



Overview

Casbin 是一個強大且高效的開源訪問控制庫，支持多種[訪問控制模型](#)，用於全面執行授權。

執行一組規則就像在[策略](#)文件中列出主體、對象和期望的允許動作（或根據您的需求以任何其他格式）一樣簡單。這在 Casbin 使用的所有流程中都是同義的。開發者/管理員對授權的佈局、執行和條件有完全的控制權，這些都是通過[模型](#)文件設置的。Casbin 提供了一個[執行器](#)，用於根據給予執行器的策略和模型文件驗證傳入的請求。

Casbin 支持的語言

Casbin 提供對多種編程語言的支持，隨時準備集成到任何項目和工作流程中：

	Go		Java
Casbin		jCasbin	
可投入生產		可投入生產	

		
PyCasbin	Casbin.NET	
生產就緒	生產就緒	

不同語言的功能集

我們始終盡最大努力使 Casbin 在所有語言中擁有相同的功能集。然而，現實並非那麼美好。

功能	Go	Java	Node.js	PHP	Python	C#	Delphi	Rust	C++	Lua	Dart	Elixir
Enforcement	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBAC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
ABAC	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Scaling ABAC (<code>eval()</code>)							✗	✓	✓	✓	✓	✓
Adapter	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗
Management API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
RBAC API	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Batch API	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✗	✗

功能	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 - ✓ 對於觀察者或角色管理員僅表示在核心庫中具有界面。這並不表明是否有觀察者或角色管理器實現可用。

什麼是Casbin?

Casbin是一個授權庫，可以用於我們希望某個對象或實體被特定用戶或主體訪問的流程中。訪問的類型，即動作，可以是_讀取_、寫入、刪除或開發者設定的任何其他動作。這是Casbin最廣泛使用的情況，稱為“標準”或經典的{主體，對象，動作}流程。

Casbin能夠處理除標準流程以外的許多複雜授權場景。可以增加角色（RBAC）、屬性（ABAC）等。

Casbin的功能

- 強制執行經典的{主體，對象，動作}形式或您自定義的形式的策略。同時支持允許和拒絕授權。
- 處理存取控制模型及其策略的儲存。
- 管理角色與用戶的映射以及角色與角色的映射（即RBAC中的角色層級）。
- 支援內建的超級用戶，如root或administrator。超級用戶可以在沒有明確權限的情況下執行任何操作。
- 提供多種內建運算子以支援規則匹配。例如，keyMatch可以將資源鍵/foo/bar映射到模式/foos*。

Casbin不做的事情

- 驗證（即當用戶登入時驗證username和password）
- 管理用戶或角色的列表。

對於項目來說，管理其用戶、角色或密碼列表更為方便。使用者通常擁有自己的密碼，而Casbin並非設計為一個密碼容器。然而，Casbin會儲存使用者與角色之間的對應關係，以應對基於角色的存取控制（RBAC）場景。

Get Started

安裝

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

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

對於 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
```

在您專案的 `composer.json` 中要求此套件以下載該套件:

```
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 以套件形式提供（目前適用於 Delphi 10.3 Rio），您可以在 IDE 中安裝它。然而，它沒有視覺元件，這意味著您可以獨立於套件使用這些單元。只需在您的專案中導入這些單元（假設您不介意它們的數量）。

```
luarocks install casbin
```

如果您收到錯誤訊息：“您的使用者沒有 /usr/local/lib/luarocks.rocks 中的寫入權限”，您可能需要以特權使用者身份運行該命令，或者使用 `--local` 來使用本地樹。要修復錯誤，您可以在命令後面添加 `--local`，如下所示：

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

新建一個 Casbin 執行器

Casbin 使用配置文件來定義訪問控制模型。

有兩個配置文件: `model.conf` 和 `policy.csv`。`model.conf` 存儲訪問模型, 而 `policy.csv` 存儲具體的用戶權限配置。Casbin 的使用非常直觀。我們只需要創建一個主要的結構: **執行器**。在構造這個結構時, `model.conf` 和 `policy.csv` 將被加載。

換句話說, 要創建一個 Casbin 執行器, 您需要提供一個 **模型** 和一個 **適配器**。

Casbin 提供了一個 **文件適配器**, 您可以使用它。查看 **適配器** 以獲取更多資訊。

- 使用 Model 文件和默認 **文件適配器** 的示例:

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

Rust Lua

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

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

import org.casbin.jcasbin.main.Enforcer;

Enforcer e = new Enforcer("path/to/model.conf", "path/to/
```

```
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;
```

```
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(())
}

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

- 使用 Model 文本與其他適配器：

Go Python

```
import (
    "log"

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

```
import casbin
import casbin_sqlalchemy_adapter

# Use SQLAlchemy Casbin adapter with SQLite 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)
```

檢查權限

在訪問發生之前，將強制鉤子添加到您的代碼中：

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

Rust Lua

```
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"}})  
  
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 還提供了運行時的權限管理 API。例如，您可以如下獲取分配給用戶的所有角色：

[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](#) 以獲取更多用法。

請參考測試用例以獲取更多用法。

How It Works

在 Casbin 中，訪問控制模型被抽象為基於 PERM 元模型（Policy、Effect、Request、 Matchers）的 CONF 文件。切換或升級項目的授權機制就像修改配置一樣簡單。您可以通過組合可用的模型來自定義自己的訪問控制模型。例如，您可以在一個模型中結合 RBAC 角色和 ABAC 屬性，並共享一組策略規則。

PERM 模型由四個基礎組成：Policy、Effect、Request 和 Matchers。這些基礎描述了資源和用戶之間的關係。

Request

定義請求參數。基本請求是一個元組對象，至少需要一個主體（訪問實體）、對象（訪問資源）和動作（訪問方法）。

例如，請求定義可能如下所示：

此定義指定了訪問控制匹配函數所需的參數名稱和順序。

Policy

定義訪問策略的模型。它指定了策略規則文檔中的字段名稱和順序。

例如：

注意：如果未定義 eft（策略結果），則策略文件中的結果字段將不被讀取，並且默認情況下將允許匹配的策略結果。

Matcher

定義請求與策略的匹配規則。

例如: `m = r.sub == p.sub && r.act == p.act && r.obj == p.obj` 策略的結果將被保存在 `p.eft` 中。

Effect

對匹配器結果進行邏輯組合判斷。

例如: `e = some(where(p.eft == allow))`

這個語句意味著，如果匹配策略結果 `p.eft` 有（某些）允許的結果，則最終結果為真。

讓我們再看另一個例子：

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

這個例子組合的邏輯意義是：如果存在一個策略匹配允許的結果，且沒有策略匹配拒絕的結果，則結果為真。換句話說，當所有匹配策略都是允許時，這是正確的。如果有任何拒絕，兩者都是錯誤的（更簡單地說，當允許和拒絕同時存在時，拒絕優先）。

Casbin中最基本和最簡單的模型是ACL。 ACL模型的CONF如下：

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

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

ACL模型的一個示例策略是：

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

這意味著：

- alice可以讀取data1
- bob可以寫入data2

我們還支持通過在末尾添加"來進行多行模式：

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

此外，如果你正在使用ABAC，你可以嘗試'in'操作符，如下面的示例所示，適用於Casbin的golang版本（jCasbin和Node-Casbin尚不支持）：

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

但您必須確保陣列的長度大於 1，否則將會引發恐慌。

更多運算子，您可以查看 [govaluate](#)。

Tutorials

在閱讀之前，請注意有些教學指南是針對 Casbin 的模型，並適用於不同語言的所有 Casbin 實現。其他一些教學指南則是特定於某種語言的。

我們的論文

- PML：一種基於解釋器的網絡服務訪問控制策略語言

這篇論文深入探討了 Casbin 的設計細節。如果您在論文中引用 Casbin/PML 作為參考，請引用以下 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}  
}
```

- 基於元模型的訪問控制策略規範語言（中文）

這是另一篇發表在《軟件學報》的較長版本論文。不同格式（Refworks、EndNote 等）的引用資訊可以在以下連結找到：[\(另一版本\) 基於元模型的訪問控制策略規範語言（中文版）](#)

視頻

- 安全金庫 - 使用 Casbin 實現授權中間件 - JuniorDevSG

- 基於 Casbin 的微服務架構中的用戶權限共享（俄語版）
- Nest.js - Casbin RESTful RBAC 授權中間件
- Gin 教程第 10 章：30 分鐘學習 Casbin 基本模型
- Gin 教程第 11 章：Casbin 中的編碼、API 和自定義函數
- Gin + Casbin：實戰學習權限控制（中文版）
- jCasbin 基礎：一個簡單的 RBAC 示例（中文）
- 基於 Casbin 的 Golang RBAC（中文）
- 學習 Gin + Casbin (1)：開啟與概覽（中文）
- ThinkPHP 5.1 + Casbin：入門介紹（中文）
- ThinkPHP 5.1 + Casbin：RBAC 授權（中文）
- ThinkPHP 5.1 + Casbin：RESTful 與中間件（中文）
- PHP-Casbin 快速入門（中文）
- ThinkPHP 5.1 + Casbin：如何使用自定義匹配函數（中文）
- Webman + Casbin：如何使用 Webman Casbin 插件（中文）

PERM 元模型（政策、效果、請求、匹配器）

- 透過不同的存取控制模型配置理解 Casbin
- 使用 PERM 在 Casbin 中建模授權
- 使用 Casbin 設計一個靈活的權限系統
- 使用存取控制列表進行授權
- 使用 PERM 和 Casbin 進行存取控制（波斯文）
- [基於角色的存取控制（RBAC）？[基於屬性的存取控制（ABAC）？..[權限、角色、實體模型（PERM）！雲端網路服務和應用的新授權方式（俄文）
- 使用 Casbin 與 PERM 進行靈活授權的實踐與示例（俄文）
- 使用 Casbin 進行權限管理（中文）
- Casbin 分析（中文）
- 系統權限設計（中文）

- Casbin: 一個權限引擎 (中文)
- 使用Casbin實現ABAC (中文)
- Casbin源碼分析 (中文)
- 使用Casbin進行權限評估 (中文)
- Casbin: Go語言的每日庫 (中文)

Go Java Node.js PHP .NET Rust Lua

HTTP與RESTful

- Go 中基於角色的 HTTP 授權與 Casbin (或中文翻譯)

監視器

- 通過 Casbin 監視器實現 RBAC 分佈式同步 (中文)

Beego

- 在 Beego 中使用 Casbin: 1. 開始與測試 (中文)
- 在 Beego 中使用 Casbin: 2. 策略存儲 (中文)
- 在 Beego 中使用 Casbin: 3. 策略查詢 (中文)
- 使用 Casbin 與 Beego: 4. 政策更新 (中文)
- 使用 Casbin 與 Beego: 5. 政策更新 (續) (中文)

Gin

- 在 Golang 項目中使用 Casbin 進行授權
- 教程: 將 Gin 與 Casbin 整合
- 在 K8s 上使用 Pipeline 進行政策執行
- 在 Gin 應用中使用 JWT 和 Casbin 進行身份驗證和授權
- [使用 Go 構建後端 API: 1. 基於 JWT 的驗證 (中文)]
- [使用 Go 構建後端 API: 2. 基於 Casbin 的授權 (中文)]

- 在 Gin 和 GORM 中使用 Go 的授權庫 Casbin (日文)

Echo

- 使用 Casbin 進行 Web 授權

Iris

- Iris + Casbin: 權限管理的實踐 (中文)
- 從零開始學習 iris + Casbin

Argo CD

- 在 Argo CD 中使用 Casbin 實現組織 RBAC

GShark

- GShark: 輕鬆有效地掃描 Github 中的敏感信息 (中文)

SpringBoot

- jCasbin: 一個更輕量級的權限管理解決方案 (中文)
- 將 jCasbin 與 JFinal 集成 (中文)

Express

- 如何在 AWS 上的無服務器 HTTP API 中添加基於角色的訪問控制

Koa

- 使用 Casbin 和 Koa 進行授權 Part 1
- 使用 Casbin 和 Koa 進行授權 (第二部分)

Nest

- 如何使用 Casbin 和 Nest.js 創建基於角色的授權中間件
- nest.js: Casbin RESTful RBAC 授權中間件 (視頻)
- 基於 Casbin 的 Node.js 屬性基礎訪問控制示範應用

- 多租戶 SaaS 啟動套件，具有 cqrs graphql 微服務架構

Fastify

- 在 Node.js 中使用 Fastify 和 Casbin 進行訪問控制
- Casbin，為您的項目提供強大且高效的 ACL
- 在 dotnet 中使用 Casbin 進行授權
- Rust 中基於角色的 HTTP 授權與 Casbin
- [如何在 Rust 網絡應用中使用 Casbin 授權 第一部分]
- [如何在 Rust 網絡應用中使用 Casbin 授權 第二部分]

APISIX

- 在 APISIX 中使用 Casbin 進行授權



基礎知識

Understanding How Casbin Matching Works in Detail

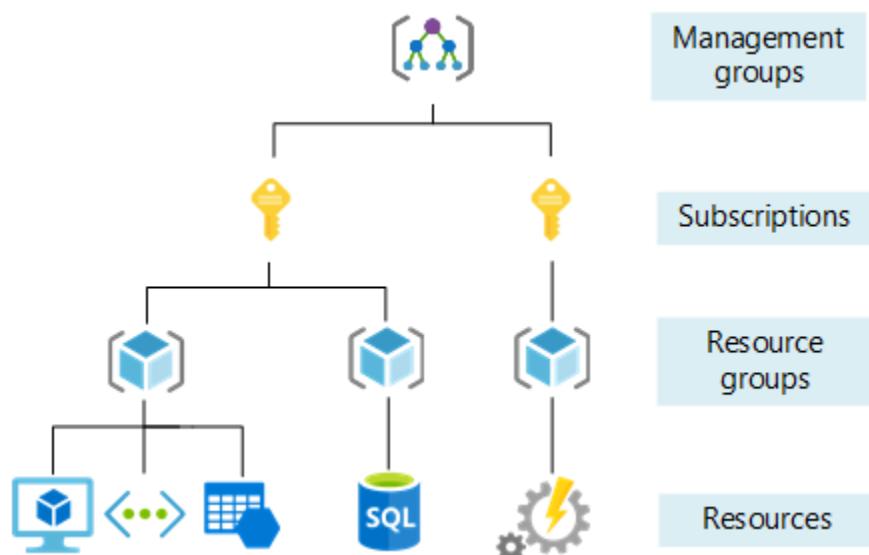
Understanding How Casbin Matching Works in Detail

在這篇文章中，我將解釋如何使用 [Casbin](#) 庫來設計和實現基於角色的訪問控制（RBAC）。對於處理多個資源層次結構和角色從更高層次繼承權限的 SaaS 平台，Casbin 提供了一個高效的替代方案來考慮。

基於角色的訪問控制（RBAC）介紹

基於角色的訪問控制（RBAC）是一種根據個人所擔任的角色來限制對資源的訪問的方法。為了更好地理解分層 RBAC 的工作原理，讓我們在下一節中看看 Azure 的 RBAC 系統，然後嘗試實現一個類似的系統。

理解 Azure 的分層 RBAC



在 Azure 中，所有資源都有一個名為 **擁有者** 的角色。假設我在訂閱層級被分配了 **擁有者** 角色，這意味著我是該訂閱下所有資源群組和資源的 **擁有者**。如果我在資源群組層級擁有 **擁有者** 角色，那麼我就是該資源群組下所有資源的 **擁有者**。

這張圖片顯示我在訂閱層級擁有 **擁有者** 訪問權限。

The screenshot shows the Microsoft Azure portal's Access control (IAM) page for a specific subscription. The 'Role assignments' tab is active. A search bar at the top contains the email address 'aravind.kumar@bootlabtech.com'. The main table displays one item, which is a role assignment for the user 'aravindakumar'. The assignment details are as follows:

Name	Type	Role	Scope	Condition
aravindakumar	User	Owner	This resource	None

當我查看此訂閱下某個資源群組的 IAM 時，你可以看到我從訂閱繼承了 **擁有者** 訪問權限。

The screenshot shows the Microsoft Azure portal interface. At the top, there's a navigation bar with 'Microsoft Azure' and a search bar. Below it, the URL 'Home > Resource groups > test-resource-group' is visible. The main content area has a title 'test-resource-group | Access control (IAM)' with a 'Resource group' icon. On the left, there's a sidebar with various sections like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings (Deployments, Security, Policies, Properties, Locks), Cost Management (Cost analysis, Cost alerts (preview), Budgets, Advisor recommendations), and Monitoring (Insights (preview), Alerts, Metrics, Diagnostic settings). The 'Access control (IAM)' section is expanded. Underneath it, there are tabs for 'Check access', 'Role assignments' (which is selected), 'Roles', 'Deny assignments', and 'Classic administrators'. A sub-section titled 'Number of role assignments for this subscription' shows '31' total assignments and '4000' maximum. A search bar at the top of this section contains 'aravind'. Below the search bar, there are filters for 'Type: All', 'Role: All', 'Scope: All scopes', and 'Group by: Role'. A message indicates 'Showing a filtered set of results. Total number of role assignments: 31'. The results table has columns for Name, Type, Role, Scope, and Condition. One row is shown: 'aravind' (User, Owner role, Subscription (Inherited), None condition).

因此，這就是 Azure 的 RBAC 具有層次結構的方式。大多數企業軟件使用層次結構的 RBAC，因為資源層級具有層次結構的本質。在本教程中，我們將嘗試使用 Casbin 實現一個類似的系統。

Casbin 是如何運作的？

在深入實施之前，了解 Casbin 是什麼以及它在高層次上如何運作是非常重要的。這種理解是必要的，因為每個基於角色的訪問控制（RBAC）系統可能會根據具體需求而有所不同。通過掌握 Casbin 的運作方式，我們可以有效地微調模型。

什麼是 ACL？

ACL 代表訪問控制列表。它是一種將用戶映射到操作，並將操作映射到資源的方法。

模型定義

讓我們考慮一個簡單的 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** 是系統的查詢模板。舉例來說，請求 `alice, write, data1` 可以被解讀為 "主體 Alice 能否執行對物件 'data1' 的 'write' 動作？"。
2. **policy_definition** 是系統的分配模板。例如，透過創建一個策略 `alice, write, data1`，您正在授予主體 Alice 對物件 'data1' 執行 'write' 動作的權限。
3. **policy_effect** 定義了策略的效果。
4. 在 **matchers** 部分，請求會根據條件 `r.sub == p.sub && r.obj == p.obj && r.act == p.act` 與策略進行匹配。

現在讓我們在 Casbin 編輯器上測試這個模型

打開 [編輯器](#) 並將上述模型貼到模型編輯器中。

在策略編輯器中貼上以下內容：

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

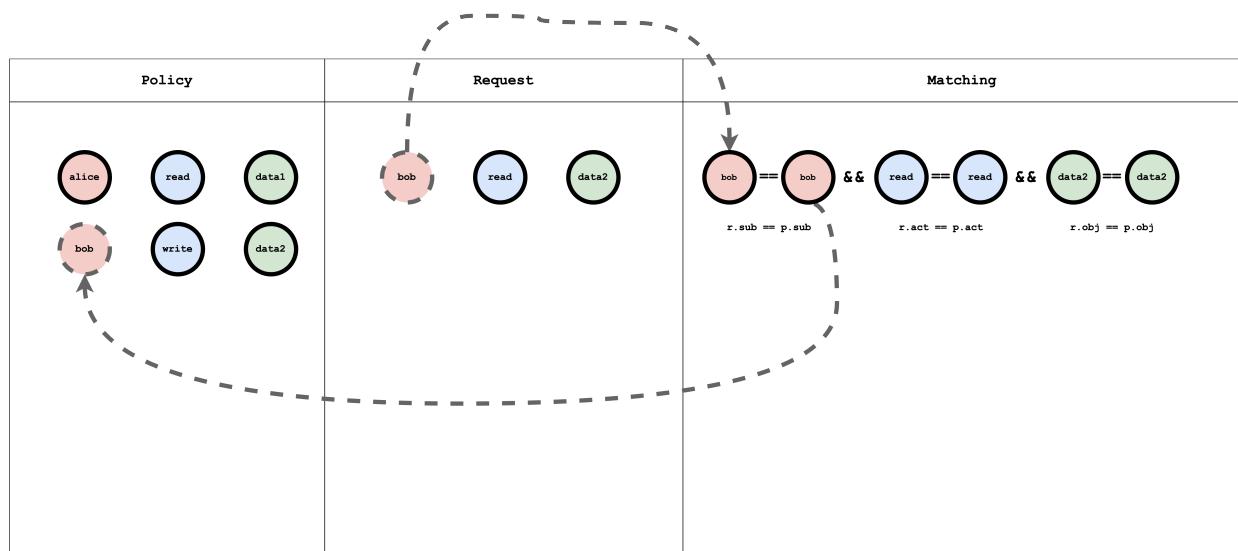
以及在請求編輯器中貼上以下內容：

```
alice, read, data1
```

結果將會是：

```
true
```

ACL 模型、政策及請求匹配的視覺呈現



什麼是 RBAC?

RBAC 代表基於角色的存取控制。在 RBAC 中，用戶被分配一個資源的角色，而角色可以包含任意操作。然後請求會檢查用戶是否有權限執行該操作於該資源上。

模型定義

讓我們考慮一個簡單的 RBAC 模型例子：

```
[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` 是一個圖形關係構建器，使用圖形來比較請求對象與政策對象。

現在讓我們在 Casbin 編輯器上測試這個模型

打開[編輯器](#)並將上述模型貼到模型編輯器中。

在策略編輯器中貼上以下內容：

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

g, reader, read
```

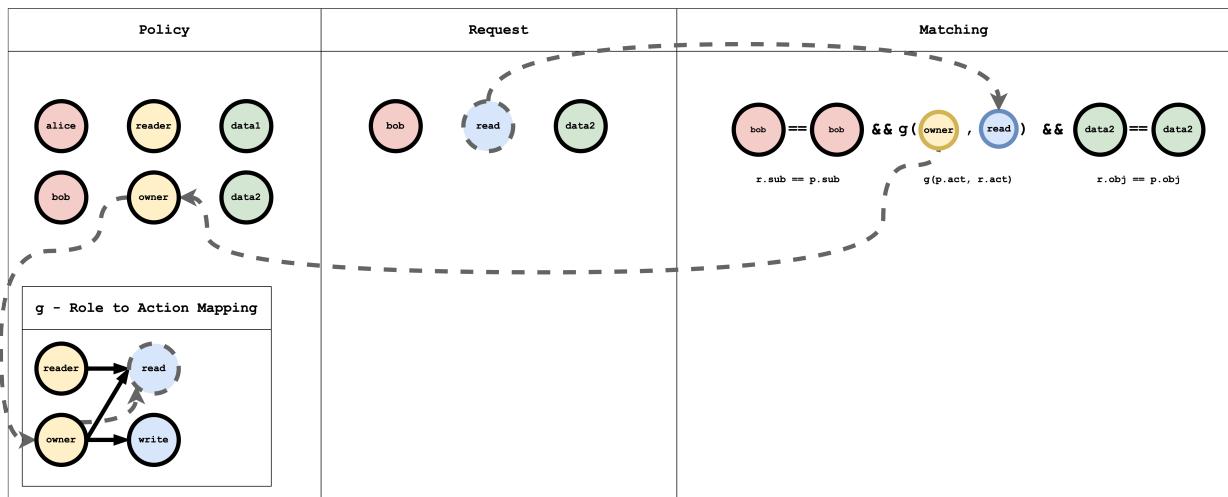
以及在請求編輯器中貼上以下內容：

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

結果將是：

