Casbin service.
authorizer.setUser("Tom");

// Evaluate the permission
result = authorizer.can("read", "data1");
result.then((success, failed) => {
    if (success) {
        // Some frontend procedure ...
    }
});
```

相应地，你需要暴露一个接口（例如，一个RestAPI）来生成权限对象并将其传递给前端。 在你的API控制器中，调用`CasbinJsGetUserPermission`来构造权限对象。 这是一个在Beego中的例子：

> **备注**
>
> 你的端点服务器应该返回类似这样的东西
>
> ```
> {
>  "other":"other",
>  "data": "What you get from `CasbinJsGetPermissionForUser`"
> }
> ```

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

// Controller
func (c *APIController) GetFrontendPermission() {
    // Get the visitor from the GET parameters. (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()
}
```

> **备注**
>
> 目前，`CasbinJsGetPermissionForUser` API只在Go Casbin和Node-Casbin中支持。 如果你希望这个API在其他语言中得到支持，请提出一个问题或在下面留言。

API列表

`setPermission(permission: string)`

设置权限对象。 总是在`manual`模式中使用。

`setUser(user: string)`

设置访问者身份并更新权限。 总是在`auto`模式中使用。

`can(action: string, object: string)`

检查用户是否可以对`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>)`

检查用户是否可以对`objects`中的**所有**对象执行`action`。

`canAny(action: string, objects: Array<object>)`

检查用户是否可以对`objects`中的**任何一个**对象执行`action`。

为什么选择Casbin.js

人们可能会对Node-Casbin和Casbin.js之间的区别感到疑惑。 简单来说，Node-Casbin是在NodeJS环境中实现的Casbin的核心，通常用作服务器端的访问控制管理工具包。 Casbin.js是一个前端库，帮助你在客户端使用Casbin对你的网页用户进行授权。

通常，直接在Web前端应用程序中建立Casbin服务并执行授权/执行任务是不合适的，原因如下：

1. 当有人打开客户端时，执行器将被初始化，并将从后端持久层拉取所有策略。 高并发可能会给数据库带来巨大压力，并消耗大量的网络吞吐量。
2. 将所有策略加载到客户端可能带来安全风险。
3. 难以分离客户端和服务器，以及促进敏捷开发。

我们需要一个工具来简化在前端使用Casbin的过程。 实际上，Casbin.js的核心是在客户端操作当前用户的权限。 如你所述，Casbin.js会从指定的端点获取数据。 这个过程将用户的权限与后端Casbin服务同步。 在获取了权限数据后，开发者可以使用Casbin.js接口来管理用户在前端的行为。

Casbin.js避免了上述两个问题：Casbin服务不再被反复拉起，客户端和服务器之间传递消息的大小也减小了。 我们也避免了在前端存储所有的策略。 用户只能访问他们自己的权限，但对访问控制模型和其他用户的权限一无所知。 此外，Casbin.js还可以有效地在授权管理中解耦客户端和服务器。

编辑器

Online Editor

在网页浏览器中编写Casbin模型和策略

IDE Plugins

Casbin IDE 插件

Online Editor

您也可以使用在线编辑器在您的网页浏览器中编写您的Casbin模型和策略。 它提供了如“语法高亮”和“代码补全”等功能，就像一个编程语言的IDE一样。

Editor Features

The online editor provides several powerful features to enhance your Casbin development experience:

AI-Powered Assistance

The editor includes AI assistance to help you write better Casbin models and policies.

Engine Selection

Choose from multiple Casbin engines including Go, Java, Node.js, Python, Rust, and .NET implementations.

Multiple Engine Support

Switch between different Casbin implementations to test your models across various platforms.

Endpoint Configuration

Configure custom endpoints for testing your Casbin models against different environments.

File Upload

Upload existing model and policy files to quickly get started with the editor.

Use Custom Functions

If you need to use a customized matching function (e.g. "RBAC with Patterns"), you can add it by clicking the "Add Role Matching" button at the bottom left corner of the editor.

.

如果你想写等价的代码，你需要通过相关的API指定模式匹配函数。 有关更多信息，请参阅带模式的RBAC。

> **备注**
>
> The editor supports multiple Casbin implementations, including Node-Casbin (Node.js), JCasbin (Java), Casbin (Go), Casbin-rs (Rust), PyCasbin (Python), and Casbin.NET (C#). You can switch between different implementations in the upper right corner to test your model and policy.
>
> Although the editor validates through a remote CLI, due to environment differences, the validation results may differ slightly from the results you get in your local environment. If you encounter any issues, please submit them to the corresponding Casbin implementation repository.