```
true  
false  
true  
true  
false
```

RBAC 模型、策略和請求匹配的視覺表示



g - 角色到操作的映射表格有一個圖表將角色映射到操作。這個圖表可以編碼為一組邊緣列表，如策略中所示，這是表示圖表的常見方式：

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

① 信息

p 表示一個可以使用 `==` 運算符進行比較的普通策略。 **g** 是一個基於圖表的比較函數。 您可以透過添加數字後綴（如 `g, g2, g3, ...` 等）來定義多個圖形比較器。

什麼是層次式 RBAC？

在層次式 RBAC 中，存在多種類型的資源，並且資源類型之間存在繼承關係。例如，「訂閱」是一種類型，而「資源群組」是另一種類型。一個類型為 `訂閱` 的 `sub1` 可以包含多個類型為 `資源群組` 的 `rg1` 和 `rg2`。

類似於資源層次結構，將存在兩種類型的角色和操作：訂閱角色和操作，以及資源群組角色和操作。訂閱角色和資源群組角色之間存在任意關係。例如，考慮一個訂閱角色 `sub-owner`。這個角色被資源群組角色 `rg-owner` 繼承，這意味著如果我在訂閱 `sub1` 上被分配了 `sub-owner` 角色，那麼我也自動獲得在 `rg1` 和 `rg2` 上的 `rg-owner` 角色。

模型定義

讓我們以一個簡單的 層次式RBAC 模型為例：

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

1. 角色定義 是一個圖關係構建器，它使用一個圖來比較請求對象與策略對象。

現在讓我們在 Casbin 編輯器上測試這個模型

打開 [編輯器](#) 並將上述模型貼到模型編輯器中。

在策略編輯器中貼上以下內容：

```
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
```

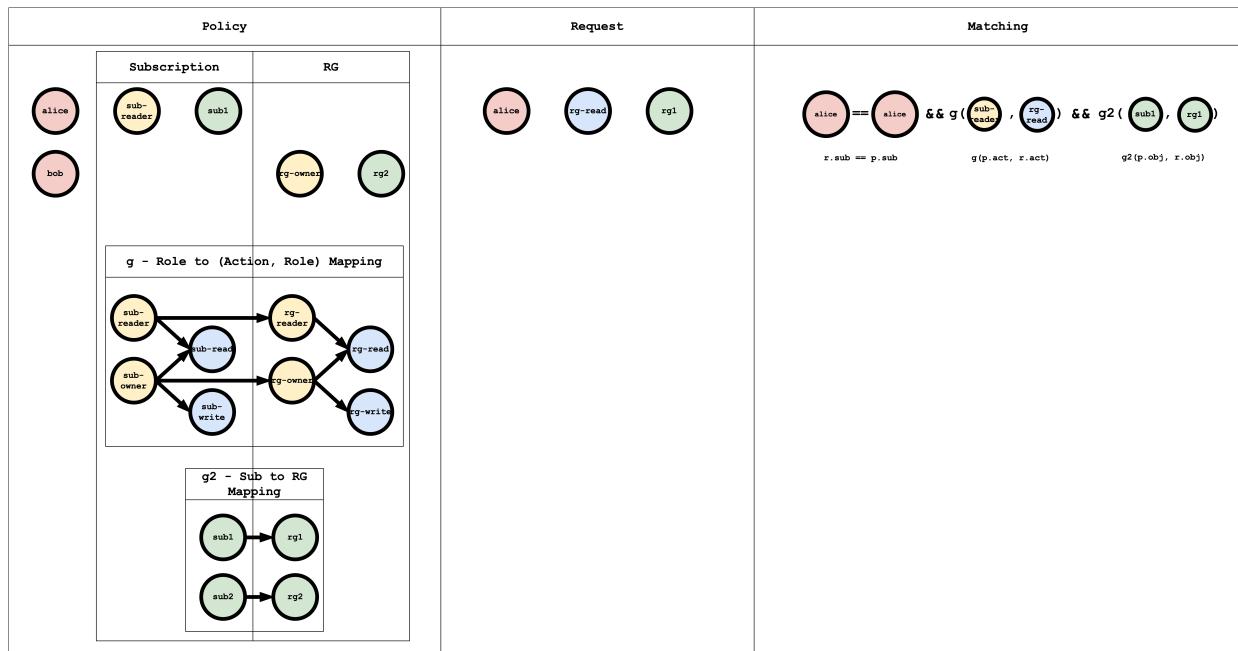
在請求編輯器中貼上以下內容：

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

結果將是：

true

RBAC 模型的視覺表示，策略和請求匹配



g - 角色到 (動作, 角色) 映射 表有一個圖，將角色映射到動作，角色映射。此圖可以編碼為一個邊列表，如政策所示，這是表示圖的常見方式：

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

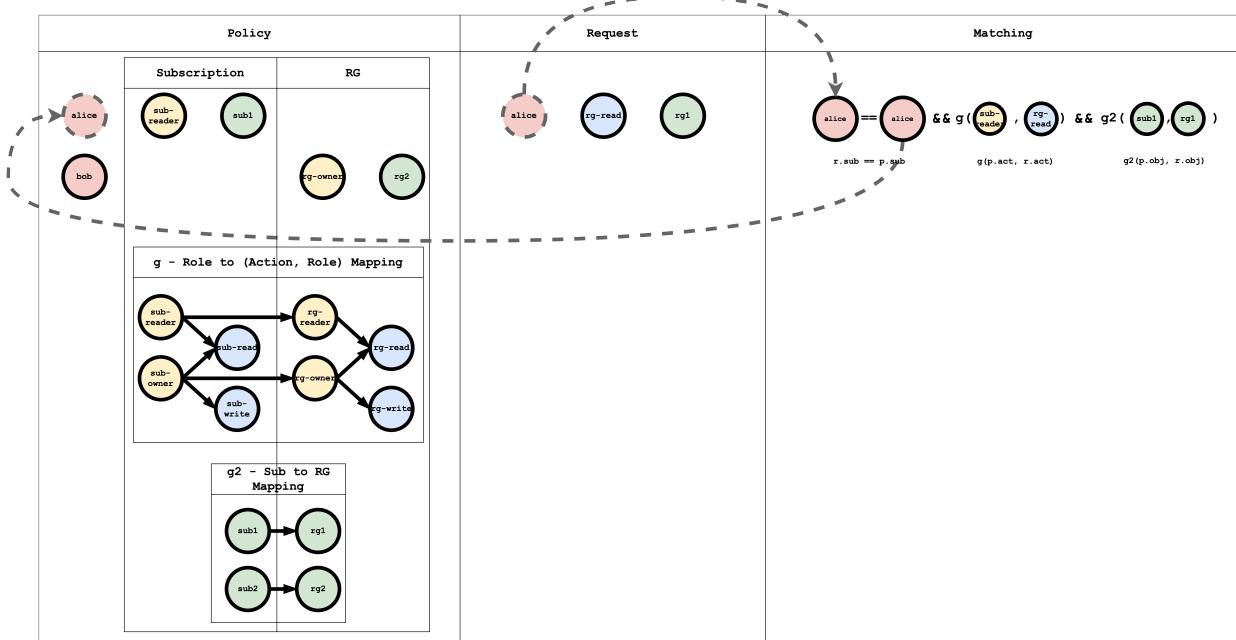
// resourceGroup role to resourceGroup action mapping
g, rg-reader, rg-read
g, rg-owner, rg-read
g, rg-owner, rg-write

// subscription role to resourceGroup role mapping
```

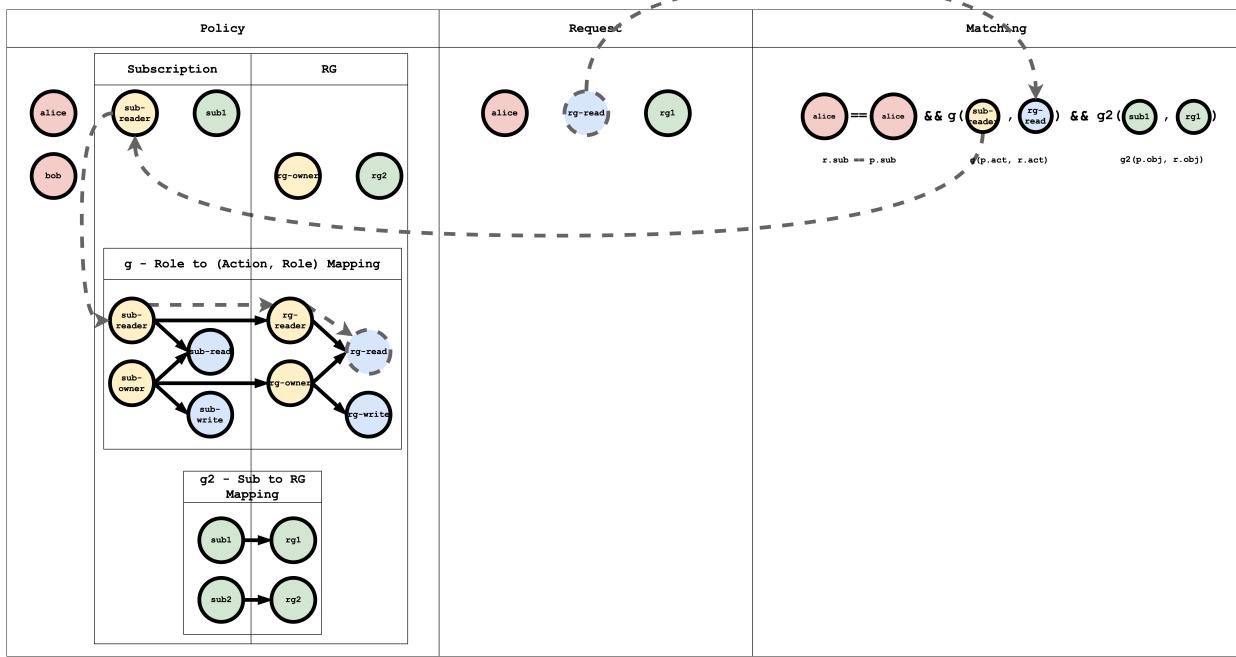
g2 - Sub to RG Mapping 表格具有將訂閱映射到資源組的圖形映射：

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

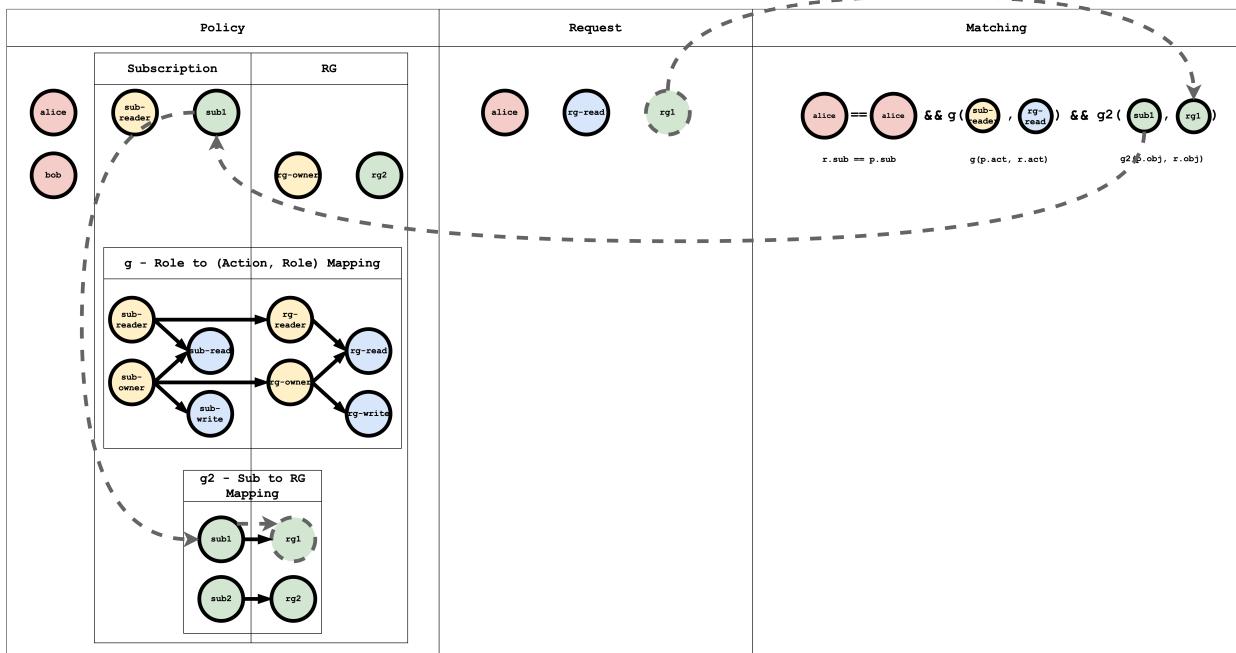
主體匹配視覺表示



動作匹配視覺表示



對象匹配視覺表示



① 信息

當向 Casbin 提交請求時，所有政策都會進行此匹配。如果至少有一個政策匹配，則請求的結果為真。如果沒有任何政策符合請求，則結果為假。

結論

在本教程中，我們學習了不同的授權模型如何運作，以及如何使用Casbin對其進行建模。在本教程的第二部分中，我們將在一個演示的Spring Boot應用程序中實現這一點，並使用Casbin保護API。



>

機型

機型



Supported Models

Casbin 支援的模型



Syntax for Models

模型語法



Effector

Casbin 中的 Effector 介面



Functions

使用內建函數或指定自訂函數

RBAC

Casbin RBAC 使用

RBAC with Pattern

帶有模式的RBAC

RBAC with Domains

RBAC 在域中的使用

RBAC with Conditions

RBAC帶條件的使用

Casbin RBAC vs. RBAC96

Casbin RBAC 與 RBAC96 之間的區別

 **ABAC**

基於Casbin的ABAC

 **Priority Model**

Casbin 的優先級模型用於管理具有不同優先級的政策

 **Super Admin**

超級管理員是整個系統的管理者。 它可以用於RBAC、ABAC以及帶有域的RBAC等模型。

Supported Models

1. **ACL (存取控制列表)**
2. 帶有**超級用戶的ACL**
3. **無用戶的ACL**: 這對於沒有驗證或用戶登入的系統特別有用。
4. **無資源的ACL**: 在某些情況下，目標是某種類型的資源而非特定資源。可以使用如 "寫文章" 和 "讀日誌" 這樣的權限。這不控制對特定文章或日誌的存取。
5. ****RBAC (基於角色的存取控制) ****
6. **RBAC 與資源角色**: 用戶和資源可以同時擁有角色（或群組）。
7. **RBAC 與域/租戶**: 用戶可以為不同的域/租戶擁有不同的角色集合。
8. **ABAC (基於屬性的訪問控制)** : 可以使用類似 "resource.Owner" 的語法糖來獲取資源的屬性。
9. **RESTful**: 支持路徑如 "/res/*"、"/res/:id" 以及 HTTP 方法如 "GET"、"POST"、"PUT"、"DELETE"。
10. **拒絕覆蓋**: 同時支持允許和拒絕授權，其中拒絕覆蓋允許。
11. **優先級**: 策略規則可以被賦予優先級，類似於防火牆規則。

示例

模型	模型文件	策略文件
ACL	基本模型配置文件	基本政策CSV文件
帶有超級 用戶的ACL	帶有根用戶的基本模型配置文件	基本政策CSV文件

模型	模型文件	策略文件
不包含用 戶的ACL	不包含用戶的基本模型配置文件	不包含用戶的基本政策CSV文件
不包含資 源的ACL	無資源的基本模型配置	無資源的基本政策CSV
基於角 色的訪問控 制 (RBAC)	基於角色的訪問控制模型配置	基於角色的訪問控制政策CSV
帶有資 源角 色的基 於角 色的 訪問控 制	帶有資源角色的基於角色的訪問控 制模型配置	帶有資源角色的基於角色的訪問 控制政策CSV
帶有域/租 戶的基於 角色的訪 問控制	帶有域的基於角色的訪問控制模型 配置	rbac_with_domains_policy.csv
ABAC	abac_model.conf	N/A
RESTful	keymatch_model.conf	keymatch_policy.csv
拒絕覆蓋	rbac_with_not_deny_model.conf	rbac_with_deny_policy.csv
允許與拒 絕	rbac_with_deny_model.conf	rbac_with_deny_policy.csv

模型	模型文件	策略文件
優先級	priority_model.conf	priority_policy.csv
明確優先級	priority_model_explicit	priority_policy_explicit.csv
主體優先級	subject_priority_model.conf	subject_priority_policyl.csv

Syntax for Models

- 模型配置（CONF）應至少包含四個部分：[request_definition]、[policy_definition]、[policy_effect] 和 [matchers]。
- 如果模型使用基於角色的訪問控制（RBAC），則還應包括 [role_definition] 部分。
- 模型配置（CONF）可以包含註釋。 註釋以 # 符號開始，# 符號之後的所有內容都將被註釋掉。

請求定義

[request_definition] 部分定義了 e.Enforce(...) 函數中的參數。

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

在此示例中，sub、obj 和 act 代表經典的訪問三元組：主體（訪問實體）、對象（被訪問資源）和動作（訪問方法）。然而，您可以自訂您自己的請求格式。例如，如果您不需要指定特定資源，可以使用 sub, act，或者如果您有兩個存取實體，可以使用 sub, sub2, obj, act。

政策定義

[policy_definition] 是政策的定義。 它定義了政策的含義。 例如，我們有以下模型：

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

並且我們有以下政策（如果在政策文件中）：

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

政策中的每一行稱為一個政策規則。每個政策規則以一個 `政策類型` 開始，例如 `p` 或 `p2`。它用於匹配政策定義，如果有多個定義。上述政策顯示了以下約束。該約束可用於匹配器中。

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

💡 提示

政策規則中的元素總是被視為 `字串`。如果您對此有任何疑問，請參閱以下討論：<https://github.com/casbin/casbin/issues/113>

政策效果

`[policy_effect]` 是政策效果的定義。它決定了當多個政策規則匹配請求時，是否應批准訪問請求。例如，一條規則允許，另一條規則拒絕。

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

上述政策效果意味著如果有任何匹配的政策規則為 `allow`，最終效果為 `allow`（也稱為允許覆蓋）。`p.eft` 是政策的效應，它可以是 `allow` 或 `deny`。這是可選的，預設值為 `allow`。由於我們在上面沒有指定它，它使用預設值。

另一個政策效果的例子是：

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

這意味著如果沒有匹配的 `deny` 政策規則，最終效果是 `allow`（也稱為拒絕覆蓋）。`some` 意味著存在一個匹配的政策規則。`any` 意味著所有匹配的政策規則（在此未使用）。政策效果甚至可以與邏輯表達式相連接：

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

這意味著必須至少有一個匹配的 `allow` 政策規則，並且不能有任何匹配的 `deny` 政策規則。因此，通過這種方式，既支援允許也支援拒絕授權，並且拒絕覆蓋。

① 注意

儘管我們如上設計了政策效果的語法，當前的實現僅使用硬編碼的政策效果。這是因為我們發現對於那種程度的靈活性需求並不大。因此，目前您必須使用內建的政策效果之一，而不是自定義您自己的。

支援的內建政策效果包括：

政策效果	含義	範例
<code>some(where (p.eft == allow))</code>	允許覆蓋	<code>ACL,</code>

政策效果	含義	範例
		RBAC, 等
!some(where (p.eft == deny))	拒絕覆蓋	拒絕覆蓋
some(where (p.eft == allow)) && !some(where (p.eft == deny))	允許與拒絕	允許與拒絕
priority(p.eft) deny	優先權	優先權
subjectPriority(p.eft)	基於角色的優先權	主題優先

匹配器

[matchers] 是政策匹配器的定義。 匹配器是表達式，定義了政策規則如何針對請求進行評估。

[matchers]

```
m = r.sub == p.sub && r.obj == p.obj && r.act == p.act
```

上述匹配器是最簡單的，意味著請求中的主題、對象和動作應與政策規則中的相匹配。

算術運算符如 +, -, *, / 和邏輯運算符如 &&, ||, ! 可以用於匹配器中。

表達式順序在匹配器中

表達式的順序可以極大地影響性能。 請參考以下示例以獲取更多詳情：

```

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)
    }
}

```

執行時間可能非常長，最長可達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，如果我們調整匹配器中表達式的順序，將更耗時的表達式如函數放在後面，執行時間將會非常短。

將上述示例中匹配器內表達式的順序更改為：

```
[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
```

多種區段類型

如果你需要多個策略定義或多個匹配器，你可以使用 `p2` 或 `m2` 作為示例。事實上，上述提到的所有四個區段都可以使用多種類型，語法是 `r` 後跟一個數字，例如 `r2` 或 `e2`。默認情況下，這四個區段應該一對一對應。例如，你的 `r2` 區段將僅使用 `m2` 匹配器來匹配 `p2` 策略。

你可以將一個 `EnforceContext` 作為 `enforce` 方法的第一個參數傳遞，以指定類型。

`EnforceContext` 定義如下：

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

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

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

這裡是一個使用示例。請參考 [模型](#) 和 [政策](#)。請求如下：

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

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

// Pass in a suffix as a parameter to NewEnforceContext, such
```

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

特殊語法

您也可以使用 "in" 運算子，這是唯一具有文字名稱的運算子。此運算子檢查右側的陣列，看它是否包含一個等於左側值的值。等式是通過使用 == 運算子來確定的，並且此庫不檢查值之間的類型。只要兩個值可以轉換為 interface{} 並且仍然可以通過 == 檢查等式，它們就會按預期運作。請注意，您可以使用陣列的參數，但它必須是 []interface{}。

另請參考 `rbac_model_matcher_using_in_op`、`keyget2_model` 和 `keyget_model`。

範例：

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

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

表達式評估器

Casbin 中的匹配器評估是由每種語言的表達式評估器實現的。 Casbin 整合了它們的功能，以提供統一的 PERM 語言。除了此處提供的模型語法外，這些表達式評估器可能提供其他語言或實現不支援的額外功能。使用此功能時請謹慎。

每個 Casbin 實現使用的表達式評估器如下：

實現	語言	表達式評估器
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/

實現	語言	表達式評估器
		TExpressionParser
casbin-rs	Rust	https://github.com/jonathandturner/rhai
casbin-cpp	C++	https://github.com/ArashPartow/exprtk

ⓘ 注意

如果您遇到Casbin的效能問題，很可能是由於表達式評估器的低效率所導致。您可以將問題直接反映給Casbin或表達式評估器，以尋求提升效能的建議。更多詳情，請參閱基準測試部分。

Effector

Effect 代表一條政策規則的結果，而 Effector 是處理 Casbin 中效果的介面。

MergeEffects()

MergeEffects() 函數用於將執行器收集的所有匹配結果合併為單一決策。

例如：

Go

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

在這個例子中：

- Effect 是此函數合併的最終決策（初始化為 Indeterminate）。
- explainIndex 是 eft (Allow 或 Deny) 的索引，初始化為 -1。
- err 用於檢查效果是否被支援。
- expr 是政策效果的字符串表示。
- effects 是一個效果陣列，可以是 Allow、Indeterminate 或 Deny。
- matches 是一個陣列，指示結果是否與政策匹配。
- policyIndex 是模型中政策的索引。
- policyLength 是政策的長度。

上述代碼說明了如何將參數傳遞給 `MergeEffects()` 函數，該函數將根據 `expr` 處理效果和匹配。

要使用 `Effector`，請遵循以下步驟：

Go

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

`MergeEffects()` 的基本思想是，如果 `expr` 能夠匹配結果，表明 `p_eft` 是 `allow`，那麼所有效果都可以合併。如果沒有匹配到拒絕規則，則決策是允許。

ⓘ 注意

如果 `expr` 不符合條件 `"priority(p_eft) || deny"`，並且 `policyIndex` 小於 `policyLength-1`，它將在某些效果中間進行短路操作。

Functions

匹配器中的函數

您甚至可以在匹配器中指定函數以使其更強大。 您可以使用內建函數或指定您自己的函數。 內建的鍵匹配函數採用以下格式：

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

它們返回一個布林值，指示 `url` 是否匹配 `pattern`。

支援的內建函數有：

函數	網址	模式	範例
關鍵匹配	一個類似 <code>/alice_data/resource1</code> 的網址路徑	一個網址路徑或一個類似 <code>/alice_data/*</code> 的 <code>*</code> 模式	keymatch_model.conf/keymatch_policy.csv
關鍵匹配2	一個類似 <code>/alice_data/resource1</code> 的網址路徑	一個網址路徑或一個類似 <code>/alice_data/:resource</code> 的 <code>:</code> 模式	keymatch2_model.conf/keymatch2_policy.csv
keyMatch3	類似 <code>/alice_data/resource1</code> 的 URL 路徑	類似 <code>/alice_data/{resource}</code> 的 URL 路徑或 <code>{}</code> 模式	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L171-L196
keyMatch4	類似 <code>/alice_data/123/book/123</code> 的 URL 路徑	類似 <code>/alice_data/{id}/book/{id}</code> 的 URL 路徑或 <code>{}</code> 模式	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L208-L222
keyMatch5	一個URL路徑，例如 <code>/alice_data/123/?status=1</code>	一個URL路徑，包含 <code>{}</code> 或 <code>*</code> 模式，例如 <code>/alice_data/{id}/*</code>	https://github.com/casbin/casbin/blob/1cde2646d10ad1190c0d784c3a1c0e1ace1b5bc9/util/builtin_operators_test.go#L485-L526
正則表達式匹配	任何字符串	一個正則表達式模式	keymatch_model.conf/keymatch_policy.csv
IP地址匹配	一個IP地址，例	一個IP地址或CIDR，例如	ipmatch_model.conf/ipmatch_policy.csv

函數	網址	模式	範例
	如 192.168.2.123	192.168.2.0/24	
globMatch	類似路徑的路徑，如 /alice_data/ resource1	類似 glob 模式的 /alice_data/*	https://github.com/casbin/casbin/blob/277c1a2b85698272f764d71a94d2595a8d425915/util/builtin_operators_test.go#L426-L466

對於獲取鍵的函數，它們通常接受三個參數（除了 `keyGet`）：

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

它們將返回匹配模式的鍵 `key_name` 的值，如果沒有匹配到任何內容，則返回 ""。

例如，`KeyGet2("/resource1/action", "/:res/action", "res")` 將返回 "resource1"，而 `KeyGet3("/resource1_admin/action", "/{res}_admin/*", "res")` 將返回 "resource1"。至於 `KeyGet`，它接受兩個參數，`KeyGet("/resource1/action", "/")` 將返回 "resource1/action"。

函數	網址	模式	鍵名	範例
鍵獲取	一個類似 /proj/ resource1 的網址路徑	一個網址路徑或一個 * 模式， 如 /proj/*	\	keyget_model.conf/keymatch_policy.csv
鍵獲取2	一個URL路徑，例如 /proj/ resource1	一個URL路徑或 : 模式，例如 /proj/:resource	模式 中指 定的 鍵名 稱	keyget2_model.conf/keymatch2_policy.csv
keyGet3	一個URL路徑，例如 /proj/ res3_admin/	一個URL路徑或 {} 模式，例如 /proj/{resource}_admin/*	模式 中指 定的 鍵名 稱	https://github.com/casbin/casbin/blob/7bd496f94f5a2739a392d333a9aaa10ae397673/util/builtin_operators_test.go#L209-L247

有關上述功能的詳細信息，請參閱：https://github.com/casbin/casbin/blob/master/util/builtin_operators_test.go

如何添加自定義函數

首先，準備你的函數。它接受多個參數並返回一個布林值：