IDE Plugins

我们为以下IDE提供插件：

JetBrains IDEs

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

VSCode

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

更多

Our Adopters

Casbin's Adopters

Contributing

为Casbin做贡献

Privacy Policy

Casbin网站隐私政策

Terms of Service

Casbin 服务条款

📄 Refund Policy

Casbin网站退款政策

Our Adopters

直接集成

Go Java Node.js Python

名称	描述	模型	策略
VMware Harbor	VMware的开源可信赖的云原生注册项目，用于存储、签名和扫描内容。	代码	Beego ORM
Intel RMD	Intel的资源管理守护进程。	.conf	.csv
VMware Dispatch	一个用于部署和管理无服务器风格应用的框架。	代码	代码
Skydive	一个开源的实时网络拓扑和协议分析器。	代码	.csv
Zenpress	一个用Golang编写的CMS系统。	.conf	Gorm
Argo CD	用于Kubernetes的GitOps持续交付。	.conf	.csv
Muxi Cloud	Muxi Cloud的PaaS，更简单的管理Kubernetes集群的方式。	.conf	Code
EngineerCMS	一个用于管理工程师知识的CMS。	.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档案元数据数据库。	.conf	.csv
Qilin API	ProtocolONE的游戏内容许可管理工具。	Code	.csv
Devtron Labs	适用于Kubernetes的软件交付工作流。	.conf	Xorm
lighty.io	OpenDaylight的SDN控制器解决方案。	README	N/A
Notadd	基于Nest.js的微服务开发架构。	.conf	DB adapter
ARC API	基于Loopback创建的SourceFuse微服务目录。	Usage	Provider
dtrace	EduScaled的跟踪系统。	Commit	N/A

通过插件集成

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 (由Docker推荐)	.conf	.csv
Gobis	Orange的轻量级API网关，用go编写	casbin	Code	Request

Contributing

Casbin是一个强大的授权库，支持许多编程语言的访问控制模型实现。 如果你精通任何一种编程语言，你可以为Casbin的开发做出贡献。 我们始终欢迎新的贡献者。

目前，主要有两种类型的项目：

- **以算法为导向的项目** - 这些项目涉及在不同的编程语言中实现算法。 Casbin支持多种语言，包括Golang、Java、C++、Elixir、Dart和Rust，以及它们的相关产品。

Casbin	jCasbin
生产就绪	生产就绪

PyCasbin	Casbin.NET
生产就绪	生产就绪

- **以应用为导向的项目** - 这些项目与基于Casbin构建的应用有关。

项目	演示	详情	技能栈
Casdoor	Casdoor	Casdoor是一个以UI为主的基于OAuth 2.0/OIDC的集中式身份验证/单点登录(SSO)平台。	JavaScript + React 和 Golang + Beego + SQL
Casnode	Casbin 论坛	Casnode是下一代论坛软件。	JavaScript + React 和 Golang + Beego + SQL
Casbin OA	OA系统	Casbin-OA是Casbin技术写作的官方稿件处理、评估和展示系统。	JavaScript + React 和 Golang +

项目	演示	详情	技能栈
			Beego + MySQL
Casbin 编辑器	Casbin 编辑器	Casbin-editor是一个用于Casbin模型和策略的基于web的编辑器。	TypeScript + React

参与其中

有许多方式可以为Casbin做出贡献。 以下是一些开始的建议:

- **使用Casbin并报告问题!** 在使用Casbin时，报告你遇到的任何问题，以帮助推动Casbin的开发。 无论是bug还是提案，在GitHub上提交问题都是推荐的做法。 然而，在提交问题之前，最好先在Discord或GitHub Discussions上进行讨论。

  > 注意：在报告问题时，请用英语描述你的问题的详细情况。

- **帮助完善文档!** 对文档的贡献是你开始贡献的好起点。

- **帮助解决问题!** 我们准备了一个包含适合初学者的简单任务的表格，不同级别的挑战用不同的标签标记。 你可以在这里查看表格。

拉取请求

Casbin使用GitHub作为其开发平台，因此拉取请求是主要的贡献方式。

在开启一个拉取请求之前，你需要知道一些事情:

- 解释你为什么发送拉取请求，以及它将为仓库做什么。

- 确保拉取请求只做一件事。 如果有多个更改，请将它们分割成单独的拉取请求。

- 如果你正在添加新的文件，请在新文件的顶部包含Casbin许可证。

```
// Copyright 2021 The casbin Authors. All Rights Reserved.
//
// Licensed under the Apache License, Version 2.0 (the
"License");
// you may not use this file except in compliance with the
License.
// You may obtain a copy of the License at
//
//      http://www.apache.org/licenses/LICENSE-2.0
//
// Unless required by applicable law or agreed to in
writing, software
// distributed under the License is distributed on an "AS
IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either
express or implied.
// See the License for the specific language governing
permissions and
// limitations under the License.
```