```
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]
}
```

然後，用 `interface{}` 類型包裹它：

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

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

最後，將該函數註冊到 Casbin 執行器中：

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

現在，你可以在你的模型 CONF 中像這樣使用該函數：

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

RBAC

角色定義

[role_definition] 用於定義 RBAC 角色繼承關係。Casbin 支持多個 RBAC 系統實例，其中用戶可以擁有角色及其繼承關係，資源也可以擁有角色及其繼承關係。這兩個 RBAC 系統不會互相干擾。

本節為選擇性內容。如果您在模型中不使用 RBAC 角色，則省略此節。

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

上述角色定義顯示 g 是一個 RBAC 系統，而 g2 是另一個 RBAC 系統。_,_ 表示在繼承關係中涉及兩個角色。在大多數情況下，如果您只需要用戶的角色，通常會單獨使用 g。當您需要為用戶和資源都設置角色（或群組）時，您也可以同時使用 g 和 g2。請參閱 [rbac_model.conf](#) 和 [rbac_model_with_resource_roles.conf](#) 以獲取範例。

Casbin 將實際的用戶角色映射（或如果您使用資源上的角色，則是資源角色映射）存儲在策略中。例如：

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

這意味著 alice 繼承/是角色 data2_admin 的成員。在這裡，alice 可以是用戶、資源或角色。Casbin 僅將其識別為字符串。

然後，在匹配器中，您應該如下檢查角色：

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

這意味著請求中的 `sub` 在策略中應該具有 `sub` 的角色。

① 注意

1. Casbin 僅存儲用戶-角色映射。
2. Casbin 不會驗證用戶是否為有效用戶或角色是否為有效角色。這應該由身份驗證來處理。
3. 在 RBAC 系統中不要使用相同的名稱作為用戶和角色，因為 Casbin 將用戶和角色識別為字符串，並且 Casbin 無法知道您是指定用戶 `alice` 還是角色 `alice`。您可以通過使用 `role_alice` 簡單地解決這個問題。
4. 如果 `A` 具有角色 `B`，而 `B` 具有角色 `C`，則 `A` 具有角色 `C`。目前這種傳遞性是無限的。

① 令牌名稱約定

通常，策略定義中的主體令牌名稱為 `sub` 並放置在開頭。現在，Golang Casbin 支援自訂權杖名稱和位置。如果主體權杖名稱是 `sub`，主體權杖可以放置在任意位置，無需任何額外操作。如果主體權杖名稱不是 `sub`，則在執行器初始化後，無論其位置如何，都應該呼叫 `e.SetFieldIndex()` 對 `constant.SubjectIndex` 進行設置。

```
# `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
```

角色階層

Casbin 的 RBAC 支援 RBAC1 的角色階層功能，這意味著如果 `alice` 擁有 `role1`，而 `role1` 擁有 `role2`，那麼 `alice` 也將擁有 `role2` 並繼承其權限。

在此，我們有一個稱為階層級別的概念。因此，在這個例子中，階層級別是 2。對於 Casbin 內建的角色管理器，您可以指定最大階層級別。預設值是 10。這意味著像 `alice` 這樣的終端用戶只能繼承 10 層角色。

```
// 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  
}
```

如何區分角色與使用者？

Casbin 在其 RBAC 中不區分角色和使用者。它們都被視為字串。如果你只使用單層級

RBAC（其中角色永遠不會是另一個角色的成員），你可以使用 `e.GetAllSubjects()` 來獲取所有使用者，並使用 `e.GetAllRoles()` 來獲取所有角色。它們將分別列出所有 `u` 和所有 `r`，在所有 `g, u, r` 規則中。

但如果你使用多層級 RBAC（帶有角色層次結構），並且你的應用程式不記錄一個名稱（字串）是使用者還是角色，或者你有一個使用者和一個同名的角色，你可以在將角色傳遞給 Casbin 之前為其添加一個前綴，例如 `role::admin`。這樣，你就可以通過檢查這個前綴來知道它是否是一個角色。

如何查詢隱含的角色或權限？

當使用者通過 RBAC 層次結構繼承角色或權限，而不是在策略規則中直接分配給他們時，我們稱這種分配為“隱含”的。要查詢這種隱含關係，你需要使用這兩個 API: `GetImplicitRolesForUser()` 和 `GetImplicitPermissionsForUser()`，而不是 `GetRolesForUser()` 和 `GetPermissionsForUser()`。如需更多詳細信息，請參閱此 [GitHub 問題](#)。

在 RBAC 中使用模式匹配

參見 [帶有模式的 RBAC](#)

角色管理器

詳情請參閱 [角色管理器](#) 部分。

RBAC with Pattern

快速開始

- 在 `g(_, _)` 中使用模式。

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

- 在域中使用模式。

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

- 使用所有模式。

只需結合兩個API的使用。

如上所示，在您創建 `enforcer` 實例後，您需要通過 `AddNamedMatchingFunc` 和 `AddNamedDomainMatchingFunc` API 激活模式匹配，這些API決定了模式如何匹配。

 注意

如果您使用在線編輯器，它會在左下角指定模式匹配函數。

```
    g10match
12      *
  matchingDomainForGFunction:
  'keyMatch'
13      */
14      matchingForGFunction:
  'keyMatch2',
15
  matchingDomainForGFunction:
  'keyMatch2'
16  };
17 })();
```

Request

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

在RBAC中使用模式匹配

有時候，您希望某些主體、對象或具有特定模式的域/租戶能夠自動被授予角色。RBAC中的模式匹配功能可以幫助您實現這一點。模式匹配函數的參數和返回值與之前的[匹配器函數](#)相同。

模式匹配函數支持 `g` 的每個參數。

我們知道，通常RBAC在匹配器中表示為 `g(r.sub, p.sub)`。然後我們可以使用類似以下的策略：

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

因此，`alice`可以閱讀所有書籍，包括 `book 1` 和 `book 2`。但可能有成千上萬本書，如果每本書都要通過一條 `g` 策略規則添加到書籍角色（或群組）中，將會非常繁瑣。

但透過模式匹配函數，您可以僅用一行代碼編寫策略：

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

Casbin 會自動將 `/book/1` 和 `/book/2` 匹配到模式 `/book/:id`。您只需像這樣向 enforcer 註冊函數：

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

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

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

在使用模式匹配函數於域/租戶時，您需要向 enforcer 和模型註冊該函數。

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

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

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

如果您不明白 `g(r.sub, p.sub, r.dom)` 的含義，請閱讀 [帶域的RBAC](#)。簡而言之，`g(r.sub, p.sub, r.dom)` 會檢查用戶 `r.sub` 是否在域 `r.dom` 中擁有角色 `p.sub`。這就是匹配器的工作原理。您可以在此處查看完整示例 [這裡](#)。

除了上述模式匹配語法外，我們還可以使用純域模式。

例如，如果我們希望 `sub` 在不同的域 `domain1` 和 `domain2` 中擁有訪問權限，我們可

以使用純域模式：

```
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
```

在此範例中，我們希望 `alice` 能夠在 `domain1` 和 `domain2` 中讀取和寫入 `data`。在 `g` 中使用模式匹配 `*` 使得 `alice` 能夠存取兩個域。

透過使用模式匹配，尤其是在更複雜且需要考慮大量域或對象的情境中，我們能夠以更優雅且有效的方式實現 `policy_definition`。

RBAC with Domains

帶域租戶的角色定義

Casbin 中的 RBAC 角色可以是全域的或特定於域的。特定於域的角色意味著用戶在不同域/租戶中的角色可以不同。這對於大型系統（如雲端）非常有用，因為用戶通常位於不同的租戶中。

帶域/租戶的角色定義應該看起來像這樣：

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

第三個 `_` 代表域/租戶的名稱，這部分不應更改。然後策略可以是：

```
p, admin, tenant1, data1, read  
p, admin, tenant2, data2, read  
  
g, alice, admin, tenant1  
g, alice, user, tenant2
```

這意味著在 `tenant1` 中的 `admin` 角色可以讀取 `data1`。而 `alice` 在 `tenant1` 中擁有 `admin` 角色，在 `tenant2` 中擁有 `user` 角色。因此，她可以讀取 `data1`。然而，由於 `alice` 在 `tenant2` 中不是 `admin`，她無法讀取 `data2`。

接著，在匹配器中，你應該如下檢查角色：

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

請參考 `rbac_with_domains_model.conf` 中的範例。

① 令牌名稱慣例

注意：慣例上，策略定義中的域令牌名稱是 `dom`，並且放置為第二個令牌 (`sub, dom, obj, act`)。現在，Golang Casbin 支持自定義令牌名稱和位置。如果域令牌名稱是 `dom`，域令牌可以放置在任意位置而無需任何額外操作。如果域標記名稱不是 `dom`，則在強制器初始化後，無論其位置如何，都應該調用 `e.SetFieldIndex()` 以設置 `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

帶條件的角色管理器

`ConditionalRoleManager` 支援在策略層級使用自訂條件函數。

例如，當我們需要一個臨時的角色策略時，我們可以遵循以下方法：

`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 `(_, _)` 來包含傳遞給條件函數的參數列表，並使用 `_` 作為參數佔位符

`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, _
```

基本用法

透過 `AddNamedLinkConditionFunc` 為角色策略（`g` 類型策略）添加條件函數，如果檢查通過，則相應的角色策略（`g` 類型策略）有效，否則無效。

```
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",
util.TimeMatchFunc)
```

自定義條件函數

自定義條件函數需要符合以下函數類型

```
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",
endTime); err != nil {
            return false, err
        } else if now.After(end) {
            return false, nil
        }
    }
}
```

帶域的條件性角色管理器

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
```

基本用法

透過 `AddNamedDomainLinkConditionFunc` 為角色策略 (`g` 類型策略) 添加條件函數，如果檢查通過，則相應的角色策略 (`g` 類型策略) 有效，否則無效。

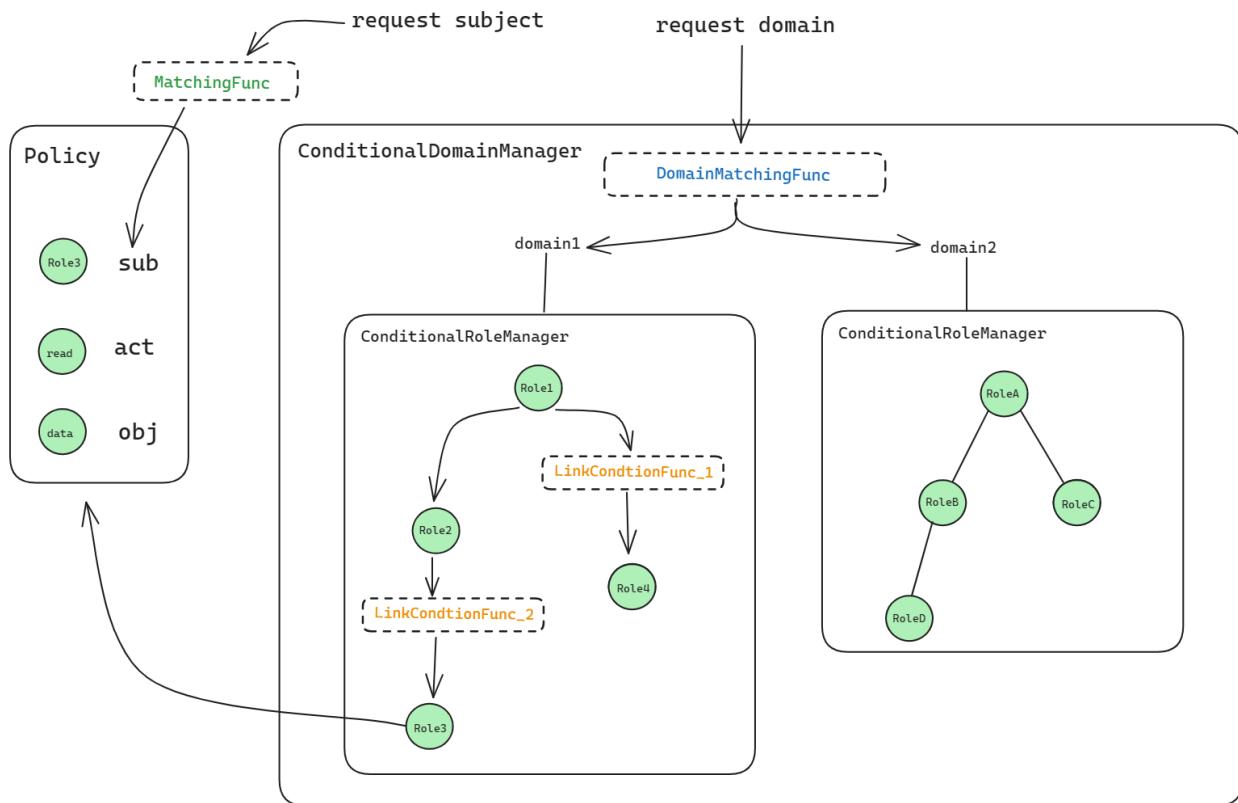
```
e.AddNamedDomainLinkConditionFunc("g", "alice", "data2_admin",
"domain2", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data3_admin",
"domain3", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "alice", "data4_admin",
"domain4", util.TimeMatchFunc)
e.AddNamedDomainLinkConditionFunc("g", "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
e.enforce("alice", "domain7", "data7", "read") //
```

自訂條件函數

與基本的 `Conditional RoleManager` 類似，支援自訂函數，使用上沒有差異。

請注意，`DomainMatchingFunc`、`MatchingFunc` 和 `LinkConditionFunc` 處於不同層級，並在不同情況下使用。



Casbin RBAC vs. RBAC96

Casbin RBAC 與 RBAC96

在本文件中，我們將比較 Casbin RBAC 與 [RBAC96](#)。

Casbin RBAC 支援幾乎所有 RBAC96 的功能，並在此基礎上增加了新功能。

RBAC 版本	支援等級	描述
RBAC0	完全支援	RBAC0 是 RBAC96 的基本版本。它明確了使用者、角色和權限之間的關係。
RBAC1	完全支援	RBAC1 在 RBAC0 的基礎上增加了角色層級。這意味著如果 <code>alice</code> 擁有 <code>role1</code> ，而 <code>role1</code> 擁有 <code>role2</code> ，那麼 <code>alice</code> 也將擁有 <code>role2</code> 並繼承其權限。
RBAC2	支援互斥處理 (如這裡所示)	RBAC2 在 RBAC0 的基礎上增加了約束條件。這使得 RBAC2 能夠處理相互排斥的政策。然而，不支援定量限制。
RBAC3	支援相互排斥處理 (如這裡)	RBAC3 是 RBAC1 和 RBAC2 的結合。它支援 RBAC1 和 RBAC2 中的角色階層和約束。然而，不支援定量限制。

Casbin RBAC 與 RBAC96 的區別

1. 在 Casbin 中，用戶和角色的區分不像在 RBAC96 中那麼明確。

在 Casbin 中，用戶和角色都被視為字串。舉例來說，考慮以下政策文件：

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

如果你使用 Casbin Enforcer 的實例調用 `GetAllSubjects()` 方法：

```
e.GetAllSubjects()
```

返回值將是：

```
[admin alice]
```

這是因為在 Casbin 中，主體包括用戶和角色。

然而，如果你調用 `GetAllRoles()` 方法：

```
e.GetAllRoles()
```

返回值將是：

```
[admin]
```

由此可見，Casbin 中用戶和角色之間有所區別，但不像 RBAC96 那樣明確。當然，你可以在政策中添加前綴，例如 `user::alice` 和 `role::admin`，以明確它們的關係。

2. Casbin RBAC 提供的權限比 RBAC96 更多。

RBAC96 僅定義了 7 種權限：讀取、寫入、附加、執行、信用、借記和查詢。

然而，在 Casbin 中，我們將權限視為字串。這使您能夠創建更符合您需求的權限。

3. Casbin 的 RBAC 支援域。

在 Casbin 中，您可以根據域進行授權。這項功能使您的訪問控制模型更加靈活。

ABAC

什麼是ABAC模型？

ABAC代表基於屬性的訪問控制。它允許你通過使用主體、對象或動作的屬性（屬性）而不是使用字符串值本身來控制訪問。你可能聽說過一種複雜的ABAC訪問控制語言，稱為XACML。而Casbin的ABAC則簡單得多。在Casbin的ABAC中，您可以使用結構體或類實例而不是字符串作為模型元素。

讓我們來看一個官方的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，我們使用 `r.obj.Owner` 而不是 `r.obj`。在 `Enforce()` 函數中傳遞的 `r.obj` 將是一個結構體或類實例，而不是一個字符串。Casbin將使用反射為您檢索該結構體或類中的 `obj` 成員變量。

以下是 `r.obj` 結構體或類的定義：

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

如果你想通過JSON傳遞參數給執行器，您需要啟用該功能，使用
`e.EnableAcceptJsonRequest(true)`。

例如：

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

data1Json := `{"Name": "data1", "Owner": "bob"}  
`  
  
ok, _ := e.Enforce("alice", data1Json, "read")
```

① 注意

啟用接受JSON參數的功能可能會導致性能下降1.1到1.5倍。

如何使用ABAC？

要使用ABAC，您需要做兩件事：

1. 在模型匹配器中指定屬性。
2. 將元素的結構體或類實例作為參數傳遞給Casbin的 `Enforce()` 函數。

🔥 危險

目前，只有像 `r.sub`、`r.obj`、`r.act` 等請求元素支持ABAC。您不能在像

`p.sub` 這樣的策略元素上使用它，因為在Casbin的策略中無法定義結構體或類。你不能在像 `p.sub` 這樣的策略元素上使用它，因為在Casbin的策略中無法定義結構體或類。

💡 提示

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

💡 提示

如果您需要在策略中使用與CSV分隔符衝突的逗號，可以通過用引號包圍語句來轉義它。例如，`"keyMatch("bob", r.sub.Role)"`不會被分割。例如，`"keyMatch("bob", r.sub.Role)"`不會被分割。

擴展模型以適應複雜和大量ABAC規則

上述ABAC模型的實現是其核心的簡單實現。然而，在許多情況下，授權系統需要複雜且大量的ABAC規則。為了滿足這一需求，建議在策略中添加規則，而不是在模型中。這可以通過引入 `eval()` 函數構造來實現。以下是一個示例：

這是用于定義ABAC模型的 `CONF` 文件的定義。

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

[policy_definition]
p = sub_rule, obj, act

[policy_effect]
```

在這個示例中，`p.sub_rule` 是一個包含在策略中使用的必要屬性的結構體或類（用戶定義類型）。

這是用于 `Enforcement` 的模型對應的策略。現在，您可以使用傳遞給 `eval()` 的對象實例作為參數來定義某些ABAC約束。

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

Priority Model

Casbin 支持加載具有優先級的政策。

加載具有隱含優先級的政策

這非常簡單：順序決定優先級；出現較早的政策具有較高的優先級。

model.conf:

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

加載具有明確優先級的政策

另請參見：[casbin#550](#)

較小的優先級值表示較高的優先級。如果優先級中存在非數字字符，它將被放置在最後而不是拋出錯誤。

① 令牌名稱約定

在策略定義中通常使用的優先級令牌名稱是 "priority"。要使用自定義的名稱，你需要調用 `e.SetFieldIndex()` 並重新加載策略（參見 [TestCustomizedFieldIndex](#) 上的完整示例）。

model.conf:

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

Golang 代碼示例:

```
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，明確的優先級僅支持 `AddPolicy` 和 `AddPolicies`。如果已經調用了 `UpdatePolicy`，你不應該更改優先級屬性。

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
```

請求:

```
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
```

根據角色和用戶層級加載具有優先級的策略

角色和用戶的繼承結構只能是多棵樹，而不是圖形。如果一個用戶擁有多個角色，必須確保該用戶在不同的樹中處於相同的層級。如果兩個角色處於相同的層級，則出現較早的策略（與角色相關聯）具有更高的優先級。更多詳情，請參閱 [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
```

請求:

```
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
```

角色層級如下所示：

```
role: root  
└ role: admin  
  └ role editor  
    └ user: jane  
    |  
    └ role: subscriber  
      └ user: alice
```

優先順序自動顯示如下：

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

Super Admin

超級管理員是整個系統的管理者。 它可以用於RBAC、ABAC以及帶有域的RBAC等模型。 詳細的例子如下：

```
[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"
```

這個例子說明了，通過定義的 `request_definition`、`policy_definition`、`policy_effect` 和 `matchers`，Casbin判斷請求是否能匹配策略。一個重要的方面是檢查 `sub` 是否為根。如果判斷正確，則授予授權，用戶有權執行所有操作。

類似於Linux系統中的root用戶，被授權為root可獲得對所有文件和設置的訪問權限。如果我們希望一個 `sub` 能夠完全訪問整個系統，我們可以將其角色分配為超級管理員，授予該 `sub` 執行所有操作的權限。



>

儲存

儲存

**Model Storage**

模型儲存

**Policy Storage**

政策儲存

**Policy Subset Loading**

載入過濾後的政策

Model Storage

與策略不同，模型只能被載入，無法被保存。我們認為模型不是一個動態組件，不應在運行時被修改，因此我們沒有實現將模型保存到儲存的API。

然而，有好消息。我們提供了三種等效的方式來載入模型，無論是靜態還是動態：

從.CONF文件載入模型

這是使用Casbin最常見的方式。對初學者來說容易理解，當你需要Casbin團隊的幫助時也很方便分享。

.CONF 檔案 `examples/rbac_model.conf` 的內容如下：

```
[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
```