- 在像Casdoor、Casnode和Casbin OA这样的项目中，你可能需要设置一个演示来向维护者展示你的拉取请求如何帮助项目的开发。

- 在开启一个拉取请求和提交你的贡献时，建议使用以下格式的语义提交：`<type>(<scope>): <subject>`。 `<scope>`是可选的。 更详细的使用方法，请参考Conventional Commits。

许可证

通过对Casbin的贡献，你同意你的贡献将在Apache许可证下被许可。

Privacy Policy

您的隐私对我们非常重要。 Casbin的政策是尊重您的隐私，我们可能会在我们的文档网站以及我们拥有和运营的其他网站上收集您的信息。

我们只有在真正需要提供服务给您的时候才会要求您提供个人信息。 我们通过公平和合法的方式收集信息，同时获得您的知情同意。 我们也会告知您我们为何收集这些信息以及如何使用它们。

我们只保留收集的信息，直到提供您所请求的服务所需的时间为止。 我们存储的数据将在商业上可接受的方式内得到保护，以防止丢失和盗窃，以及未经授权的访问、披露、复制、使用或修改。

除法律要求外，我们不会公开或与第三方分享任何个人识别信息。

我们的网站可能会链接到我们未运营的外部网站。 请注意，我们无法控制这些网站的内容和做法，也无法对其各自的隐私政策承担责任或责任。

您可以拒绝我们对您个人信息的请求，但请理解我们可能无法为您提供一些您期望的服务。

您继续使用我们的网站将被视为接受我们关于隐私和个人信息的做法。 如果您对我们如何处理用户数据和个人信息有任何疑问，欢迎随时联系我们。

此政策自2020年6月29日起生效。

Terms of Service

1. 条款

   通过访问https://casbin.org网站，您同意受这些服务条款、所有适用的法律和法规的约束，并同意您有责任遵守任何适用的当地法律。 如果您不同意这些条款中的任何一条，您被禁止使用或访问此网站。 本网站包含的材料受适用的版权和商标法保护。

2. 使用许可

   a. 允许您临时下载Casbin网站上的材料（信息或软件）的一份副本，仅供个人、非商业的暂时查看。 这是许可的授予，而不是所有权的转让，在此许可下，您不得：

   - i. 修改或复制材料；
   - ii. 将材料用于任何商业目的，或用于任何公开展示（商业或非商业）；
   - iii. 尝试反编译或反向工程Casbin网站上的任何软件；
   - iv. 从材料中删除任何版权或其他专有注释；或
   - v. 将材料转移到另一个人，或在任何其他服务器上“镜像”材料。

   b. 如果您违反这些限制，此许可将自动终止，并且Casbin可以随时终止此许可。 在终止查看这些材料或终止此许可后，您必须销毁您拥有的任何已下载的材料，无论是电子格式还是打印格式。

3. 免责声明

   a. Casbin网站上的材料是按'原样'提供的。 Casbin不做任何保证，明示或暗示，并在此否认和否定所有其他保证，包括但不限于暗示的适销性、特定用途的适用性，或不侵犯知识产权或其他权利的保证。

b. 此外，Casbin不保证或对其网站上的材料的使用或其他与此类材料有关或与本网站链接的任何网站的准确性、可能的结果或可靠性做任何声明。

4. 限制

   在任何情况下，Casbin或其供应商均不对任何损害（包括但不限于数据丢失或利润损失，或由于业务中断而引起的损害）负责，这些损害是由于使用或无法使用Casbin网站上的材料引起的，即使Casbin或Casbin的授权代表已经口头或书面通知了这种损害的可能性。 因为一些司法管辖区不允许对默示保证进行限制，或对间接或偶然损害进行责任限制，所以这些限制可能不适用于您。

5. 材料的准确性

   在Casbin的网站上出现的材料可能包括技术、排版或摄影错误。 Casbin不保证其网站上的任何材料都是准确的、完整的或最新的。 Casbin可能会在不通知的情况下随时更改其网站上包含的材料。 然而，Casbin并未承诺更新这些材料。

6. 链接

   Casbin并未审查所有链接到其网站的站点，对任何此类链接站点的内容不负责任。 任何链接的包含并不意味着Casbin对该站点的认可。 使用任何此类链接的网站风险由用户自行承担。

7. 修改

   Casbin可以随时在不通知的情况下修改其网站的服务条款。 通过使用此网站，您同意遵守这些服务条款的当前版本。

8. 管辖法律

   这些条款和条件受到加利福尼亚州旧金山的法律管辖，并按照这些法律进行解释，您无条件地提交给该州或地点的法院的专属管辖权。

Refund Policy

在大多数情况下，Casbin订阅的付款是不可退款的。

如果您的账户有问题，或者您认为账单有误，请联系支持寻求帮助。