然後你可以如下載入模型檔案：

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

從程式碼載入模型

模型可以從程式碼動態初始化，而不是使用 `.CONF` 檔案。以下是 RBAC 模型的範例：

```
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)
```

從字串載入模型

或者，你可以從多行字串載入整個模型文字。這種方法的優點是你不需要維護一個模型檔案。

```
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，政策儲存是以 [適配器](#) 的形式實現的。

從 .CSV 文件載入政策

這是使用 Casbin 最常見的方式。 對初學者來說容易理解，並且在你向 Casbin 團隊尋求幫助時方便共享。

.csv 文件 [examples/rbac_policy.csv](#) 的內容如下：

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

ⓘ 注意

如果你的文件包含逗號，你應該將它們用雙引號包裹。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)
```

如果你的檔案包含逗號和雙引號，你應該將欄位用雙引號括起來，並將任何嵌入的雙引號加倍。

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

相關問題: [casbin#886](#)

適配器 API

方法	類型	描述
LoadPolicy()	基本	從存儲中加載所有策略規則
SavePolicy()	基本	將所有政策規則保存到儲存空間
添加政策()	可選	向儲存空間添加一個政策規則
移除政策()	可選	從儲存空間移除一個政策規則
移除過濾後的政策()	可選	從儲存中移除符合篩選條件的政策規則

資料庫儲存格式

您的政策檔案

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

對應的資料庫結構（例如 MySQL）

識別碼	政策類型	值0	值1	值2	值3	v4	v5
1	p	資料2_管理員	資料2	讀取			
2	p	資料2_管理員	data2	寫入			
3	g	愛麗絲	管理員				

各欄位的意義

- `id`: 資料庫中的主鍵。它不作為 `casbin` 政策 的一部分存在。其生成方式取決於特定的適配器。
- `ptype`: 它對應於 `p`, `g`, `g2` 等。
- `v0-v5`: 這些列名沒有特定含義，並且對應於 `policy csv` 中從左到右的值。列的數量取決於您自己定義多少。理論上，可以有無限多的列，但通常在適配器中只實現了 6 列。如果這對您來說不夠，請向相應的適配器倉庫提交一個問題。

適配器詳情

有關適配器 API 的使用和數據庫表結構設計的更多詳細信息，請訪問：[/docs/adapters](#)

Policy Subset Loading

某些適配器支援過濾後的政策管理。這意味著Casbin載入的政策是根據給定過濾器從數據庫中存儲的政策子集。這使得在大型、多租戶環境中，解析整個政策成為性能瓶頸的情況下，能夠有效地執行政策。

要使用支援的適配器與過濾後的政策，只需調用 `LoadFilteredPolicy` 方法。過濾器參數的有效格式取決於所使用的適配器。為了防止意外的數據丟失，當載入過濾後的政策時，`SavePolicy` 方法被禁用。

例如，以下代碼片段使用內建的過濾文件適配器和帶有域的RBAC模型。在這種情況下，過濾器將策略限制在單一網域。任何針對 `"domain1"` 以外網域的策略行都會從已載入的策略中被省略：

```
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"},
}
```

還有另一種方法支援子集載入功能：`LoadIncrementalFilteredPolicy`。

`LoadIncrementalFilteredPolicy` 與 `LoadFilteredPolicy` 相似，但它不會清除先前載入的策略。它僅將過濾後的策略附加到現有策略中。



>

場景

場景



Data Permissions

資料權限解決方案



Menu Permissions

菜單權限範例

Data Permissions

我們有兩種解決資料權限（過濾）的方案：使用隱式分配API或使用 `BatchEnforce()` API。

1. 查詢隱式角色或權限

當用戶透過RBAC層次結構繼承角色或權限，而不是在策略規則中直接分配給他們時，我們將這種類型的分配稱為“隱式”。要查詢這種隱式關係，您需要使用以下兩個API：
`GetImplicitRolesForUser()` 和 `GetImplicitPermissionsForUser()`，而不是
`GetRolesForUser()` 和 `GetPermissionsForUser()`。更多詳情，請參考[這個GitHub問題](#)。

2. 使用 `BatchEnforce()`

`BatchEnforce()` 會對每個請求進行強制執行，並以布林陣列的形式返回結果。

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

我們首先介紹一個包含菜單系統的Spring Boot範例。此範例利用jCasbin來管理菜單權限。最終目標是抽象出一個中間件，專門用於菜單權限，該中間件可以擴展到Casbin支持的其他語言，如Go和Python。

1. 配置文件

您需要在 `policy.csv` 文件中設置角色和權限管理，以及菜單項之間的父子關係。更多詳情，請參考[此GitHub倉庫](#)。

1.1 概述

使用 `policy.csv`，您可以靈活配置角色權限和菜單結構，以實現細粒度的訪問控制。此配置文件定義了不同角色對各種菜單項的訪問權限、用戶與角色之間的關聯以及菜單項之間的層次關係。

1.2 權限定義（策略）

- **策略規則：**策略以 `p` 前綴定義，指定角色 (`sub`) 及其在菜單項 (`obj`) 上的權限 (`act`)，以及規則的效果 (`efc`)，其中 `allow` 表示允許，`deny` 表示拒絕。

示例：

- `p, ROLE_ROOT, SystemMenu, read, allow` 表示 `ROLE_ROOT` 角色對 `SystemMenu` 菜單項具有讀取權限。
- `p, ROLE_ROOT, UserMenu, read, deny` 表示 `ROLE_ROOT` 角色對 `UserMenu` 菜單項的讀取權限被拒絕。

1.3 角色與用戶關聯

- **角色繼承：**用戶-角色關係和角色層次結構以 `g` 前綴定義。這允許用戶從一個或多個角色繼承權限。

範例：

- `g, user, ROLE_USER` 表示用戶 `user` 被分配了 `ROLE_USER` 角色。
- `g, ROLE_ADMIN, ROLE_USER` 表示 `ROLE_ADMIN` 從 `ROLE_USER` 繼承權限。

1.4 菜單項目層次

- **菜單關係：**菜單項目之間的父子關係使用 `g2` 前綴定義，有助於構建菜單的結構。

範例：

- `g2, UserSubMenu_allow, UserMenu` 表示 `UserSubMenu_allow` 是 `UserMenu` 的子菜單。
- `g2, (NULL), SystemMenu` 表示 `SystemMenu` 沒有子菜單項目，意味著它是一個頂級菜單項目。

1.5 菜單權限繼承和默認規則

在使用jCasbin管理菜單權限時，父菜單與子菜單之間的權限關係遵循特定的繼承規則，其中有兩個重要的默認規則：

父菜單權限的繼承：

如果一個父菜單被明確授予 `allow` 權限，則其所有子菜單也默認為 `allow` 權限，除非被特別標記為 `deny`。這意味著一旦一個父菜單可訪問，其子菜單也默認可訪問。

處理無直接權限設置的父菜單：

如果一個父菜單沒有直接的權限設置（既未明確允許也未拒絕）但至少有一個子菜單被明確授予允許權限，則該父菜單被隱含地視為具有允許權限。這確保用戶可以導航到這些子菜單。

1.6 特殊權限繼承規則

關於角色間權限的繼承，尤其是在涉及拒絕權限的情況下，必須遵循以下規則以確保系統安全和精確的權限控制：

明確拒絕與默認拒絕的區分：

如果某個角色，例如 `ROLE_ADMIN`，被明確拒絕訪問某個選單項目，例如 `AdminSubMenu_deny`（標記為 `deny`），那麼即使這個角色被另一個角色（例如 `ROLE_ROOT`）繼承，繼承的角色也不允許訪問被拒絕的選單項目。這確保了明確的安全政策不會因為角色繼承而被繞過。

默認拒絕權限的繼承：

相反地，如果某個角色對某個選單項目的訪問被拒絕（例如 `UserSubMenu_deny`）是默認的（未明確標記為 `deny`，而是因為未明確授予 `allow`），那麼當這個角色被另一個角色（例如 `ROLE_ADMIN`）繼承時，繼承的角色可能會覆蓋默認的 `deny` 狀態，允許訪問這些選單項目。

1.7 示例描述

政策：

```
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
```

選單名稱	ROLE_ROOT	ROLE_ADMIN	ROLE_USER
SystemMenu	✓	✗	✗
UserMenu	✗	✓	✗
UserSubMenu_allow	✗	✓	✓
UserSubSubMenu	✗	✓	✓
UserSubMenu_deny	✗	✓	✗
AdminMenu	✓	✓	✗
AdminSubMenu_allow	✓	✓	✗
AdminSubSubMenu_deny	✓	✗	✗

2. 選單權限控制

透過[MenuService](#)中的 `findAccessibleMenus()` 函數，可以識別特定用戶名可訪問的所有選單項目。要檢查特定用戶是否有權訪問指定的選單項目，可以使用 `checkMenuAccess()` 方法。此方法確保選單權限得到有效控制，利用jCasbin的能力高效管理訪問權限。



>

擴充功能

擴充功能



Enforcers

「執行者」是Casbin中的主要結構，作為用戶對政策規則和模型進行操作的接口。



Adapters

支援的轉接器及其使用



Watchers

維持多個 Casbin 執行器實例之間的一致性



Dispatchers

調度器提供了一種同步政策增量變更的方法。

Role Managers

角色管理員用於管理Casbin中的RBAC角色層級。

Middlewares

Casbin 中介軟體

GraphQL Middlewares

GraphQL 端點的授權

Cloud Native Middlewares

雲原生中間件

Enforcers

執行者 是Casbin中的主要結構。 它作為用戶對政策規則和模型進行操作的接口。

支援的執行者

以下提供Casbin執行者的完整列表。 歡迎任何第三方對新執行者的貢獻。 請通知我們，我們將其添加到此列表中 :)

[Go](#) [Python](#)

執行者	作者	描述
執行者	Casbin	執行者 是使用者與 Casbin 政策和模型互動的基本結構。 你可以在 這裡 找到更多關於 執行者 API 的詳細資訊。
快取執行者	Casbin	快取執行者 基於 執行者，並支援使用映射在記憶體中快取請求的評估結果。 它提供了在指定過期時間內清除快取的能力。 此外，它通過讀寫鎖保證了線程安全。 您可以使用 <code>EnableCache</code> 來啟用評估結果的緩存（默認已啟用）。 <code>CachedEnforcer</code> 的其他 API 方法與 <code>Enforcer</code> 相同。
DistributedEnforcer	Casbin	<code>DistributedEnforcer</code> 支持分布式集群中的多個實例。 它將 <code>SyncedEnforcer</code> 包裝

執行者	作者	描述
		用於調度器。 您可以在 這裡 找到更多關於調度器的詳細信息。
SyncedEnforcer	Casbin	SyncedEnforcer 是基於 Enforcer 並提供同步存取的機制。 它是執行緒安全的。
SyncedCachedEnforcer	Casbin	SyncedCachedEnforcer 封裝了 Enforcer 並提供決策同步快取。
Enforcer	作者	描述
Enforcer	Casbin	Enforcer 是使用者與 Casbin 政策和模型互動的基本結構。 您可以在 這裡 找到更多關於 Enforcer API 的詳細資訊。
DistributedEnforcer	Casbin	DistributedEnforcer 支援分散式叢集中的多個實例。 它封裝了 SyncedEnforcer 以供調度器使用。 您可以在 這裡 找到更多關於調度器的詳細資訊。
SyncedEnforcer	Casbin	SyncedEnforcer 基於 Enforcer 並提供同步存取功能。 它是執行緒安全的。
AsyncEnforcer	Casbin	AsyncEnforcer 提供非同步 API。
FastEnforcer	Casbin	FastEnforcer 使用一個新的模型，速度比普通模型快 50 倍。 你可以在 這裡 找到更多資訊。

Adapters

在 Casbin 中，策略存儲被實現為一個轉接器（即 Casbin 的中間件）。Casbin 用戶可以使用轉接器從存儲中加載策略規則（即 `LoadPolicy()`），或將策略規則保存到其中（即 `SavePolicy()`）。為了保持輕量級，我們不在主庫中包含轉接器代碼。

支援的轉接器

以下提供了 Casbin 轉接器的完整列表。歡迎任何第三方對新轉接器的貢獻，請通知我們，我們將在此列表中添加它：

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

Adapter	Type	Author	AutoSave	Description
File Adapter (built-in)	File	Casbin	✗	For <code>.CSV (Comma-Separated Values)</code> files
Filtered File Adapter (built-in)	File	@faceless-saint	✗	For <code>.CSV (Comma-Separated Values)</code> files with policy subset loading support
SQL Adapter	SQL	@Blank-Xu	✓	MySQL, PostgreSQL, SQL Server, SQLite3 are supported in <code>master</code> branch and Oracle is supported in <code>oracle</code> branch by <code>database/sql</code>
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 <code>master</code> branch and Oracle is supported in <code>oracle</code> 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
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
Dynacasbin	NoSQL	NewbMiao	✓	For Amazon DynamoDB

Adapter	Type	Author	AutoSave	Description
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
Redis Adapter	KV store	@mlsen	✓	For Redis

Adapter	Type	Author	AutoSave	Description
(go-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
JDBC	JDBC	Casbin	✓	MySQL, Oracle, PostgreSQL, DB2, Sybase,

Adapter	Type	Author	AutoSave	Description	
Adapter				SQL Server are supported by JDBC	
Hibernate Adapter	ORM	Casbin	<input checked="" type="checkbox"/>	Oracle, DB2, SQL Server, Sybase, MySQL, PostgreSQL are supported by Hibernate	
MyBatis Adapter	ORM	Casbin	<input checked="" type="checkbox"/>	MySQL, Oracle, PostgreSQL, DB2, Sybase, SQL Server (the same as JDBC) are supported by MyBatis 3	
Hutool Adapter	ORM	@mapleafgo	<input checked="" type="checkbox"/>	MySQL, Oracle, PostgreSQL, SQLite are supported by Hutool	
MongoDB Adapter	NoSQL	Casbin	<input checked="" type="checkbox"/>	MongoDB is supported by mongodb-driver-sync	
DynamoDB Adapter	NoSQL	Casbin	<input type="checkbox"/>	For Amazon DynamoDB	
Redis Adapter	KV store	Casbin	<input checked="" type="checkbox"/>	For Redis	
Adapter	Type	Author		AutoSave	Description
File Adapter (built-in)	File	Casbin		<input type="checkbox"/>	For .CSV (Comma-Separated Values) files
Filtered File Adapter (built-in)	File	Casbin		<input type="checkbox"/>	For .CSV (Comma-Separated Values) files with policy subset loading support
String Adapter (built-in)	String	@calebfaruki		<input type="checkbox"/>	For String
Basic Adapter	Native ORM	Casbin		<input checked="" type="checkbox"/>	pg, mysql, mysql2, sqlite3, oracledb, mssql are

Adapter	Type	Author	AutoSave	Description
				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 Native	SQL	@touchifyapp	✓	PostgreSQL adapter with advanced policy subset loading support and

Adapter	Type	Author		AutoSave	Description
Adapter					improved performances built with node-postgres .
Mongoose Adapter	NoSQL	elastic.io and Casbin		<input checked="" type="checkbox"/>	MongoDB is supported by Mongoose
Mongoose Adapter (No-Transaction)	NoSQL	minhducck		<input checked="" type="checkbox"/>	MongoDB is supported by Mongoose
Node MongoDB Native Adapter	NoSQL	@juicyleff		<input checked="" type="checkbox"/>	For Node MongoDB Native
DynamoDB Adapter	NoSQL	@fospitia		<input checked="" type="checkbox"/>	For Amazon DynamoDB
Couchbase Adapter	NoSQL	@MarkMYoung		<input checked="" type="checkbox"/>	For Couchbase
Redis Adapter	KV store	Casbin		<input type="checkbox"/>	For Redis
Redis Adapter	KV store	@NandaKishorJeripothula		<input type="checkbox"/>	For Redis
Adapter	Type	Author	AutoSave	Description	
File Adapter (built-in)	File	Casbin	<input type="checkbox"/>	For .CSV (Comma-Separated Values) files	
Database Adapter	ORM	Casbin	<input checked="" type="checkbox"/>	MySQL, PostgreSQL, SQLite, Microsoft SQL Server are supported by techone/database	

Adapter	Type	Author	AutoSave	Description
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

Adapter	Type	Author	AutoSave	Description
				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, Microsoft SQL Server, Firebird,

Adapter	Type	Author	AutoSave	Description
				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
MongoEngine Adapter	ORM	@zhangbailong945	✗	MongoDB is supported by MongoEngine

Adapter	Type	Author	AutoSave	Description
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 <code>remove_filtered_policy()</code>)	For Couchbase
DynamoDB Adapter	NoSQL	@abqadeer	✓	For DynamoDB

Adapter		Type	Author		AutoSave		Description
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 Cosmos DB Adapter	Cloud	@sagarkhandelwal			For Microsoft Azure Cosmos DB		

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
JSON Adapter	String	Casbin	✓	For JSON
YAML Adapter	String	Casbin	✓	For YAML

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 (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	
LuaSQL Adapter	ORM	Casbin	✓	MySQL, PostgreSQL, SQLite3 are supported by LuaSQL	
4DaysORM Adapter	ORM	Casbin	✓	MySQL, SQLite3 are supported by 4DaysORM	

Adapter	Type	Author	AutoSave	Description
OpenResty Adapter	ORM	@tom2nonames	<input checked="" type="checkbox"/>	MySQL, PostgreSQL are supported by it

ⓘ 注意

1. 如果使用明確或隱含的轉接器調用 `casbin.NewEnforcer()`, 策略將自動加載。
2. 您可以呼叫 `e.LoadPolicy()` 從儲存中重新載入策略規則。
3. 如果適配器不支援 `自動保存` 功能, 當您新增或移除策略時, 策略規則無法自動保存回儲存中。您必須手動呼叫 `SavePolicy()` 來保存所有策略規則。

範例

在此，我們提供幾個範例：

檔案適配器（內建）

以下展示如何從內建的檔案適配器初始化一個執行器：

Go PHP Rust

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

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

use Casbin\Enforcer;

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

use casbin::prelude::*;

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

這與以下相同：

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

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

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

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

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

use casbin::prelude::*;

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

MySQL 適配器

以下展示如何從 MySQL 資料庫初始化一個執行器。它連接到位於127.0.0.1:3306的MySQL資料庫，使用root用戶和空白密碼。

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

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

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

```

// https://github.com/casbin-rs/diesel-adapter
// 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?;

// 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_sqlIsrv,
    // mysqli,sqlanywhere,sqlsrv,ibm_db2 (unstable),drizzle_pdo_mysql
    'driver'      => 'pdo_mysql',
    'host'        => '127.0.0.1',
    'dbname'      => 'test',
    'user'        => 'root',
    'password'    => '',
    'port'        => '3306',
];

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

```

使用您自己的存儲適配器

您可以像下面這樣使用您自己的適配器：

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

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

在不同適配器之間遷移/轉換

如果您想將適配器從 A 轉換為 B，可以這樣操作：

1. 從A將策略加載到內存中

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

或者

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

2. 將您的適配器從A轉換為B

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

3. 從內存中將策略保存到B

```
e.SavePolicy()
```

在運行時加載/保存

您可能還希望在初始化後重新載入模型、重新載入策略或保存策略：

```
// Reload the model from the model CONF file.
```

自動保存

有一個稱為「自動保存」的功能適用於適配器。當一個適配器支援「自動保存」時，這意味著它可以支援將單一策略規則添加到存儲中，或從存儲中移除單一策略規則。這與「保存策略」不同，因為後者將刪除存儲中的所有策略規則，並將Casbin執行器中的所有策略規則保存到存儲中。因此，當策略規則數量龐大時，可能會遇到性能問題。

當適配器支援「自動保存」時，您可以通過「Enforcer.EnableAutoSave()」函數切換此選項。該選項默認是開啟的（如果適配器支援的話）。

① 注意

1. 「自動保存」功能是可選的。適配器可以選擇實現與否。
2. 「自動保存」功能僅在 Casbin 執行器所使用的適配器支持時才有效。
3. 請參閱上述適配器列表中的「自動保存」欄，以確認某適配器是否支持「自動保存」功能。

以下是如何使用「自動保存」的範例：

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

// By default, the AutoSave option is enabled for an enforcer.
a := xormadapter.NewAdapter("mysql",
    "mysql_username:mysql_password@tcp(127.0.0.1:3306)/")
e := casbin.NewEnforcer("examples/basic_model.conf", a)

// Disable the AutoSave option.
e.EnableAutoSave(false)

// Because AutoSave is disabled, the policy change only affects the policy in
// Casbin enforcer,
// it doesn't affect the policy in the storage.
e.AddPolicy(...)
e.RemovePolicy(...)

// Enable the AutoSave option.
e.EnableAutoSave(true)
```

更多範例請參閱：https://github.com/casbin/xorm-adapter/blob/master/adapter_test.go

如何編寫一個適配器

所有適配器都應實現 `Adapter` 接口，至少提供兩個必要方法：`LoadPolicy(model model.Model) error` 和 `SavePolicy(model model.Model) error`。

其他三個功能是可選的。如果適配器支持 `自動保存` 功能，則應實現這些功能。

方法	類型	描述
載入政策()	強制性	從儲存中載入所有政策規則
保存政策()	強制性	將所有政策規則保存到儲存中
添加政策()	可選	將一個政策規則添加到儲存中
移除政策()	可選的	從儲存中移除一個政策規則
移除篩選後的政策()	可選的	從儲存中移除符合篩選條件的政策規則

① 注意

如果一個適配器不支援「自動保存」，它應該為這三個可選函數提供一個空的實現。這裡有一個 Golang 的例子：

```
// 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
int, fieldValues ...string) error {
    return errors.New("not implemented")
```

Casbin 執行器在調用這三個可選函數時會忽略「未實現」的錯誤。

關於如何編寫適配器的詳細說明。

- 資料結構。 轉接器應支援讀取至少六個欄位。
- 資料庫名稱。 預設的資料庫名稱應為 `casbin`。
- 表格名稱。 預設的表格名稱應為 `casbin_rule`。
- Ptype 欄位。 此欄位的名稱應為 `ptype` 而非 `p_type` 或 `Ptype`。
- 表格定義應為 `(id int primary key, ptype varchar, v0 varchar, v1 varchar, v2 varchar, v3 varchar, v4 varchar, v5 varchar)`。
- 應在欄位 `ptype, v0, v1, v2, v3, v4, v5` 上建立唯一鍵索引。
- `LoadFilteredPolicy` 需要一個 `filter` 作為參數。 過濾器應該類似這樣。

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

誰負責創建資料庫？

按照慣例，適配器應該能夠自動創建一個名為 `casbin` 的資料庫（如果它不存在），並使用它來存儲策略。
請參考 Xorm 適配器的實現作為範例：<https://github.com/casbin/xorm-adapter>

上下文適配器

`ContextAdapter` 提供了對 Casbin 適配器的上下文感知接口。

通過上下文，你可以實現適配器 API 的超時控制等功能。

範例

`gormadapter` 支持帶上下文的適配器，以下是使用上下文實現的超時控制範例。

```
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()
```

如何撰寫一個上下文適配器

`ContextAdapter` API 僅比普通的 `Adapter` API 多了一層上下文處理,

`gormadapter` 的簡單參考: [adapter.go](#)

Watchers

我們支持使用像 [etcd](#) 這樣的分佈式消息系統來維持多個 Casbin 執行實例之間的一致性。這使得我們的使用者能夠同時使用多個 Casbin 執行器來處理大量的權限檢查請求。

類似於政策存儲適配器，我們不在主庫中包含監視器代碼。任何對新訊息系統的支援應該以監視器的方式實施。以下提供完整的Casbin觀察者名單。我們歡迎任何第三方對於新監視器的貢獻，請告知我們，我們將會將其加入此列表：

[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 system	Gepsr	Watcher for Apache ZooKeeper

Watcher	Type	Author	Description
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
Lettuce-Based Redis Watcher	KV store	Casbin	Watcher for Redis based on Lettuce)
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

Watcher		Type		Author	Description
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

為了支援多個實例之間的增量同步，我們提供了 `WatcherEx` 介面。我們希望當政策變更時，它能通知其他實例，但目前尚未實現 `WatcherEx`。我們建議您使用調度器來達成這一點。

相較於 `Watcher` 介面，`WatcherEx` 能夠區分接收到的更新操作類型，例如 `AddPolicy` 和 `RemovePolicy`。

觀察者Ex 蜜蜂：

API 在傳統中文中通常保持不變，因為它是一個技術術語，廣泛被接受和使用。所以，API 的翻譯仍然是「API」。	描述
設定更新回調函數 (<code>func(字串)</code>) 錯誤	<code>SetUpdateCallback</code> 設定了一個回調函數，當資料庫中的策略被其他實例更改時，監視器將會調用這個函數。 經典的回呼是 <code>Enforcer.LoadPolicy()</code> 。
更新() 錯誤	更新調用其他實例的更新回調，以同步它們的策略。通常在資料庫（DB）中更改政策後會進行呼叫，例如 <code>Enforcer.SavePolicy()</code> 、 <code>Enforcer.AddPolicy()</code> 、 <code>Enforcer.RemovePolicy()</code> 等等。
關閉()	關閉停止並釋放監視器，回調函數將不再被調用。
<code>UpdateForAddPolicy(sec string, ptype string, params ...string) error</code>	<code>UpdateForAddPolicy</code> 會呼叫其他實例的更新回調，以同步它們的策略。這是在透過 <code>Enforcer.AddPolicy()</code> 、 <code>Enforcer.AddNamedPolicy()</code> 、 <code>Enforcer.AddGroupingPolicy()</code> 和 <code>Enforcer.AddNamedGroupingPolicy()</code> 添加政策後被稱為的。
<code>UpdateForRemovePolicy(sec string, ptype string, params ...string) error</code>	<code>UpdateForRemovePolicy</code> 會呼叫其他實例的更新回調，以同步它們的策略。在執行者（ <code>Enforcer</code> ）移除

API 在傳統中文中通常保持不變，因為它是一個技術術語，廣泛被接受和使用。所以，API 的翻譯仍然是「API」。	描述
	政策後，稱之為：Enforcer.RemovePolicy()、Enforcer.RemoveNamedPolicy()、Enforcer.RemoveGroupingPolicy() 和 Enforcer.RemoveNamedGroupingPolicy()。
UpdateForRemoveFilteredPolicy(sec, ptype string, fieldIndex int, fieldValues ...string) error	UpdateForRemoveFilteredPolicy 會呼叫其他實例的更新回調，以同步它們的策略。這些方法分別被稱為 Enforcer.RemoveFilteredPolicy()、Enforcer.RemoveFilteredNamedPolicy()、Enforcer.RemoveFilteredGroupingPolicy() 和 Enforcer.RemoveFilteredNamedGroupingPolicy()。
更新保存策略的函數 (UpdateForSavePolicy) 接受一個模型 (model.Model) 作為參數，並返回一個錯誤 (error)。	UpdateForSavePolicy 會呼叫其他實例的更新回調，以同步它們的策略。這被稱為 Enforcer.SavePolicy()。
UpdateForAddPolicies(sec string, ptype string, rules ...[]string) error 的翻譯如下：更新添加政策(sec字串, ptype字串, 規則...[]字串) 錯誤	UpdateForAddPolicies 會呼叫其他實例的更新回調，以同步它們的策略。這些方法分別被稱為：Enforcer.AddPolicies()、Enforcer.AddNamedPolicies()、Enforcer.AddGroupingPolicies() 和 Enforcer.AddNamedGroupingPolicies()。
UpdateForRemovePolicies(sec string, ptype string, rules ...[]string) error	UpdateForRemovePolicies 會呼叫其他實例的更新回調，以同步它們的策略。這些方法分別被稱為 Enforcer.RemovePolicies()、Enforcer.RemoveNamedPolicies()、Enforcer.RemoveGroupingPolicies() 和 Enforcer.RemoveNamedGroupingPolicies()。

Dispatchers

調度器提供了一種同步政策增量變更的方法。它們應基於一致性算法，如Raft，以確保所有執行器實例的一致性。透過調度器，用戶可以輕鬆建立分散式集群。

調度器的方法分為兩部分。第一部分是與Casbin結合的方法。這些方法應在Casbin內部被調用。用戶可以使用Casbin本身提供的更完整的API。

另一部分是調度器本身定義的方法，包括調度器初始化方法，以及不同算法提供的不同功能，如動態成員資格和配置變更。

ⓘ 注意

我們希望調度器僅在運行時確保Casbin執行器的一致性。因此，如果在初始化期間策略不一致，調度器將無法正常工作。用戶需要在使用調度器之前確保所有實例的狀態一致。

以下提供了Casbin調度器的完整列表。歡迎任何第三方對新調度器的貢獻。請通知我們，我們將其添加到此列表中。

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

分佈式執行器

分佈式執行器封裝了用於調度器的同步執行器。

Go

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

Role Managers

角色管理員用於管理Casbin中的RBAC角色層級（用戶-角色映射）。 角色管理員可以從Casbin策略規則或外部來源（如LDAP、Okta、Auth0、Azure AD等）檢索角色數據。我們支持不同實現的角色管理員。為保持輕量級，我們不在主庫中包含角色管理員代碼（除了默認角色管理員）。以下提供完整的Casbin角色管理員列表。歡迎任何第三方為新角色管理員做出貢獻。請通知我們，我們將其添加到此列表中：）

[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

對開發者而言：所有角色管理器必須實現 [RoleManager](#) 接口。[Session Role Manager](#) 可作為參考實現。

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

對開發者而言：所有角色管理器必須實現 [RoleManager](#) 接口。 [Default Role Manager](#) 可作為參考實現。

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

對開發者而言：所有角色管理器必須實現 [RoleManager](#) 接口。 [Default Role Manager](#) 可作為參考實現。

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

對開發者而言：所有角色管理器必須實現 [RoleManager](#) 接口。 [Default Role Manager](#) 可作為參考實現。

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

對開發者而言：所有角色管理器必須實現 [RoleManager](#) 接口。[Default Role Manager](#) 可作為參考實現。

API

詳細資訊請參閱[API](#)部分。

Middlewares

網頁框架

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

[Swift](#)

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
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
Macaron	A high productive and modular web framework in Go, via plugin: authz

Name	Description
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 framework, via plugin: jfinal-authz
Nutz	Web framework (MVC/IOC/AOP/DAO/JSON) for all Java developers,

Name	Description
	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
Fastify	Fast and low overhead web framework, for Node.js. via plugin: fastify-casbin or fastify-casbin-rest

Name	Description
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
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
ThinkPHP 5.1	The ThinkPHP 5.1 framework, via plugin: think-casbin
ThinkPHP 6.0	The ThinkPHP 6.0 framework, via plugin: think-authz

Name	Description
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-authorization
Flask	A microframework for Python based on Werkzeug, Jinja 2 and good intentions, via plugin: flask-authz or Flask-Casbin (3rd-

Name	Description
	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-authz 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
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
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

雲端服務提供商

[Node.js](#)

名稱	描述
Okta	一個值得信賴的平台，用於保護每個身份，通過插件： casbin-spring-boot-demo
Auth0	一個易於實施、可調整的認證和授權平台，通過插件： casbin-auth0-rbac



GraphQL Middlewares

Casbin 遵循官方建議的方式，通過擁有一個授權的單一真實來源，為 GraphQL 端點提供授權：<https://graphql.org/learn/authorization/>。換句話說，Casbin 應該放置在 GraphQL 層和您的業務邏輯之間。

```
// 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);
      }
    }
  }
});
```

支援的 GraphQL 中介層

以下提供 Casbin GraphQL 中介層的完整列表。歡迎任何第三方對新的 GraphQL 中介層的貢獻。請通知我們，我們將會將其添加到此列表中:)

Go

Node.js

Python

中介層	GraphQL 實作	作者	描述
graphql-authz	graphql	Casbin	一個用於 graphql-go 的授權中介軟體
graphql-casbin	graphql	@esmaeilpour	使用 GraphQL 和 Casbin 結合的實現
gqlgen_casbin_RBAC_範例	gqlgen	@WenyXu	(空白)
中間件	GraphQL 實現	作者	描述
graphql-authz	GraphQL.js	Casbin	一個適用於 GraphQL.js 的 Casbin 授權中介軟體
中介軟體	GraphQL 實作	作者	描述
graphql-authz	GraphQL-core 3	@Checho3388	適用於 GraphQL-core 3 的 Casbin 授權中介軟體

Cloud Native Middlewares

雲原生項目

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

項目	作者	描述
k8s-authz	Casbin	Kubernetes 的授權中間件
envoy-authz	Casbin	適用於 Istio 和 Envoy 的授權中介軟體
kubesphere-authz	Casbin	適用於 kubeSphere 的授權中介軟體

專案	作者	描述
ODPF Shield	開放數據平台	ODPF Shield 是一個雲原生基於角色的授權感知反向代理服務。

API

API Overview

Casbin API 使用

Management API

這個基礎API提供了對Casbin策略管理的全面支援。

RBAC API

一個更友好的RBAC API。 此API是管理API的子集。 RBAC用戶可以使用此API來簡化代碼。

RBAC with Domains API

一個更用戶友好的RBAC帶域API。 此API是管理API的一個子集。 RBAC用戶可以使用此API來簡化他們的代碼。

RBAC with Conditions API

一個更用戶友好的帶條件RBAC API。

RoleManager API

RoleManager API 提供了一個介面，用於定義管理角色的操作。在 RoleManager 中增加匹配函數，允許在角色名稱和域中使用萬用字元。

API Overview

本概覽僅向您展示如何使用 Casbin API，並不會解釋 Casbin 的安裝方式或其工作原理。您可以在以下鏈接找到相關教程：[Casbin 安裝](#) 和 [Casbin 工作原理](#)。因此，當您開始閱讀本教程時，我們假設您已經完全安裝並將 Casbin 導入到您的代碼中。

強制執行 API

讓我們從 Casbin 的強制執行 API 開始。我們將從 `model.conf` 加載 RBAC 模型，並從 `policy.csv` 加載策略。您可以在[這裡](#)了解模型語法，我們在本教程中不會討論這一點。我們假設您能理解以下給出的配置文件：

`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
```

閱讀完配置文件後，請閱讀以下代碼。

```
// 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")
```

此代碼從本地文件加載訪問控制模型和策略。函數 `casbin.NewEnforcer()` 將返回一個執行器。它會將其兩個參數識別為文件路徑並從那裡加載文件。過程中發生的錯誤存儲在變量 `err` 中。此代碼使用默認適配器來加載模型和策略，當然，您也可以通過使用第三方適配器來達到相同的效果。

代碼 `ok, err := enforcer.Enforce("alice", "data1", "read")` 用於確認訪問權限。如果 Alice 能夠使用讀取操作存取 data1，則返回的 `ok` 值將為 `true`；否則，它將為 `false`。在此範例中，`ok` 的值為 `true`。

EnforceEx API

有時你可能想知道是哪個政策允許了請求，因此我們準備了 `EnforceEx()` 函數。你可以這樣使用它：

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

`EnforceEx()` 函數將在返回值 `reason` 中返回確切的政策字符串。在此範例中，`amber` 是一個 `admin` 角色，因此政策 `p, admin, data1, read` 允許此請求為 `true`。此代碼的輸出在註釋中。

Casbin 提供了許多類似於此的 API。這些 API 為基本功能添加了一些額外的功能。它們包括：

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

此函數使用匹配器。

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

這是 `EnforceWithMatcher()` 和 `EnforceEx()` 的結合。

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

此函數允許處理一系列任務並返回一個數組。

這是 Casbin 的一個簡單使用案例。您可以使用 Casbin 利用這些 API 啟動一個授權伺服器。我們將在以下段落中展示您其他類型的 API。

管理API

獲取API

這些API用於從策略中檢索特定對象。在這個例子中，我們正在加載一個執行器並從中檢索某些內容。

請查看以下代碼：

```
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)
```

與前面的例子類似，前四行用於從本地文件加載必要信息。我們在此不再進一步討論這一點。

代碼 `allSubjects := enforcer.GetAllSubjects()` 檢索策略文件中的所有主體並將其作為數組返回。然後我們打印該數組。

通常，該代碼的輸出應該是：

```
[admin alice bob]
```

您也可以將函數 `GetAllSubjects()` 更改為 `GetAllNamedSubjects()`，以獲取當前命名策略中出現的主體列表。

同樣地，我們已經為 `Objects`, `Actions`, `Roles` 準備了 `GetAll` 函數。要訪問這些函數，您只需將函數名稱中的 `Subject` 替換為所需的類別即可。

此外，還有更多用於策略的獲取器可用。調用方法和返回值與上述提到的類似。

- `policy = e.GetPolicy()` 獲取策略中的所有授權規則。
- `filteredPolicy := e.GetFilteredPolicy(0, "alice")` 獲取策略中具有指定字段過濾器的所有授權規則。
- `namedPolicy := e.GetNamedPolicy("p")` 獲取命名策略中的所有授權規則。
- `filteredNamedPolicy = e.GetFilteredNamedPolicy("p", 0, "bob")` 獲取命名策略中具有指定字段過濾器的所有授權規則。
- `groupingPolicy := e.GetGroupingPolicy()` 獲取策略中的所有角色繼承規則。
- `filteredGroupingPolicy := e.GetFilteredGroupingPolicy(0, "alice")` 會擷取所有在政策中帶有指定欄位篩選器的角色繼承規則。
- `namedGroupingPolicy := e.GetNamedGroupingPolicy("g")` 會擷取所有在政策中的角色繼承規則。
- `namedGroupingPolicy := e.GetFilteredNamedGroupingPolicy("g", 0, "alice")` 會擷取所有在政策中帶有指定欄位篩選器的角色繼承規則。

新增、刪除、更新 API

Casbin 提供了多種 API，以便在運行時動態地新增、刪除或修改政策。

以下程式碼展示了如何新增、移除和更新政策，以及如何檢查某個政策是否存在：

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

透過使用這些 API，您可以動態地編輯您的政策。同樣地，我們也提供了類似的 API 用於 `FilteredPolicy`, `NamedPolicy`, `FilteredNamedPolicy`, `GroupingPolicy`, `NamedGroupingPolicy`, `FilteredGroupingPolicy`, `FilteredNamedGroupingPolicy`。要使用它們，只需將函數名稱中的 `Policy` 替換為適當的分類即可。

此外，通過將參數更改為陣列，您可以對您的政策進行批量編輯。

例如，考慮這樣的函數：

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

如果我們將 `Policy` 改為 `Policies` 並修改參數如下：

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

那麼我們就可以批量編輯這些策略。

相同的操作也可以應用於 `GroupingPolicy`, `NamedGroupingPolicy`。

AddEx API

Casbin 提供了 AddEx 系列 API 來幫助用戶批量添加規則。

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

這些方法與沒有 Ex 後綴的方法的區別在於，如果其中一條規則已經存在，它們會繼續檢查下一條規則，而不是立即返回 false。

例如，讓我們比較 `AddPolicies` 和 `AddPoliciesEx`。

你可以運行並觀察以下代碼，將其複製到 casbin 的測試中。

```
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 提供了一些 API 供您修改 RBAC 模型和策略。如果您熟悉 RBAC，您可以輕鬆使用這些 API。

在此，我們僅向您展示如何使用 Casbin 的 RBAC API，而不會討論 RBAC 本身。您可以在此處獲得更詳細的資訊 [/docs/rbac](#)。

我們使用以下代碼來加載模型和策略，就像之前一樣。

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

然後，我們可以使用 Enforcer 的一個實例 `enforcer` 來訪問這些 API。

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

`GetRolesForUser()` 返回一個包含 `amber` 所有角色的數組。在這個例子中，`amber` 只有一個角色，即 `admin`，因此數組 `roles` 是 `[admin]`。同樣地，您可以使用 `GetUsersForRole()` 來獲得屬於某個角色的用戶。此函數的返回值也是一個數組。

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

您可以使用 `HasRoleForUser()` 來確認用戶是否屬於某個角色。在此範例中，`amber` 是 `admin` 的成員，因此該函數的返回值為 `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()` 來刪除一個權限。

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

並且使用 `DeletePermissionForUser()` 來刪除某個用戶的權限。

Casbin 擁有許多類似這樣的 API。它們的調用方法和返回值與上述 API 具有相同的風格。您可以在[下一個文件](#)中找到這些 API。

Management API

這個基礎API提供了對Casbin策略管理的全面支援。

Filtered API

幾乎所有過濾API都有相同的參數 (`fieldIndex int, fieldValues ...string`)。

`fieldIndex` 是匹配開始的索引，`fieldValues` 表示結果應該包含的值。注意，`fieldValues` 中的空字符串可以代表任何詞語。

範例：

```
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]]
```

參考

全域變數 `e` 是 Enforcer 實例。

[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 決定一個 "主體" 是否能夠訪問一個 "對象" 並執行 "操作"，輸入參數通常是：(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 使用自定義匹配器來決定一個 "主體" 是否能夠訪問一個 "對象" 並執行 "操作"，輸入參數通常是：(matcher, sub, obj, act)，當 matcher 為空時，默認使用模型匹配器。

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 通過告知匹配的規則來解釋執行情況。

例如：

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

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

const ok = await e.enforceEx(request);

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

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

EnforceExWithMatcher()

EnforceExWithMatcher 使用自訂匹配器並通過告知匹配規則來解釋執行情況。

例如：

[Go](#)

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

BatchEnforce()

BatchEnforce 對每個請求進行強制執行，並以布林數組的形式返回結果。

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 獲取當前策略中出現的主體列表。

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 取得目前命名政策中出現的主體清單。

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 取得目前政策中出現的物件清單。

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 取得目前命名政策中出現的物件清單。

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 取得目前政策中出現的動作清單。

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 獲取當前命名策略中顯示的操作列表。

例如：

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

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

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

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

all_named_actions = e.get_all_named_actions("p")

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

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

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

GetAllRoles()

GetAllRoles 獲取當前策略中顯示的角色列表。

例如：

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

```
allRoles = e.GetAllRoles()

const allRoles = await e.getAllRoles()

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

```
all_roles = e.get_all_roles()

var allRoles = e.GetAllRoles();

let all_roles = e.get_all_roles();

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

GetAllNamedRoles()

GetAllNamedRoles 獲取當前命名策略中顯示的角色列表。

例如：

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

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

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

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

all_named_roles = e.get_all_named_roles("g")

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

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

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

GetPolicy()

取得政策會獲取政策中的所有授權規則。

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()

取得過濾後的政策會獲取政策中的所有授權規則，並可指定欄位過濾條件。

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()

取得命名政策會獲取命名政策中的所有授權規則。

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 取得指定名稱策略中的所有授權規則，可以指定欄位過濾條件。

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 取得策略中的所有角色繼承規則。

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 取得策略中的所有角色繼承規則，可以指定欄位過濾條件。

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 取得指定名稱策略中的所有角色繼承規則。

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 獲取策略中的所有角色繼承規則。

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 判斷是否存在授權規則。

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 判斷是否存在指定的授權規則。

例如：

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

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

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

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

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

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

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

AddPolicy()

AddPolicy 會將一條授權規則添加到當前策略中。如果該規則已經存在，函數將返回 `false` 並且該規則不會被添加。否則，函數將通過添加新規則返回 `true`。

例如：

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

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

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

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

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

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

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

AddPolicies()

AddPolicies 會將授權規則添加到當前策略中。該操作本質上是原子性的。因此，如果授權規則包含與當前策略不一致的規則，函數將返回 `false` 並且不會將任何策略規則添加到當前策略中。如果所有授權規則都與策略規則一致，函數將返回 `true` 並且每個策略規則都會被添加到當前策略中。

例如：

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

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

```

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

const 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 將授權規則添加到當前策略中。如果規則已經存在，則不會添加該規則。但與 AddPolicies 不同，其他不存在的規則會被添加，而不是直接返回 false

例如：

Go

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

AddNamedPolicy()

AddNamedPolicy 將授權規則添加到當前命名策略中。如果規則已經存在，函數返回 false 且不會添加該規則。否則，函數通過添加新規則返回 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 將授權規則添加到當前命名策略中。該操作本質上是原子的。因此，如果授權規則包含與當前策略不一致的規則，函數將返回 `false`，並且不會將任何策略規則添加到當前策略中。如果所有授權規則都與策略規則一致，函數將返回 `true`，並且每個策略規則都會被添加到當前策略中。

例如：

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

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

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

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

```

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 將授權規則添加到當前命名策略中。如果規則已經存在，則該規則不會被添加。但與 AddNamedPolicies 不同，其他不存在的規則會被添加，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 將授權規則添加到當前命名策略中，並禁用 autoNotifyWatcher。如果規則已經存在，則該規則不會被添加。但與 SelfAddPolicies 不同，其他不存在的規則會被添加，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 從當前策略中移除一個授權規則。

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 從當前策略中移除授權規則。 該操作本質上是原子的。 因此，如果授權規則包含與當前策略不一致的規則，該函數將返回 false，並且不會從當前策略中移除任何策略規則。如果所有授權規則都與策略規則一致，該函數將返回 true，並且每個策略規則都會從當前策略中移除。

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'],
```

```

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 從當前策略中移除一個授權規則，可以指定字段過濾器。 RemovePolicy 從當前策略中移除一個授權規則。

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()

移除命名策略從當前命名策略中刪除一個授權規則。

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()

移除命名策略集從當前命名策略中刪除授權規則。該操作本質上是原子的。因此，如果授權規則包含與當前策略不一致的規則，函數返回 `false`，並且當前策略中不會移除任何策略規則。如果所有授權規則都與策略規則一致，函數返回 `true`，並且每個策略規則都從當前策略中移除。

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 從當前命名策略中移除一個授權規則，可以指定字段過濾器。

例如：

Go Node.js PHP Python .NET Rust Java

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

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

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

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

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

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

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

HasGroupingPolicy()

HasGroupingPolicy 判斷是否存在一個角色繼承規則。

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 判斷是否存在一個命名的角色繼承規則。

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 將一個角色繼承規則添加到當前策略中。如果規則已經存在，該函數將返回 false，並且該規則不會被添加。否則，該函數通過添加新規則返回 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 將角色繼承規則添加到當前策略中。該操作在性質上是原子的。因此，如果授權規則包含與當前策略不一致的規則，該函數將返回 `false`，並且不會將任何策略規則添加到當前策略中。如果所有授權規則都與策略規則一致，該函數將返回 `true`，並且每個策略規則都會被添加到當前策略中。

例如：

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

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

areRulesAdded := e.AddGroupingPolicies(rules)

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

const areRulesAdded = await e.addGroupingPolicies(groupingRules);

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

are_rules_added = e.add_grouping_policies(rules)
```

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

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

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

AddGroupingPoliciesEx()

AddGroupingPoliciesEx 將角色繼承規則添加到當前策略中。如果規則已經存在，則不會添加該規則。但與 AddGroupingPolicies 不同，其他不存在的規則會被添加，而不是直接返回 false。

例如：

Go

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

AddNamedGroupingPolicy()

AddNamedGroupingPolicy 將一個命名的角色繼承規則添加到當前策略中。如果規則已經存在，該函數返回 false 且不會添加該規則。否則，該函數通過添加新規則返回 true。

例如：

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

AddNamedGroupingPolicies()

AddNamedGroupingPolicies 將命名的角色繼承規則添加到當前策略中。該操作本質上是原子的。因此，如果授權規則包含與當前策略不一致的規則，函數將返回 `false`，並且不會將任何策略規則添加到當前策略中。如果所有授權規則都與策略規則一致，函數將返回 `true`，並且每個策略規則都會被添加到當前策略中。

例如：

```

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

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

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

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

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

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

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

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

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

```

AddNamedGroupingPoliciesEx()

AddNamedGroupingPoliciesEx 將命名的角色繼承規則添加到當前策略中。如果規則已經存在，該規則將不會被添加。但與 AddNamedGroupingPolicies 不同，其他不存在的規則會被添加，而不是直接返回 false。

例如：

Go

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

RemoveGroupingPolicy()

移除群組策略從當前策略中移除一個角色繼承規則。

例如：

Go Node.js PHP Python .NET Rust Java

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

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

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

RemoveGroupingPolicies()

移除多個群組策略從當前策略中移除角色繼承規則。該操作本質上是原子的。因此，如果授權規則包含與當前策略不一致的規則，函數返回 `false` 且當前策略中不會移除任何策略規則。如果所有授權規則都與策略規則一致，函數返回 `true` 且每個策略規則都從當前策略中移除。

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 從當前策略中移除一個角色繼承規則，可以指定字段過濾器。

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 從當前命名策略中移除一個角色繼承規則。

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 從當前策略中移除命名角色繼承規則。該操作本質上是原子的。因此，如果授權規則包含與當前策略不一致的規則，該函數返回 `false`，並且不會從當前策略中移除任何策略規則。如果所有授權規則都與策略規則一致，該函數返回 `true`，並且每個策略規則都會從當前策略中移除。

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()],
    vec!["jack".to_owned(), "data5_admin".to_owned()]
]
```

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

RemoveFilteredNamedGroupingPolicy()

RemoveFilteredNamedGroupingPolicy 從當前命名策略中移除一個角色繼承規則，可以指定字段過濾器。

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 將舊策略更新為新策略。

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 將所有舊策略更新為新策略。

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)
```

AddFunction()

AddFunction 添加一個自定義函數。

例如：

Go Node.js PHP Python Rust Java

```
func CustomFunction(key1 string, key2 string) bool {  
    if key1 == "/alice_data2/myid/using/res_id" && key2 ==  
"/alice_data/:resource" {  
        return true  
    } else if key1 == "/alice_data2/myid/using/res_id" && key2 ==  
"/alice_data2/:id/using/:resId" {  
        return true  
    } else {  
        return false  
    }  
}  
  
func CustomFunctionWrapper(args ...interface{}) (interface{}, error) {  
    key1 := args[0].(string)  
    key2 := args[1].(string)  
  
    return bool(CustomFunction(key1, key2)), nil  
}
```

```

function customFunction(key1, key2){
    if(key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data/:resource") {
        return true
    } else if(key1 == "/alice_data2/myid/using/res_id" && key2 ==
"/alice_data2/:id/using/:resId") {
        return true
    } else {
        return false
    }
}

e.addFunction("keyMatchCustom", customFunction);

func customFunction($key1, $key2) {
    if ($key1 == "/alice_data2/myid/using/res_id" && $key2 ==
"/alice_data/:resource") {
        return true;
    } elseif ($key1 == "/alice_data2/myid/using/res_id" && $key2 ==
"/alice_data2/:id/using/:resId") {
        return true;
    } else {
        return false;
    }
}

func customFunctionWrapper(...$args){
    $key1 := $args[0];
    $key2 := $args[1];

    return customFunction($key1, $key2);
}

$e->addFunction("keyMatchCustom", customFunctionWrapper);

def custom_function(key1, key2):
    return ((key1 == "/alice_data2/myid/using/res_id" and key2 ==
"/alice_data/:resource") or (key1 == "/alice_data2/myid/using/res_id"

```

```

fn custom_function(key1: 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 從文件/數據庫加載過濾後的策略。

例如：

[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 從文件/數據庫附加一個過濾後的策略。

例如：

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

```
err := e.LoadIncrementalFilteredPolicy()

const ok = await e.loadIncrementalFilteredPolicy();

adapter =
casbin.persist.adapters.FilteredAdapter("rbac_with_domains_policy.csv")
```

UpdateGroupingPolicy()

UpdateGroupingPolicy 將 `g` 區段中的 oldRule 更新為 newRule

例如：

[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 將名為 `ptype` 的 oldRule 更新為 `g` 區段中的 newRule

例如：

[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 支援對 `sub`, `obj`, `domain` 和 `priority` 的常規名稱和位置進行自定義

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

例如：

Go

```
e.SetFieldIndex("p", constant.PriorityIndex, 0)
e.SetFieldIndex("p", constant.SubjectIndex, 4)
```

RBAC API

一個更友好的RBAC API。 此API是管理API的子集。 RBAC用戶可以使用此API來簡化代碼。

參考

全局變量 `e` 是Enforcer實例。

[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()

取得使用者角色會獲取某個使用者所擁有的角色。

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()

取得角色使用者會獲取擁有某個角色的使用者。

例如：

Go

Node.js

PHP

Python

.NET

Rust

Java

```
res := e.GetUsersForRole("data1_admin")  
  
const res = await e.getUsersForRole('data1_admin')  
  
$res = $e->getUsersForRole("data1_admin");  
  
users = e.get_users_for_role("data1_admin")  
  
var res = e.GetUsersForRole("data1_admin");  
  
let users = e.get_users_for_role("data1_admin", None); // No  
domain  
  
List<String> res = e.getUsersForRole("data1_admin");
```

HasRoleForUser()

使用者是否有角色用於判斷某個使用者是否擁有某個角色。

例如：

Go

Node.js

PHP

Python

.NET

Rust

Java

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

```
const res = await e.hasRoleForUser('alice', 'data1_admin')

$res = $e->hasRoleForUser("alice", "data1_admin");

has = e.has_role_for_user("alice", "data1_admin")

var res = e.HasRoleForUser("alice", "data1_admin");

let has = e.has_role_for_user("alice", "data1_admin", None); //  
No domain

boolean res = e.hasRoleForUser("alice", "data1_admin");
```

AddRoleForUser()

AddRoleForUser 為用戶添加一個角色。如果用戶已經擁有該角色，則返回 false (即未受影響)。

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 為用戶添加多個角色。如果用戶已經擁有其中一個角色，則返回 false（即未受影響）。

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 為用戶刪除一個角色。如果用戶沒有該角色，則返回 false（即未受影響）。

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 刪除用戶的所有角色。如果用戶沒有任何角色（即未受影響），則

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 刪除一個用戶。如果用戶不存在（即未受影響），則返回 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 刪除一個角色。

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 刪除一個權限。如果權限不存在則返回 false（即未受影響）。

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 為用戶或角色添加一個權限。如果用戶或角色已經擁有該權限則返回 false (即未受影響)。

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 為使用者或角色新增多項權限。如果使用者或角色已擁有其中一項權限，則返回 false（即未受影響）。

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 為使用者或角色刪除一項權限。如果使用者或角色未擁有該權限，則返回 false (即未受影響)。

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 為使用者或角色刪除多項權限。如果使用者或角色沒有任何權限（即未受影響），則返回 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 獲取使用者或角色的權限。

例如：

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

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

```
await e.getPermissionsForUser('bob')
```

```
$e->getPermissionsForUser("bob");
```

```
e.get_permissions_for_user("bob")
```

```
var permissions = e.GetPermissionsForUser("bob");
```

```
List<List<String>> permissions = e.getPermissionsForUser("bob");
```

HasPermissionForUser()

HasPermissionForUser 判斷使用者是否擁有某項權限。

例如：

```
e.HasPermissionForUser("alice", []string{"read"})  
  
await e.hasPermissionForUser('alice', 'read')  
  
$e->hasPermissionForUser("alice", []string{"read"});  
  
has = e.has_permission_for_user("alice", "read")  
  
var has = e.HasPermissionForUser("bob", "read");  
  
let has = e.has_permission_for_user("alice",  
    vec!["data1".to_owned(), "read".to_owned()]);  
  
boolean has = e.hasPermissionForUser("alice", "read");
```

GetImplicitRolesForUser()

GetImplicitRolesForUser 獲取使用者擁有的隱含角色。相較於 GetRolesForUser()，此函數除了直接角色外，還會檢索間接角色。

例如：

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

GetRolesForUser("alice") 只能獲取：["role:admin"]。但
GetImplicitRolesForUser("alice") 將獲取：["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");
```

GetImplicitUsersForRole()

GetImplicitUsersForRole 獲取所有繼承該角色的用戶。相較於 GetUsersForRole()，此函數會檢索間接用戶。

例如：

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

GetUsersForRole("role:user") 只能獲取：["role:admin"]。但

GetImplicitUesrsForRole("role:user") 將獲得: ["role:admin", "alice"]。

例如:

Go Node.js Java

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

GetImplicitPermissionsForUser()

GetImplicitPermissionsForUser 獲取用戶或角色的隱含權限。\\ 與 GetPermissionsForUser() 相比，此函數檢索繼承角色的權限。

例如:

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

GetPermissionsForUser("alice") 只能獲得: [["alice", "data2", "read"]]。\\ 但 GetImplicitPermissionsForUser("alice") 將獲得: [[["admin", "data1", "read"], ["alice", "data2", "read"]]]。

例如:

```
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 透過指定政策名稱，獲取用戶或角色的隱含權限

例如：

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

`GetImplicitPermissionsForUser("alice")` 僅獲取: `[["admin", "data1", "read"]]`, 其政策為默認的 "p"

但您可以指定政策為 "p2", 透過 `GetNamedImplicitPermissionsForUser("p2","alice")` 獲取: `[["admin", "create"]]`

例如:

[Go](#) [Python](#)

```
e.GetNamedImplicitPermissionsForUser("p2", "alice")  
  
e.get_named_implicit_permissions_for_user("p2", "alice")
```

GetDomainsForUser()

`GetDomainsForUser` 獲取用戶所擁有的所有域

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")` 可能獲取 `["domain1", "domain2"]`

例如:

[Go](#)

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

GetImplicitResourcesForUser()

GetImplicitResourcesForUser 返回所有對用戶應該為真的策略。

例如：

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

g, alice, data2_admin
```

GetImplicitResourcesForUser("alice") 將返回

[Go](#)

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

GetImplicitUsersForPermission()

GetImplicitUsersForPermission 獲取某個權限的隱含用戶。

例如：

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

GetImplicitUsersForPermission("data1", "read") 將返回: `["alice", "bob"]`。

注意: 只會返回用戶, 角色 ("g" 中的第二個參數) 將被排除。

Go

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

GetAllowedObjectConditions()

GetAllowedObjectConditions 返回一個字串陣列, 包含用戶可以訪問的對象條件。

例如:

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

注意:

0. 前綴: 您可以自定義對象條件的前綴, 而 "r.obj." 通常用作前綴。 移除前綴後, 剩餘

部分即為對象的條件。如果有不符合前綴要求的 obj 策略，將返回 `errors.ERR_OBJ_CONDITION`。

1. 如果 'objectConditions' 陣列為空，則返回 `errors.ERR_EMPTY_CONDITION`。

Go

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

GetImplicitUsersForResource()

`GetImplicitUsersForResource` 根據資源返回隱含的使用者。

例如：

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

`GetImplicitUsersForResource("data2")` 將返回 `[["bob", "data2", "write"], ["alice", "data2", "read"] ["alice", "data2", "write"]], nil`。

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

Go

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

① 注意

只會返回使用者，角色（"g" 中的第二個參數）將被排除。

RBAC with Domains API

一個更用戶友好的RBAC帶域API。此API是管理API的一個子集。RBAC用戶可以使用此API來簡化他們的代碼。

參考

全局變量 `e` 代表Enforcer實例。

[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()` 函數會擷取在一個特定領域內擁有某個角色的使用者。

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()` 函數會擷取一個使用者在特定領域內擁有的角色。

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()` 函數會擷取一個使用者或角色在特定領域內的權限。

例如：

Go Java

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

AddRoleForUserInDomain()

`AddRoleForUserInDomain()` 函數在特定域中為用戶添加角色。如果用戶已經擁有該

角色，則返回 `false`（未進行任何更改）。

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()` 函數在特定域中為用戶移除角色。如果用戶沒有該角色，則返回 `false`（未進行任何更改）。

例如：

[Go](#) [Java](#)

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

DeleteRolesForUserInDomain()

`DeleteRolesForUserInDomain()` 函數在特定域中為用戶移除所有角色。如果用戶沒有任何角色，則返回 `false`（未進行任何更改）。

例如：

[Go](#)

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

GetAllUsersByDomain()

`GetAllUsersByDomain()` 函數會擷取與指定域相關的所有用戶。如果模型中未定義域，則返回一個空字符串數組。

例如：

[Go](#)

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

DeleteAllUsersByDomain()

`DeleteAllUsersByDomain()` 函數會刪除與指定域相關的所有用戶。如果模型中未定義域，則返回 `false`。

例如：

[Go](#)

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

DeleteDomains()

DeleteDomains 會刪除所有相關的使用者和角色。如果未提供參數，它將刪除所有域。

例如：

[Go](#)

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

GetAllDomains()

GetAllDomains 會獲取所有域。

例如：

[Go](#)

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

i 注意

如果你正在處理一個類似 `name::domain` 的域，這可能會導致意外行為。在 Casbin 中，`::` 是一個保留關鍵字，就像編程語言中的 `for`、`if` 一樣，我們絕不應該在域中放入 `::`。

GetAllRolesByDomain()

`GetAllRolesByDomain` 會獲取與該域相關的所有角色。

例如：

[Go](#)

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

i 注意

此方法不適用於具有繼承關係的領域，也稱為隱含角色。

GetImplicitUsersForResourceByDomain()

`GetImplicitUsersForResourceByDomain` 根據資源和領域返回隱含的使用者。

例如：

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

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

Go

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

ⓘ 注意

只會返回使用者，角色（"g" 中的第二個參數）將被排除。

RBAC with Conditions API

一個更用戶友好的API，用於帶條件的RBAC。

參考

AddNamedLinkConditionFunc

AddNamedLinkConditionFunc 為連結 `userName->roleName` 添加條件函數 fn,

[Go](#)

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

AddNamedDomainLinkConditionFunc

AddNamedDomainLinkConditionFunc 為連結 `userName-> {roleName, domain}`
添加條件函數 fn,

[Go](#)

```
e.AddNamedDomainLinkConditionFunc("g", "userName", "roleName",  
"domainName", YourLinkConditionFunc)
```

SetNamedLinkConditionFuncParams

`SetNamedLinkConditionFuncParams` 設置連結 `userName->roleName` 的條件函數 fn 的參數

Go

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

SetNamedDomainLinkConditionFuncParams

`SetNamedDomainLinkConditionFuncParams` 設置連結 `userName->{roleName, domain}` 的條件函數 fn 的參數

Go

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

RoleManager API

RoleManager

RoleManager 提供了一個介面，用於定義管理角色的操作。在 RoleManager 中增加匹
配函數，允許在角色名稱和域中使用萬用字元。

AddNamedMatchingFunc()

AddNamedMatchingFunc 函數將 MatchingFunc 按 Ptype 添加到 RoleManager 中。
MatchingFunc 將在執行角色匹配時被使用。

[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 函數通過 Ptype 向 RoleManager 添加一個 MatchingFunc。DomainMatchingFunc 與上述列出的 MatchingFunc 相似。

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 函數獲取當前為 `g` 的角色管理器。

例如：

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

```
rm := e.GetRoleManager()  
  
const rm = await e.getRoleManager();  
  
rm = e.get_role_manager()
```

GetNamedRoleManager()

GetNamedRoleManager 函數通過指定的 Ptype 獲取角色管理器。

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 函數設置 `g` 的當前角色管理器。

例如：

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

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

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

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

SetNamedRoleManager()

SetNamedRoleManager 函數通過指定的 Ptype 設置角色管理器。

例如：

[Go](#) [Python](#)

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

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

Clear()

Clear 函數清除所有存儲的數據並將角色管理器重置為其初始狀態。

例如：

Go Node.js Python

```
rm.Clear()  
  
await rm.clear();  
  
rm.clear()
```

AddLink()

AddLink 在兩個角色之間添加繼承連結。 角色：name1 和角色：name2。 域是角色的前綴（可用於其他目的）。

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 刪除兩個角色之間的繼承連結。 角色: name1 和角色: name2。 域是角色的前綴 (可用於其他目的)。

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 判斷兩個角色之間是否存在連結。 角色: name1 繼承角色: name2。 域 (Domain) 是角色的一個前綴 (可用於其他目的)。

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 獲取用戶繼承的角色。 域（Domain）是角色的一個前綴（可用於其他目的）。

例如：

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

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

await rm.getRoles('u1', 'domain1');

rm.get_roles("u1", "domain")
```

GetUsers()

取得使用者會獲取繼承某個角色的使用者。 網域是使用者名稱的前綴（可用於其他用途）。

例如：

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

```
rm.GetUsers("g1")  
  
await rm.getUsers('g1');  
  
rm.get_users("g1")
```

PrintRoles()

列印角色會將所有角色輸出到日誌。

例如：

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

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

SetLogger()

設置日誌器會設定角色管理器的日誌器。

例如：

[Go](#)

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

GetDomains()

取得網域功能會獲取用戶擁有的網域

例如：

Go

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



> 高級用法

高級用法



Multi-threading

在多執行緒環境中使用 Casbin



Benchmarks

Casbin 中政策執行的開銷



Performance Optimization

Casbin 效能優化



Authorization of Kubernetes

基於 Casbin 的 Kubernetes (k8s) RBAC 與 ABAC 授權中介軟體



Admission Webhook for K8s

基於 Casbin 的 Kubernetes (K8s) RBAC 與 ABAC 授權中介軟體



Authorization of Service Mesh through Envoy

透過 Envoy 授權服務網格

Multi-threading

在多執行緒環境中使用 Casbin 時，您可以使用 Casbin enforcer 的同步包裝器：

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，它還提供了對 "AutoLoad" 功能的支援，允許 Casbin enforcer 在政策規則發生變更時自動從資料庫載入最新的政策規則。要啟動定期自動載入政策，請調用 "StartAutoLoadPolicy()" 函數。同樣地，要停止此自動載入，請調用 "StopAutoLoadPolicy()" 函數。

Benchmarks

Go Python C++ Lua (JIT)

政策執行的開銷已在 `model_b_test.go` 中進行了基準測試。測試平台配置如下：

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

以下是通過運行 `go test -bench=.` 獲得的基準測試結果 `-benchmem` (`op` = 一次 `Enforce()` 調用, `ms` = 毫秒, `KB` = 千字節)：

測試案例	規則大小	時間開銷 (ms/op)	記憶體開銷 (KB)
存取控制列表 (ACL)	2條規則 (2個使用者)	0.015493	5.649
角色基礎存取控制 (RBAC)	5條規則 (2個使用者, 1個角色)	0.021738	7.522
小型角色基礎存取控制 (RBAC)	1100條規則 (1000位用戶, 100個角色)	0.164309	80.620
RBAC (中等規模)	11000條規則 (10000位用戶, 1000個角色)	2.258262	765.152
RBAC (大型規模)	110000條規則 (100000位用戶, 10000個角色)	23.916776	7,606
基於資源角色的RBAC	6條規則 (2個用戶, 2個角色)	0.021146	7.906
帶有域/租戶的RBAC	6條規則 (2個用戶, 1個角色, 2個域)	0.032696	10.755
ABAC	0 規則 (0 使用者)	0.007510	2.328
RESTful	5 規則 (3 使用者)	0.045398	91.774
拒絕覆蓋	6 規則 (2 使用者, 1 角色)	0.023281	8.370
優先權	9 條規則 (2 個用戶, 2 個角色)	0.016389	5.313

在 `Pycasbin` 中，政策執行的開銷已在 `tests/benchmarks` 目錄中進行了基準測試。測試平台配置如下：

```
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
```

以下是執行 `casbin_benchmark` 獲得的基準測試結果 (`op` = 一次 `enforce()` 調用, `ms` = 毫秒)：

測試案例	規則大小	時間開銷 (毫秒/操作)
存取控制列表 (ACL)	2條規則 (2個用戶)	0.067691
基於角色的存取控制 (RBAC)	5條規則 (2個用戶, 1個角色)	0.080045

測試案例	規則大小	時間開銷 (毫秒/操作)
小型RBAC	1100條規則 (1000個用戶, 100個角色)	0.853590
RBAC (中等)	11000條規則 (10000個用戶, 1000個角色)	6.986668
RBAC (大型)	110000條規則 (100000個用戶, 10000個角色)	77.922851
帶有資源角色的RBAC	6條規則 (2個用戶, 2個角色)	0.106090
帶有域/租戶的RBAC	6條規則 (2個用戶, 1個角色, 2個領域)	0.103628
ABAC	0條規則 (0個用戶)	0.053213
RESTful	5條規則 (3個用戶)	不適用
拒絕覆蓋	6條規則 (2個用戶, 1個角色)	NA
優先權	9 條規則 (2 個用戶, 2 個角色)	0.084684

在 Casbin CPP 中，政策執行的開銷已經在 `tests/benchmarks` 目錄下使用 Google 的基準測試工具 進行了基準測試。測試平台配置如下：

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

以下是從在 `Release` 配置中構建的 `casbin_benchmark` 目標執行獲得的基準測試結果 (`op = 一次 enforce() 調用, ms = 毫秒`) :

測試案例	規則數量	時間開銷 (ms/op)
ACL	2條規則 (2位用戶)	0.0195
RBAC	5條規則 (2位用戶, 1個角色)	0.0288
RBAC (小型)	1100條規則 (1000位用戶, 100個角色)	0.300
RBAC (中型)	11000條規則 (10000位使用者, 1000個角色)	2.113
大型RBAC	110000條規則 (100000位使用者, 10000個角色)	21.450
帶有資源角色的RBAC	6條規則 (2位使用者, 2個角色)	0.03
帶有域/租戶的RBAC	6條規則 (2位使用者, 1個角色, 2個域)	0.041
ABAC	0 規則 (0 用戶)	不適用
RESTful	5 規則 (3 用戶)	不適用
拒絕覆蓋	6 規則 (2 用戶, 1 角色)	0.0246
優先權	9 條規則 (2 個用戶, 2 個角色)	0.035

在 [Lua Casbin](#) 中，政策執行的開銷已在 `bench.lua` 中進行了基準測試。測試平台配置如下：

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

以下是通過運行 `luajit bench.lua` 獲得的基準測試結果（op = 一次 `enforce()` 調用，ms = 毫秒）：

測試案例	規則大小	時間開銷 (ms/op)
ACL	2條規則 (2位用戶)	0.0533
基於角色的存取控制 (RBAC)	5條規則 (2位用戶, 1個角色)	0.0972
基於角色的存取控制 (小型)	1100條規則 (1000位用戶, 100個角色)	0.8598
基於角色的存取控制 (中型)	11000條規則 (10000位用戶, 1000個角色)	8.6848
RBAC (大型)	110000條規則 (100000名用戶, 10000個角色)	90.3217
帶有資源角色的RBAC	6條規則 (2名用戶, 2個角色)	0.1124
帶有域/租戶的RBAC	6條規則 (2名用戶, 1個角色, 2個域)	0.1978
ABAC	0 規則 (0 用戶)	0.0305
RESTful	5 規則 (3 用戶)	0.1085
拒絕覆蓋	6 規則 (2 用戶, 1 角色)	0.1934
優先級	9條規則 (2個用戶, 2個角色)	0.1437

基準監控

在下方嵌入的網頁中，您可以看到Casbin每個提交的性能變化。

您也可以直接在以下網址瀏覽：<https://v1.casbin.org/casbin/benchmark-monitoring>

Last Update:
Repository:

[Download data as JSON](#)

Powered by [github-action-benchmark](#)

Performance Optimization

當應用於擁有數百萬用戶或權限的生產環境時，您可能會遇到 Casbin 執行效能下降的情況。通常有兩個原因：

高流量

每秒接收的請求數量過大，例如，單一 Casbin 實例每秒接收 10,000 個請求。在這種情況下，單一 Casbin 實例通常不足以處理所有請求。有兩種可能的解決方案：

1. 使用多線程來啟用多個 Casbin 實例，以便充分利用機器上的所有核心。更多詳情，請參閱：[多線程](#)。
2. 將 Casbin 實例部署到集群（多台機器），並使用 Watcher 確保所有 Casbin 實例保持一致。更多詳情，請參閱：[Watcher](#)。

注意

您可以同時使用上述兩種方法，例如，將 Casbin 部署到一個由 10 台機器組成的集群，每台機器同時運行 5 個線程處理 Casbin 的執行請求。

大量策略規則

在雲端或多租戶環境中，可能需要數百萬條策略規則。每次執行調用或在初始時加載策略規則都可能非常緩慢。這類情況通常可以通過以下幾種方式緩解：

1. 檢查您的 Casbin 模型或策略是否設計得當。一個設計良好的模型和策略能夠抽象出每個用戶/租戶的重複邏輯，並將規則數量減少到非常低的水平 ($\lt 100$)。例如，您可以跨所有租戶共享一些預設規則，並允許用戶稍後自定義他們的規則。自定義規則可以覆蓋預設規則。如果您有進一步的問題，請在 Casbin 倉庫上開啟一個

GitHub問題。

2. 進行分片，使Casbin執行器僅加載一小部分策略規則。例如，enforcer_0可以服務租戶_0到租戶_99，而enforcer_1可以服務租戶_100到租戶_199。要僅加載所有策略規則的一個子集，請參閱：[策略子集加載](#)。
3. 將權限授予RBAC角色，而不是直接授予用戶。Casbin的RBAC是通過角色繼承樹（作為緩存）實現的。因此，給定一個用戶，如Alice，Casbin僅需O(1)時間來查詢RBAC樹中的角色-用戶關係並執行強制。如果您的g規則不經常更改，那麼RBAC樹將不需要不斷更新。查看此討論的詳細內容請點擊這裡：<https://github.com/casbin/casbin/issues/681#issuecomment-763801583>

ⓘ 注意

您可以同時嘗試上述所有方法。

Authorization of Kubernetes

[K8s-authz](#) 是一個基於 Casbin 的 Kubernetes (k8s) 授權中介軟體，利用 RBAC（基於角色的存取控制）和 ABAC（基於屬性的存取控制）進行政策執行。此中介軟體與 K8s 驗證准入網路鉤子整合，以驗證 Casbin 為每個對 K8s 資源的請求所定義的政策。自訂准入控制器使用 `ValidatingAdmissionWebhook` 向 Kubernetes 註冊，以對 API 伺服器轉發的請求對象進行驗證，並提供回應，指示請求是否應被允許或拒絕。

為了決定何時將傳入請求發送到准入控制器，已實施了一個驗證網路鉤子。此網路鉤子代理任何類型的 K8s 資源或子資源的請求，並執行政策驗證。只有當用戶被 Casbin 執行者授權時，才能對這些資源進行操作。[執行者](#) 檢查用戶的角色，如政策中所定義。此中介軟體的部署目標是 K8s 叢集。

需求

在繼續之前，請確保您已具備以下條件：

- 一個運行的 Kubernetes 叢集。您可以使用 Docker 設置本地叢集，或在您的伺服器上設置完整的 Kubernetes 生態系統。有關詳細說明，請參考此 [指南](#) 以在 Windows 上設置本地 Kubernetes 叢集，或參考此 [指南](#) 以在 Linux 上設置叢集。
- Kubectl CLI。有關在 Windows 上安裝 Kubectl 的說明可以在 [這裡](#) 找到，而在 Linux 上的安裝說明可以在 [這裡](#) 找到。
- OpenSSL

使用方法

依照以下步驟使用 K8s-authz:

1. 使用 OpenSSL 為每位使用者生成證書和密鑰。運行以下腳本：

```
./gen_cert.sh
```

2. 通過運行以下命令手動從 Dockerfile 構建 Docker 映像。記得在命令和部署文件中相應地更改構建版本。

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

3. 在 model.conf 和 policy.csv 文件中定義 Casbin 策略。有關這些策略如何工作的更多信息，請參閱 [文檔](#)。
4. 在部署之前，您可以根據您的具體需求修改 main.go 文件以及驗證 webhook 配置文件 中的端口。
5. 通過運行以下命令在 Kubernetes 集群上部署驗證控制器和 webhook：

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

6. 對於生產服務器，建議創建一個 Kubernetes secret 來保護證書：

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

7. 完成上述步驟後，您需要在 `main.go` 和 `manifests` 中更新證書目錄，並將其指向創建的 `secret` 目錄。

現在，伺服器應該已經啟動並運行，準備好驗證對 K8s 資源的請求並相應地執行策略。

Admission Webhook for K8s

1. Casbin K8s-Gatekeeper 概述與文件

Casbin K8s-GateKeeper 是一個 Kubernetes 准入 Webhook，整合了 Casbin 作為存取控制工具。通過使用 Casbin K8s-GateKeeper，您可以建立靈活的規則來授權或攔截對 K8s 資源的任何操作，無需編寫任何程式碼，僅需幾行 Casbin 模型和策略的宣告式配置，這些是 Casbin ACL（存取控制列表）語言的一部分。

Casbin K8s-GateKeeper 由 Casbin 社群開發和維護。該專案的倉庫可在這裡找到：<https://github.com/casbin/k8s-gatekeeper>

0.1 一個簡單的例子

例如，您不需要編寫任何代碼，但可以使用以下配置行來實現此功能："禁止使用某些指定標籤的圖像在任何部署中"：

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
```

這些是普通的Casbin ACL語言。假設您已經閱讀了有關它們的章節，理解起來將非常容易。

Casbin K8s-Gatekeeper具有以下優點：

- 易於使用。編寫幾行ACL遠比編寫大量代碼要好。
- 它允許配置的熱更新。您不需要關閉整個插件來修改配置。

- 它具有彈性。可以在任何 K8s 資源上制定任意規則，這些規則可以通過 `kubectl gatekeeper` 進行探索。
- 它簡化了 K8s 準入 Webhook 的實現，這是非常複雜的。您不需要知道 K8s 準入 Webhook 是什麼，也不需要知道如何為其編寫代碼。您只需要知道您想要施加限制的資源，然後編寫 Casbin ACL。大家都知道 K8s 很複雜，但通過使用 Casbin K8s-Gatekeeper，可以節省您的時間。
- 它由 Casbin 社群維護。如果有任何關於此插件的疑問，或者在嘗試使用時遇到任何問題，歡迎隨時聯繫我們。

1.1 Casbin K8s-Gatekeeper 是如何工作的？

K8s-Gatekeeper 是一個 K8s 的準入 Webhook，使用 [Casbin](#) 來應用任意用戶定義的訪問控制規則，以防止管理員不希望的任何 K8s 操作。

Casbin 是一個強大且高效能的開源存取控制庫。它提供對基於各種存取控制模型的授權強制執行的支援。有關 Casbin 的更多詳情，請參閱[概述](#)。

K8s 中的準入 Webhook 是接收「準入請求」並對其進行處理的 HTTP 回調。特別是，K8s-Gatekeeper 是一種特殊的準入 Webhook：「ValidatingAdmissionWebhook」，它可以決定是否接受或拒絕此準入請求。至於準入請求，它們是描述對 K8s 指定資源（例如，創建/刪除部署）進行操作的 HTTP 請求。有關準入 Webhook 的更多資訊，請參閱 [K8s 官方文件](#)。

1.2 一個示例說明其工作原理

例如，當有人想要創建一個包含運行 nginx 的 Pod 的部署（使用 `kubectl` 或 K8s 客戶端），K8s 將生成一個準入請求，該請求（如果轉換為 YAML 格式）可能如下所示：

```
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
```

此請求將經過圖片中顯示的所有中間件處理流程，包括我們的 K8s-Gatekeeper。K8s-Gatekeeper 能夠偵測到所有存儲在 K8s 的 etcd 中的 Casbin 執行器，這些執行器是由使用者（透過 `kubectl` 或我們提供的 Go 客戶端）建立和維護的。

每個執行器包含一個 Casbin 模型和一個 Casbin 策略。准入請求將逐一被每個執行器處理，只有通過所有執行器的檢查，請求才能被這個 K8s-Gatekeeper 接受。

(如果你不理解什麼是 Casbin 執行器、模型或策略，請參閱此文件：[開始使用](#)）。

例如，由於某些原因，管理員希望禁止使用 'nginx:1.14.1' 映像檔，而允許使用 'nginx:1.3.1'。可以建立一個包含以下規則和策略的執行器（我們將在後續章節解釋如何建立執行器，這些模型和策略是什麼，以及如何編寫它們）。

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
```

透過建立包含上述模型和策略的執行器，之前的准入請求將被該執行器拒絕，這意味著 K8s 不會建立此部署。

2 安裝 K8s-gatekeeper

安裝 K8s-gatekeeper 有三種方法可供選擇：外部 webhook、內部 webhook 和 Helm。

ⓘ 注意

注意：這些方法僅供用戶試用 K8s-gatekeeper，並不安全。如果您希望在生產環境中使用它，請確保您閱讀了 [第 5 章。進階設定](#) 並在安裝前進行必要的修改。

2.1 內部 webhook

2.1.1 步驟1：建立映像檔

對於內部 webhook 方法， webhook 本身將作為 Kubernetes 內部的服務來實現。要創建必要的服務和部署，您需要建立 K8s-gatekeeper 的映像檔。您可以通過運行以下命令來建立自己的映像檔：

```
docker build --target webhook -t k8s-gatekeeper .
```

此命令將創建一個名為 'k8s-gatekeeper:latest' 的本地映像檔。

① 注意

注意：如果您正在使用 minikube，請在執行 'docker build' 之前先執行 `eval $(minikube -p minikube docker-env)`。

2.1.2 步驟 2：為 K8s-gatekeeper 設置服務和部署

執行以下命令：

```
kubectl apply -f config/rbac.yaml  
kubectl apply -f config/webhook_deployment.yaml  
kubectl apply -f config/webhook_internal.yaml
```

這將開始運行 K8s-gatekeeper，您可以通過執行 `kubectl get pods` 來確認這一點。

2.1.3 步驟 3：為 K8s-gatekeeper 安裝 CRD 資源

執行以下命令：

```
kubectl apply -f config/auth.casbin.org_casbinmodels.yaml  
kubectl apply -f config/auth.casbin.org_casbinpolicies.yaml
```

2.2 外部 webhook

對於外部 webhook 方法，K8s-gatekeeper 將在 Kubernetes 外部運行，而 Kubernetes 將像訪問普通網站一樣訪問 K8s-gatekeeper。Kubernetes 有一個強制要求，即准入 webhook 必須是 HTTPS。為了試用 K8s-gatekeeper，我們提供了一組證書和一個私鑰（儘管這不安全）。若您偏好使用自己的憑證，請參閱[第五章 進階設定](#)以獲取調整憑證和私鑰的指示。

我們提供的憑證是為 'webhook.domain.local' 發行的。因此，請修改主機（例如，/etc/hosts）並將 'webhook.domain.local' 指向 K8s-gatekeeper 運行的 IP 地址。

然後執行以下命令：

```
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 透過 Helm 安裝 K8s-gatekeeper

2.3.1 第一步：建立映像檔

請參閱[第二章 2.1.1](#)。

2.3.2 Helm 安裝

運行命令 `helm install k8sgatekeeper ./k8sgatekeeper`。

3. 嘗試 K8s-gatekeeper

3.1 創建 Casbin 模型和策略

您有兩種方法來創建模型和策略：通過 `kubectl` 或通過我們提供的 `go-client`。

3.1.1 通過 `kubectl` 創建/更新 Casbin 模型和策略

在 K8s-gatekeeper 中，Casbin 模型存儲在一個名為 'CasbinModel' 的 CRD 資源中。其定義位於 `config/auth.casbin.org_casbinmodels.yaml`。

示例在 `example/allowed_repo/model.yaml` 中。請注意以下字段：

- `metadata.name`: 模型的名稱。此名稱必須與與此模型相關的 CasbinPolicy 物件的名稱相同，以便 K8s-gatekeeper 可以將它們配對並創建一個執行器。
- `spec.enable`: 如果此欄位設置為 "false"，此模型（以及與此模型相關的 CasbinPolicy 物件）將被忽略。
- `spec.modelText`: 包含 Casbin 模型文本的字符串。

Casbin 策略存儲在另一個名為 'CasbinPolicy' 的 CRD 資源中，其定義可以在 `config/auth.casbin.org_casbinpolicies.yaml` 中找到。

`example/allowed_repo/policy.yaml` 中有示例。請注意以下欄位：

- `metadata.name`: 策略的名稱。此名稱必須與與此策略相關的 CasbinModel 物件的名稱相同，以便 K8s-gatekeeper 可以將它們配對並創建一個執行器。

- spec.policyItem: 包含 Casbin 模型策略文本的字符串。

在創建自己的 CasbinModel 和 CasbinPolicy 文件後，使用以下命令應用它們：

```
kubectl apply -f <filename>
```

一旦建立了一對 CasbinModel 和 CasbinPolicy，K8s-gatekeeper 將能在 5 秒內偵測到它。

3.1.2 透過我們提供的 go-client 建立/更新 Casbin 模型和策略

我們理解在某些情況下，直接在 K8s 集群的節點上使用 shell 執行命令可能不太方便，例如當您正在為您的公司構建自動化雲平台時。因此，我們開發了一個 go-client 來創建和維護 CasbinModel 和 CasbinPolicy。

go-client 庫位於 `pkg/client` 中。

在 `client.go` 中，我們提供了一個創建客戶端的函數。

```
func NewK8sGateKeeperClient(externalClient bool) (*K8sGateKeeperClient, error)
```

`externalClient` 參數決定了 K8s-gatekeeper 是否在 K8s 集群內部運行。

在 `model.go` 中，我們提供了各種函數來創建、刪除和修改 CasbinModel。您可以在 `model_test.go` 中了解如何使用這些接口。

在 `policy.go` 中，我們提供了各種函數來創建、刪除和修改 CasbiPolicy。您可以在 `policy_test.go` 中了解如何使用這些介面。

3.1.2 嘗試 K8s-gatekeeper 是否運作

假設您已經在 `example/allowed_repo` 中創建了確切的模型和策略。現在，嘗試以下命令：

```
kubectl apply -f example/allowed_repo/testcase/reject_1.yaml
```

您應該會發現 K8s 會拒絕此請求，並提到網鈎是拒絕此請求的原因。然而，當您嘗試應用 `example/allowed_repo/testcase/approve_2.yaml` 時，它將被接受。

4. 如何使用 K8s-gatekeeper 編寫模型和策略

首先，確保您熟悉 Casbin 模型和策略的基本語法。如果您不熟悉，請先閱讀 [入門](#) 部分。在本章中，我們假設您已經理解什麼是 Casbin 模型和策略。

4.1 模型請求定義

當 K8s-gatekeeper 進行授權請求時，輸入總是一個對象：Admission Request 的 Go 對象。這意味著執行器將總是這樣使用：

```
ok, err := enforcer.Enforce(admission)
```

其中 `admission` 是一個由 K8s 官方 go api `"k8s.io/api/admission/v1"` 定義的 `AdmissionReview` 對象。您可以在這個倉庫中找到這個結構體的定義：<https://github.com/kubernetes/api/blob/master/admission/v1/types.go>。更多信息，您也可以參考 <https://kubernetes.io/docs/reference/access-authn-authz/extensible-admission-controllers/#webhook-request-and-response>。

因此，對於 K8s-gatekeeper 使用的任何模型，`request_definition` 的定義應該總是這樣：

```
[request_definition]
r = obj
```

名稱 'obj' 不是強制性的，只要名稱與 `[matchers]` 部分中使用的名稱一致即可。

4.2 模型匹配器

您應該使用 Casbin 的 ABAC 功能來編寫您的規則。然而，Casbin 內建的表達式評估器不支援在映射或陣列（切片）中進行索引，也不支援陣列的擴展。因此，K8s-gatekeeper 提供了各種「Casbin 函數」作為擴展來實現這些功能。如果您發現這些擴展仍無法滿足您的需求，請隨時開始一個問題，或創建一個拉取請求。

如果您不熟悉 Casbin 函數，可以參考 [Function](#) 以獲取更多信息。

以下是擴展函數：

4.2.1 擴展函數

4.2.1.1 access

Access 用於解決 Casbin 不支援在映射或陣列中進行索引的問題。`example/allowed_repo/model.yaml` 示例展示了此函數的使用方法：

```
[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
```

在這個匹配器中，`access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0, "Image")`

等同於 `r.obj.Request.Object.Object.Spec.Template.Spec.Containers[0].Image`，其中 `r.obj.Request.Object.Spec.Template.Spec.Containers` 是一個切片。

訪問也可以調用沒有參數且返回單一值的簡單函數。示例 `example/container_resource_limit/model.yaml` 展示了這一點：

```
[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)
```

在這個匹配器中，`access(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , 0,
"Resources","Limits","cpu","Value")` 等同於 `r.obj.Request.Object.Spec.Template.Spec.Containers[0].Resources.Limits["cpu"].Value()`，其中 `r.obj.Request.Object.Spec.Template.Spec.Containers[0].Resources.Limits` 是一個映射，而 `Value()` 是一個沒有參數且返回單一值的簡單函數。

4.2.1.2 accessWithWildcard

有時候，您可能會有這樣的需求：陣列中的所有元素都必須有一個前綴 "aaa"。然而，Casbin 不支持 `for` 循環。通過 `accessWithWildcard` 和 "映射/切片擴展" 功能，您可以輕鬆實現這樣的需求。

例如，假設 `a.b.c` 是一個陣列 `[aaa, bbb, ccc, ddd, eee]`，那麼 `accessWithWildcard(a, "b", "c", "*")` 的結果將是一個切片 `[aaa, bbb, ccc, ddd, eee]`。通過使用通配符 `*`，切片被擴展了。

同樣地，萬用字元可以使用多次。例如，`accessWithWildcard(a, "b", "c", "*", "*")` 的結果將是 `[a.b.c[0][0], a.b.c[0][1], ..., a.b.c[1][0], a.b.c[1][1], ...]`。

4.2.1.3 支援可變長度參數的函數

在 Casbin 的表達式評估器中，當一個參數是陣列時，它將自動展開為可變長度參數。利用這一特性來支援陣列/切片/映射的展開，我們還整合了幾個接受陣列/切片作為參數的函數：

- `contain()`: 接受多個參數並返回是否任何參數（除了最後一個參數）等於最後一個參數。
- `split(a,b,c...,sep,index)`: 返回一個切片，包含 `[splits(a,sep)[index], splits(b,sep)[index], splits(a,sep)[index], ...]`。
- `len()`: 返回可變長度參數的長度。
- `matchRegex(a,b,c...,regex)`: 返回所有給定參數 (`a, b, c, ...`) 是否匹配給定的正則表達式。匹配給定的正則表達式。

這裡是一個範例，位於 `example/disallowed_tag/model.yaml`：

```

[matchers]
m = r.obj.Request.Namespace == "default" && r.obj.Request.Resource.Resource
=="deployments" && \
contain(split(accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers
, "*", "Image"),":",1) , p.obj)

```

假設 `accessWithWildcard(r.obj.Request.Object.Object.Spec.Template.Spec.Containers , "*", "Image")` 返回 `["a:b", "c:d", "e:f", "g:h"]`，因為 `splits` 支援可變長度參數並對每個元素執行分割操作，索引 1 的元素將被選擇並返回。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 類型轉換函數

- `ParseFloat()`: 將整數解析為浮點數（這是必要的，因為在比較中使用的任何數字都必須轉換為浮點數）。
- `ToString()`: 將對象轉換為字符串。此對象必須具有基本的字符串類型（例如，當有語句 `type XXX string` 時，類型為 `XXX` 的對象）。
- `IsNil()`: 返回參數是否為 `nil`。

5. 進階設定

5.1 關於證書

在 Kubernetes (k8s) 中，網鈎必須使用 HTTPS 是強制性的。實現這一點有兩種方法：

- 使用自簽署證書（本倉庫中的示例使用此方法）
- 使用普通證書

5.1.1 自簽署證書

使用自簽署證書意味著簽發證書的證書頒發機構（CA）不是眾所周知的 CA 之一。因此，您必須讓 k8s 知道這個 CA。

目前，本倉庫中的示例使用了一個自製的 CA，其私鑰和證書分別存儲在 `config/certificate/ca.crt` 和 `config/certificate/ca.key` 中。網鈎的證書位於 `config/certificate/server.crt`，由自製的 CA 發行。此證書的域名為 "webhook.domain.local"（用於外部網鈎）和 "casbin-webhook-svc.default.svc"（用於內部網鈎）。

CA 的相關信息透過網鈎配置文件傳遞給 k8s。`config/webhook_external.yaml` 和 `config/webhook_internal.yaml` 文件中均有一個名為 "CABundle" 的字段，該字段包含 CA 證書的 base64 編碼字符串。

若需要更改證書/域名（例如，當您想將此網鈎放入 k8s 的另一個命名空間中使用內部網鈎，或在使用外部網鈎時更改域名），應遵循以下步驟：

1. 生成新的 CA:

- 生成偽造 CA 的私鑰:

```
openssl genrsa -des3 -out ca.key 2048
```

- 移除私鑰的密碼保護:

```
openssl rsa -in ca.key -out ca.key
```

2. 生成網鈎服務器的私鑰:

```
openssl genrsa -des3 -out server.key 2048  
openssl rsa -in server.key -out server.key
```

3. 使用自生成的 CA 簽署網鈎的證書:

- 複製您系統的 openssl 配置文件以供臨時使用。您可以通過運行 `openssl version -a` 來找出配置文件的位置，通常稱為 `openssl.cnf`。
- 在配置文件中:
 - 找到 `[req]` 段落並添加以下行: `req_extensions = v3_req`
 - 找到 `[v3_req]` 段落並添加以下行: `subjectAltName = @alt_names`
 - 將以下行附加到文件中:

```
[alt_names]  
DNS.2=<The domain you want>
```

注意: 如果決定修改服務名稱，請將 'casbin-webhook-svc.default.svc' 替換為您自己的服務的真實服務名稱。

- 使用修改後的配置文件生成證書請求文件:

```
openssl req -new -nodes -keyout server.key -out server.csr -config openssl.cnf
```

- 使用自製的 CA 回應請求並簽署證書:

```
openssl x509 -req -days 3650 -in server.csr -out server.crt -CA ca.crt -CAkey
```

4. 替換 'CABundle' 字段： 請以新證書更新此欄位。

5. 若您使用 Helm，需對 Helm 圖表進行類似更改。

5.1.2 法律證書

若您使用法律證書，則無需經過所有這些程序。在 `config/webhook_external.yaml` 和 `config/webhook_internal.yaml` 中移除 "CABundle" 欄位，並將這些文件中的域名更改為您所擁有的域名。

Authorization of Service Mesh through Envoy

[Envoy-authz](#) 是一個 Envoy 的中間件，透過 casbin 執行外部 RBAC 和 ABAC 授權。此中間件使用 [Envoy 的外部授權 API](#) 透過 gRPC 伺服器。此代理可以在任何基於 Envoy 的服務網格上部署，例如 Istio。

需求

- Envoy 1.17+
- Istio 或其他類型的服務網格
- grpc 依賴

依賴項目使用 `go.mod` 進行管理。

中介層（Middleware）的工作原理

- 客戶端發出 HTTP 請求。
- Envoy 代理將請求發送至 gRPC 伺服器。
- gRPC 伺服器根據 Casbin 策略授權請求。
- 如果授權成功，請求將被轉發；否則，請求將被拒絕。

gRPC 伺服器基於 Envoy 中的 [external_auth.proto](#) 協議緩衝區。

```
// A generic interface for performing authorization checks on
```

從上述 proto 文件中，我們需要在授權伺服器中使用 `Check()` 服務。

使用方法

- 根據此 [指南](#) 在配置文件中定義 Casbin 策略。

您可以使用線上[casbin-editor](#)來驗證/測試您的政策。

- 透過運行以下指令來啟動認證伺服器：

```
go build .
./authz
```

- 載入Envoy配置：

```
envoy -c authz.yaml -l info
```

一旦Envoy啟動，它將攔截請求以進行授權處理。

與Istio整合

為了使此中介軟體正常運作，您需要在JWT令牌或標頭中包含用戶名的自訂標頭。您可以參考官方[Istio文件](#)以獲取更多關於修改 `Request Headers` 的資訊。



>

管理

管理



Admin Portal

Casbin 的管理員入口



Casbin Service

使用 Casbin 作為服務



Log & Error Handling

Casbin 中的日誌記錄和錯誤處理



Frontend Usage

Casbin.js 是一個 Casbin 附加元件，有助於在前端應用程式中進行存取控制管理。

Admin Portal

我們提供一個基於網頁的入口，稱為 [Casdoor](#)，用於模型管理和策略管理：

PML Model Editor ×

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

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

[role_definition]
g = _, _

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

[matchers]
m = r.tenant == p.tenant && g(r.sub, p.sub) && keyMatch(r.obj, p.obj) && (r.act == p.act || p.act == "*") && (r.service == p.service || p.service == "*")
```

Save

Rule Type	Tenant	User	Resource Path	Action	Service	Auth Effect	Option
p	tenant1	admin1	/*	*	*	allow	
p	tenant1	user12	/*	*	nova	allow	
p	tenant1	user13	/*	*	glance	allow	
g	user11	admin1					

Save

還有使用 Casbin 作為授權引擎的第三方管理員入口項目。您可以基於這些項目開始構建您自己的 Casbin 服務。

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

項目	作者	前端	後端	描述
Casdoor	Casbin	React + Ant Design	Beego	基於 Beego + XORM + React
go-admin-team/go-admin	@go-admin-team	Vue + Element UI	Gin	基於 Gin + Casbin + GORM 的 go-admin
gin-vue-admin	flipped-aurora	Vue + Element UI	Gin	基於 Gin + GORM + Vue

項目	作者	前端	後端	描述
gin-admin	@LyricTian	React + Ant Design	Gin	基於 Gin + GORM + Casbin + Ant Design React 的 RBAC 腳手架
go-admin	@hequan2017	無	Gin	基於 Gin + GORM + JWT + RBAC (Casbin) 的 Go RESTful API 網關
zeus-admin	bullteam	Vue + Element UI	Gin	基於 JWT + Casbin 的統一權限管理平台
IrisAdminApi	@snowlyg	Vue + Element UI	Iris	基於 Iris + Casbin 的後端 API
Gfast	@tiger1103	Vue + Element UI	Go Frame	基於GF (Go Frame) 的管理後台
echo-admin (前端, 後端)	@RealLiuSha	Vue 2.x + Element UI	Echo	基於Echo + Gorm + Casbin + Uber-FX的管理後台
Spec-Center	@atul-wankhade	無	Mux	基於Casbin + MongoDB的Golang RESTful平台

專案	作者	前端	後端	描述
spring-boot-web	@BazookaW	無	SpringBoot	基於SpringBoot 2.0 + MyBatisPlus + Casbin的管理後台
項目	作者	前端	後端	描述
node-mysql-rest-api	@JoemaNequinto	無	表達	一個用於構建RESTful API 微服務的模板應用程序，使用Node.js、Express、Sequelize、JWT和Casbin。
Casbin-角色管理-儀表板-RBAC	@alikhan866	React + Material UI	Express	適合初學者的RBAC管理，集成Enforcer以即時檢查執行結果
項目	作者	前端	後端	描述
fastapi-最佳架構	@吳氏家族	Vue + 阿爾科設計	FastAPI	基於FastAPI、SQLAlchemy、JWT和RBAC的管理門戶
fastapi-mysql-生成器	@碼農魅力	無	FastAPI	FastAPI + MySQL + JWT + Casbin
FastAPI-	@xingxingzaixian	無	FastAPI	FastAPI + MySQL +

項目	作者	前端	後端	描述
MySQL-Tortoise-Casbin				Tortoise + Casbin
openstack-policy-editor	Casbin	Bootstrap	Django	Casbin 的網頁使用者介面
專案	作者	前端	後端	描述
Tadmin	@leeqvip	AmazeUI	ThinkPHP	基於 ThinkPHP 的非侵入式後端框架
video.tinywan.com	@Tinywanner	LayUI	ThinkPHP	基於 ThinkPHP5 + ORM + JWT + RBAC (Casbin) 的 RESTful API 網關
laravel-casbin-admin	@pl1998	Vue + Element UI	Laravel	基於 vue-element-admin 和 Laravel 的 RBAC 權限管理系統
larke-admin (前端, 後端)	@deatil	Vue 2 + Element UI	Laravel 8	基於 Laravel 8、JWT 和 RBAC 的管理後台

專案	作者	前端	後端	描述
hyperf-vuetify-admin	@TragicMale	Vue + Vuetify 2.x	Hyperf	基於 Hyperf、 Vuetify 和 Casbin 的管理後 台

Casbin Service

如何使用 Casbin 作為服務？

名稱	描述
Casbin 伺服器	基於 gRPC 的官方 "Casbin 作為服務" 解決方案。 提供管理 API 和 RBAC API。
middleware-acl	基於 Casbin 的 RESTful 存取控制中介軟體。
認證伺服器	校對服務的認證伺服器。

Log & Error Handling

日誌記錄

Casbin 預設使用內建的 `log` 將日誌輸出到控制台，例如：

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

日誌記錄預設未啟用。您可以通過 `Enforcer.EnableLog()` 或 `NewEnforcer()` 的最後一個參數來切換它。

ⓘ 注意

我們已經支援在 Golang 中記錄模型、執行請求、角色和策略。您可以為 Casbin 的日誌記錄定義自己的日誌。如果您使用的是 Python，pycasbin 利用了 Python 的默認日誌記錄機制。pycasbin 套件會呼叫 `logging.getLogger()` 來設置記錄器。除了在父應用程式中初始化記錄器外，不需要特殊的記錄配置。如果在父應用程式中未初始化記錄，您將不會看到來自 pycasbin 的任何日誌訊息。同時，當您在 pycasbin 中啟用日誌時，它將使用預設的日誌配置。對於其他 pycasbin 擴展，如果您是 Django 用戶，可以參考 [Django 日誌文件](#)。對於其他 Python 用戶，您應該參考 [Python 日誌文件](#) 來配置記錄器。

為不同的執行器使用不同的記錄器

每個執行器都可以有自己的記錄器來記錄資訊，並且可以在運行時更改。

您可以透過 `NewEnforcer()` 的最後一個參數使用適當的記錄器。如果你使用這種方式

來初始化你的執行器，你不需要使用啟用參數，因為日誌器中啟用字段的優先級更高。

```
// 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)
```

支援的日誌器

我們提供了一些日誌器來幫助你記錄信息。

[Go](#) [PHP](#)

日誌器	作者	描述
預設日誌器 (內建)	Casbin	使用 golang 日誌的預設日誌器。
Zap 日誌器	Casbin	使用 zap , 提供 JSON 編碼的日誌，並且您可以通過自定義的 zap 日誌記錄器進行更多設置。

日誌記錄器	作者	描述
psr3-bridge 日誌記錄器	Casbin	提供符合 PSR-3 標準的橋接器。

如何編寫一個日誌記錄器

您的日誌記錄器應該實現 [Logger](#) 接口。

方法	類型	描述
啟用日誌()	強制性	控制是否打印消息。
是否已啟用()	強制性	顯示當前日誌器的啟用狀態。
日誌模型()	強制性的	記錄與模型相關的資訊。
LogEnforce()	強制性的	記錄與執行相關的資訊。
LogRole()	強制性的	記錄與角色相關的資訊。
LogPolicy()	強制性的	記錄與政策相關的資訊。

您可以將自訂的 `logger` 傳遞給 `Enforcer.SetLogger()`。

以下是如何為 Golang 自訂 logger 的範例：

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

// DefaultLogger is the implementation for a Logger using
// golang log.
type DefaultLogger struct {
```

錯誤處理

使用 Casbin 時可能會出現錯誤或恐慌，原因如下：

1. 模型文件 (.conf) 中的無效語法。
2. 政策文件 (.csv) 中的無效語法。
3. 來自存儲適配器的自訂錯誤，例如，MySQL 連接失敗。
4. Casbin 的 bug。

關於錯誤或恐慌，您可能需要了解五個主要功能：

功能	錯誤處理行為
<code>NewEnforcer()</code>	返回一個錯誤
<code>LoadModel()</code>	返回一個錯誤
<code>LoadPolicy()</code>	返回一個錯誤
<code>SavePolicy()</code>	返回一個錯誤
執行()	返回一個錯誤

ⓘ 注意

`NewEnforcer()` 內部調用 `LoadModel()` 和 `LoadPolicy()`。因此在使用 `NewEnforcer()` 時，您不需要調用後兩者。

啟用和禁用

可以通過 `Enforcer.EnableEnforce()` 函數禁用執行器。當它被禁用時，`Enforcer.Enforce()` 將始終返回 `true`。其他操作，如添加或移除策略，不受影響。以下是一個示例：

```
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 是一個 Casbin 附加元件，有助於在前端應用程式中進行存取控制管理。

安裝

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

或

```
yarn add casbin.js
```

前端中介軟體

中介軟體	類型	作者	描述
react-authz	React	Casbin	適用於 Casbin.js 的 React 封裝
rbac-react	React	@daobeng	使用 HOCs、CASL 和 Casbin.js 在 React 中實現基於角色的存取控制

中介軟體	類型	作者	描述
vue-authz	Vue	Casbin	適用於 Casbin.js 的 Vue 封裝
angular-authz	Angular	Casbin	適用於 Casbin.js 的 Angular 封裝

快速開始

您可以在前端應用程式中使用 `manual` 模式，並在您希望的任何時候設定權限。

```
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");
```

現在我們有一個授權者，`authorizer`。我們可以透過使用 `authorizer.can()` 和 `authorizer.cannot()` API 從中獲取權限規則。這兩個 API 的返回值是 JavaScript 的 Promise ([詳情在此](#))，因此我們應該像這樣使用返回值的 `then()` 方法：

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

`cannot()` API 的使用方式相同：

```
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
```

在上面的代碼中，參數中的 `success` 變量意味著請求在沒有拋出錯誤的情況下獲得了結果，並不意味著權限規則為 `true`。`failed` 變量也與權限規則無關。只有在請求過程中出現問題時，它才有意義。

您可以參考我們的 [React 實例](#) 來查看 Casbin.js 的實際使用情況。

權限對象

Casbin.js 將接受一個 JSON 對象來操作訪問者的相應權限。例如：

```
{  
  "read": ["data1", "data2"],  
  "write": ["data1"]  
}
```

上述權限物件顯示，訪客可以對 `data1` 和 `data2` 物件進行 `讀取`，而只能對 `data1` 物件進行 `寫入`。

進階用法

Casbin.js 提供了一個完美的解決方案，將您的前端存取控制管理與後端的 Casbin 服務整合在一起。

在使用 `自動` 模式並在初始化 Casbin.js `授權者` 時指定您的端點，它將自動同步權限並操作前端狀態。

```
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) => {
```

相應地，您需要公開一個介面（例如 RestAPI）來生成權限物件並將其傳遞給前端。在您的 API 控制器中，調用 `CasbinJsGetUserPermission` 來構建權限物件。以下是 Beego 的一個範例：

① 注意

您的端點伺服器應該返回類似以下的內容

```
{  
    "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()  
}
```

① 注意

目前，`CasbinJsGetPermissionForUser` API 僅在 Go Casbin 和 Node-

Casbin 中支援。如果您希望此 API 在其他語言中得到支援，請提出問題或在下方留言。

API 列表

setPermission(permission: string)

設置權限對象。總是在 `manual` 模式下使用。

setUser(user: string)

設置訪客身份並更新權限。總是在 `auto` 模式下使用。

can(action: string, object: string)

檢查用戶是否可以對 `object` 執行 `action`。

cannot(action: string, object: string)

檢查使用者是否無法對 `物件` 執行動作。

canAll(action: string, objects: Array<object>)

檢查使用者是否能對 `物件` 中的 **所有** 物件執行動作。

```
canAny(action: string, objects:  
Array<object>)
```

檢查使用者是否能對物件中的任一物件執行動作。

為什麼選擇Casbin.js

人們可能會想知道Node-Casbin和Casbin.js之間的區別。簡而言之，Node-Casbin是在NodeJS環境中實現的Casbin核心，通常作為伺服器端的存取控制管理工具包使用。Casbin.js是一個前端庫，幫助你在客戶端使用Casbin授權你的網頁用戶。

通常，由於以下問題，直接建立一個Casbin服務並在網頁前端應用程式中進行授權/執行任務是不恰當的：

1. 當有人啟動客戶端時，執行器將被初始化，並從後端持久層拉取所有策略。高併發可能會對數據庫造成巨大的壓力，並消耗大量網絡吞吐量。
2. 將所有策略加載到客戶端可能會帶來安全風險。
3. 難以將客戶端和服務器分離，同時促進敏捷開發。

我們需要一個工具來簡化在前端使用Casbin的過程。實際上，Casbin.js的核心是在客戶端操作當前用戶的權限。如你所說，Casbin.js從指定端點進行獲取。此過程將同步用戶的權限與後端Casbin服務。獲得權限數據後，開發人員可以使用Casbin.js接口來管理用戶在前端端的行為。

Casbin.js避免了上述兩個問題：Casbin服務將不再被反復拉起，客戶端與服務器之間傳遞的消息大小也減少了。我們也避免在前端儲存所有政策。使用者只能存取自己的權限，但對於存取控制模型和其他使用者的權限一無所知。此外，Casbin.js還能有效地在授權管理中解耦客戶端和伺服器。



>

編輯器

編輯器

**Online Editor**

在網頁瀏覽器中編寫 Casbin 模型和策略

**IDE Plugins**

Casbin IDE 插件

Online Editor

您也可以使用[線上編輯器](#)在您的網頁瀏覽器中編寫您的 Casbin 模型和策略。它提供了「語法高亮」和「代碼補全」等功能，就像程式語言的 IDE 一樣。

使用模式

如果您正在使用「帶有模式的 RBAC」或「帶有所有模式的 RBAC」，模式匹配函數在左下角指定。

The screenshot shows the Online Editor interface. On the left, a code editor displays Casbin configuration code. A red arrow points from the text 'keyMatch' in line 12 to the 'Request' pane on the right. The code is as follows:

```
12     *  
13         matchingDomainForGFunction:  
14             'keyMatch'  
15         */  
16         matchingForGFunction:  
17             'keyMatch2',  
18         matchingDomainForGFunction:  
19             'keyMatch2'  
20     };  
21 }());
```

On the right, a 'Request' pane shows three items:

- 1 /book/
- 2 /book/
- 3

如果您想編寫等效的代碼，您需要通過相關 API 指定模式匹配函數。更多資訊請參閱[帶有模式的 RBAC](#)。

① 注意

編輯器基於[node-casbin](#)。由於不同版本的 Casbin 之間存在同步延遲，「編輯器」的驗證結果可能與您所使用的 Casbin 版本的驗證結果不同。如果您遇到任

何問題，請將其提交至您所使用的 Casbin 倉庫。

IDE Plugins

我們提供以下IDE的插件：

JetBrains IDEs

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

VSCode

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



>

更多

更多



Our Adopters

Casbin's Adopters



Contributing

對Casbin的貢獻



Privacy Policy

Casbin 網站隱私政策



Terms of Service

Casbin 服務條款

 Refund Policy

Casbin 官方網站退款政策

Our Adopters

直接整合

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

名稱	描述	模型	政策
VMware Harbor	VMware 的開源可信雲原生註冊表項目，用於存儲、簽署和掃描內容。	代碼	Beego ORM
Intel RMD	Intel的資源管理守護程序。	.conf	.csv
VMware Dispatch	用於部署和管理無伺服器風格應用程序的框架。	代碼	代碼
Skydive	一個開源的即時網絡拓撲和協議分析器。	程式碼	.csv
Zenpress	一個用 Golang 編寫的 CMS 系統。	.conf	Gorm
Argo CD	適用於 Kubernetes 的 GitOps 持續交付工具。	.conf	.csv
木犀雲	木犀雲的PaaS平台，一種更簡單管理 Kubernetes 集群的方式。	.conf	代碼
工程師CMS	一個為工程師管理知識的CMS系統。	.conf	SQLite

名稱	描述	模型	政策
Cyber Auth API	一個用Golang編寫的認證API項目。	.conf	.csv
Metadata DB	BB檔案館的元數據資料庫。	.conf	.csv
Qilin API	ProtocolONE用於遊戲內容授權管理的工具。	代碼	.csv
Devtron Labs	Kubernetes 軟體交付工作流程。	.conf	Xorm
名稱	描述	模型	政策
lighty.io	OpenDaylight 的 SDN 控制器解決方案。	README	N/A
名稱	描述	模型	政策
Notadd	基於 Nest.js 的微服務開發架構。	.conf	資料庫適配器
ARC API	基於 Loopback 創建的微服務目錄，由 SourceFuse 開發。	使用方法	提供者
名稱	描述	模型	政策
dtrace	EduScaled 的追蹤系統。	提交	不適用

透過插件整合

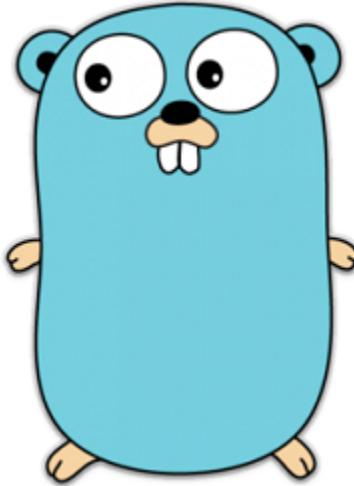
名稱	描述	插件	模型	政策
Docker	全球領先的軟體容器平台	casbin-authz-plugin (Docker推薦)	.conf	.csv
Gobis	Orange 的輕量級API網關，用Go編寫	casbin	程式碼	請求

Contributing

Casbin是一個強大的授權庫，支持多種訪問控制模型，並在許多編程語言中有實現。如果您精通任何編程語言，您可以為Casbin的開發做出貢獻。新貢獻者總是受到歡迎。

目前，主要有兩種類型的項目：

- **算法導向的項目** - 這些項目涉及在不同的編程語言中實現算法。Casbin支持廣泛的語言，包括Golang、Java、C++、Elixir、Dart和Rust，以及它們的相關產品。

		
Casbin		jCasbin
可投入生產		可投入生產



python™



PyCasbin

Casbin.NET

生產就緒

生產就緒

- **應用導向項目** - 這些項目與在 Casbin 之上構建的應用相關。

項目	示範	詳情	技能堆疊
Casdoor	Casdoor	Casdoor 是一個基於 OAuth 2.0/OIDC 的 UI 優先集中式認證/單一登入 (SSO) 平台。	JavaScript + React 和 Golang + Beego + SQL
Casnnode	Casbin 論壇	Casnnode 是一款新一代的論壇軟體。	JavaScript + React 和 Golang + Beego + SQL
Casbin OA	OA 系統	Casbin-OA 是 Casbin 技術作者的官方稿件處理、評估和展示系統。	JavaScript + React 與 Golang +

項目	示範	詳情	技能堆疊
			Beego + MySQL
Casbin 編輯器	Casbin 編輯器	Casbin-editor 是一個基於網頁的 Casbin 模型和策略編輯器。	TypeScript + React

參與其中

有許多方式可以為 Casbin 做出貢獻。以下是一些入門的想法：

- **使用 Casbin 並報告問題！** 在使用 Casbin 時，報告您遇到的任何問題，以促進 Casbin 的發展。無論是錯誤還是提案，建議在 [GitHub](#) 上提交問題。然而，在提出問題之前，最好先在 [Discord](#) 或 [GitHub 討論區](#) 進行討論。

注意：當報告問題時，請使用英文描述您的問題細節。

- **協助文件編寫！** 對文件的貢獻是開始參與的好起點。
- **協助解決問題！** 我們準備了一個表格，包含適合初學者的簡單任務，並以不同標籤標示不同難度等級。您可以查看該表格 [這裡](#)。

拉取請求

Casbin 使用 GitHub 作為開發平台，因此拉取請求是主要的貢獻方式。

在開啟拉取請求之前，有幾件事您需要知道：

- 解釋您發送拉取請求的原因以及它將為倉庫帶來什麼改變。

- 確保拉取請求只做一件事。如果有多次更改，請將其拆分為單獨的拉取請求。
- 如果你正在添加新文件，請在新文件的頂部包含 Casbin 許可證。

```
// 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 [Casdoor](#)、[Casnodel](#) 和 [Casbin OA](#) 等項目中，你可能需要設置一個演示，向維護者展示你的拉取請求如何有助於項目的開發。
- 當開啟拉取請求並提交你的貢獻時，建議使用以下格式的語義提交：
`<type>(<scope>): <subject>`。`<scope>` 是可選的。更多詳細用法，請參考 [Conventional Commits](#)。

許可證

通過向 Casbin 貢獻，你同意你的貢獻將在 Apache 許可證下授權。

Privacy Policy

您的隱私對我們很重要。 Casbin 的政策是尊重您的隱私，關於我們可能從您那裡收集的任何信息，包括我們的[文件網站](#)，以及我們擁有和運營的其他網站。

我們只在真正需要為您提供服務時才要求提供個人信息。我們以公平合法的方式收集，並在您知情和同意的情況下進行。我們也會告知您收集信息的原因以及將如何使用。

我們只保留所需的信息，以提供您要求的服務。我們存儲的數據將通過商業上可接受的手段進行保護，以防止丟失、盜竊以及未經授權的訪問、披露、複製、使用或修改。

我們不會公開分享任何個人識別信息，或與第三方分享，除非法律要求。

我們的網站可能會連結到非我們運營的外部網站。請注意，我們無法控制這些網站的內容和做法，因此無法對其各自的隱私政策承擔責任或責任。

您可以自由拒絕我們對您個人信息的要求，但請理解這可能使我們無法提供您部分所需服務。

您繼續使用我們的網站將被視為接受我們有關隱私和個人信息的實踐。如果您對我們處理用戶數據和個人信息的方式有任何疑問，歡迎隨時聯繫我們。

本政策自2020年6月29日起生效。

Terms of Service

1. 條款

當您訪問 <https://casbin.org> 網站時，即表示您同意遵守這些服務條款、所有適用的法律和法規，並同意您有責任遵守任何適用的當地法律。如果您不同意這些條款中的任何一項，則禁止您使用或訪問本網站。本網站所含的資料受適用的著作權和商標法保護。

2. 使用許可

a. 允許您暫時下載 Casbin 網站上的一份資料（信息或軟件）副本，僅供個人、非商業性的臨時查看使用。這是一份許可證的授予，而非所有權的轉讓，在這份許可證下，您不得：

- i. 修改或複製材料；
- ii. 將材料用於任何商業目的，或進行任何公開展示（無論是商業性或非商業性）；
- iii. 嘗試反編譯或逆向工程Casbin網站上的任何軟體；
- iv. 移除材料中的任何版權或其他專有標記；或
- v. 將材料轉移給另一個人或在任何其他伺服器上“鏡像”材料。

b. 本許可證將在您違反任何這些限制時自動終止，並且Casbin可隨時終止本許可證。在終止您查看這些材料或本許可證終止時，您必須銷毀您擁有的任何下載材料，無論是電子格式還是印刷格式。

3. 免責聲明

a. Casbin網站上的材料以“現狀”提供。Casbin不提供任何明示或暗示的保證，並在此明確否認並排除所有其他保證，包括但不限於對適銷性、特定用途適用性或不侵犯

知識產權或其他權利的默示保證或條件。

b. 此外，Casbin不保證或作出任何有關其網站上材料的使用準確性、可能結果或可靠性的陳述，也不保證或作出任何有關與本網站相關的材料或任何連結到本網站的網站的陳述。

4. 限制

在任何情況下，Casbin或其供應商均不對任何損害（包括但不限於因資料遺失或利潤損失，或因業務中斷）承擔責任，這些損害源自於使用或無法使用Casbin網站上的資料，即便Casbin或Casbin授權代表已口頭或書面通知此類損害的可能性。由於某些司法管轄區不允許對默示保證的限制，或對後果性或附帶損害的責任限制，這些限制可能不適用於您。

5. 資料的準確性

出現在Casbin網站上的資料可能包含技術性、打字或攝影上的錯誤。Casbin不保證其網站上的任何資料是準確、完整或最新的。Casbin可能隨時更改其網站上包含的資料，恕不另行通知。然而，Casbin並未承諾更新這些資料。

6. 連結

Casbin尚未審查所有連結至其網站的網站，並不對任何此類連結網站的內容負責。包含任何連結並不意味著Casbin對該網站的認可。使用任何此類連結網站需自行承擔風險。

7. 修改

Casbin可能隨時修訂其網站服務條款，恕不另行通知。使用本網站即表示您同意受當時有效的服務條款約束。

8. 管轄法律

這些條款和條件受舊金山，加利福尼亞州法律管轄並依其解釋，您不可撤銷地接受該

州或該地法院的專屬管轄。

Refund Policy

在大多數情況下，Casbin 訂閱的付款是不可退款的。

如果您遇到帳戶問題或認為在計費過程中出現了錯誤，請[聯繫客服](#)尋求協助。