

Ralasafe 开发实践



版本：Ralasafe 1.1 RC1

作者：汪金保

邮件：ralasafe@gmail.com

网站：www.ralasafe.cn

微博：weibo.com/ralasafe

日期：2011-6-22

内容目录

1.概述.....	5
2.集成安装.....	5
3.创建权限.....	6
3.1.基本实践步骤.....	6
3.2.权限创建.....	6
3.3.权限信息填写.....	7
3.4.权限组信息填写.....	8
3.5.权限删除.....	8
4.功能级权限.....	9
5.Ralasafe 与 WebRalasafe 区别.....	9
6.功能菜单树.....	10
7.数据级查询权限.....	11
7.1.实践案例.....	11
7.2.准备工作.....	11
7.3.编写业务代码.....	11
7.4.权限策略分析.....	13
7.5.用户分类策略定义及在线测试.....	13
7.5.1.编辑策略.....	13
7.5.2.测试策略.....	16
7.5.3.复制策略.....	17
7.6.数据查询定义及在线测试.....	19
7.6.1.编辑策略.....	19
7.6.2.测试策略.....	24
7.6.3.复制策略.....	25
7.7.编辑权限策略.....	25
7.8.总结.....	26
8.数据级决策权限.....	27
8.1.实践案例.....	27

8.2.准备工作.....	27
8.3.编写业务代码.....	27
8.4.权限策略分析.....	28
8.5.用户分类策略定义.....	28
8.6.数据查询定义——本人当天借款总额.....	29
8.7.业务数据分类策略定义及在线测试.....	30
8.7.1.编辑策略.....	30
8.7.2.测试策略.....	31
8.7.3.其他策略定制注意事项.....	32
8.8.编辑权限策略.....	33
8.9.总结.....	33
9.高级案例分析.....	33
9.1.高级案例分析 1.....	34
9.1.1.需求.....	34
9.1.2.实践策略.....	34
9.1.3.实践工作.....	35
9.2.高级案例分析 2.....	35
9.2.1.初始需求及对策.....	35
9.2.2.需求变更及对策.....	35
9.2.3.需求再变更及对策.....	36
9.2.4.需求再再变更及对策.....	37
10.安全策略命中机制.....	38
10.1.查询权限命中机制.....	38
10.2.决策权限命中机制.....	38
11.登录控制.....	39
12.Url 权限过滤.....	40
13.让 Ralasafe web 控制端安全.....	40
备注.....	40

1. 概述

Ralasafe 访问控制（数据级权限管理）中间件，非常友好易用，使用图形化方式对权限尤其是数据级权限进行管理。Ralasafe 使用 RBAC 模型对功能级权限进行管理；使用策略对数据级权限进行管理。策略定制过程完全图形化，策略定制完毕还可以在线测试，在线发布无需重启应用。

Ralasafe 使用 API 对应用提供权限管理服务。Ralasafe 是中间件不是框架，并不是通过编程框架来约束程序编写。当然，有了 API，应用开发者可以自行将其 wrap 到你的应用框架，或者封装成适应你框架的 AOP。

首先，我们必须摒弃一个误区：Ralasafe 并不是提供了很多种权限管理界面供你选择。其实提供权限管理界面，这是没有尽头的。世界上有数不清的权限管理需求，而且每时每刻都可能发生变化。

Ralasafe 提供图形化界面（Web 控制端）进行策略定制，使用策略来描述权限需求。Ralasafe 保证策略数据来源、策略计算方式，足以满足各种权限需求。

2. 集成安装

安装过程非常简单，

1. 添加工程 lib：把 `ralasafe-${version}-${build-date}.jar` 和引用的第三方包复制到你的工程 lib 目录下，做为你工程的 lib；
2. 复制 Web 内容：将 `WEB-CONTENT/ralasafe` 下面所有内容复制到你的工程的 Web 目录下；
3. 复制 `ralasafe` 配置信息：复制 `WEB-INF/ralasafe` 下面所有配置信息到你的 `WEB-INF/ralasafe` 目录下；
4. 合并 `web.xml`：将 `ralasafe-web.xml` 内容合并到你工程的 `web.xml` 文件里面（如果你是新建工程，直接将 `ralasafe` 发行包里面的 `web.xml` 复制过去，然后删除 `web.xml` 里面 `demo` 信息）；

5. 配置 ralasafe：参考《Ralasafe 配置手册》配置数据源、用户元数据和 XML 文件存储路径。

至此，一切 Ready！Go！！！！

3. 创建权限

3.1. 基本实践步骤

当我们构建软件系统时，我们肯定要知道系统有哪些地方需求权限控制，需要怎样的权限控制。

我们的构建策略步骤是：

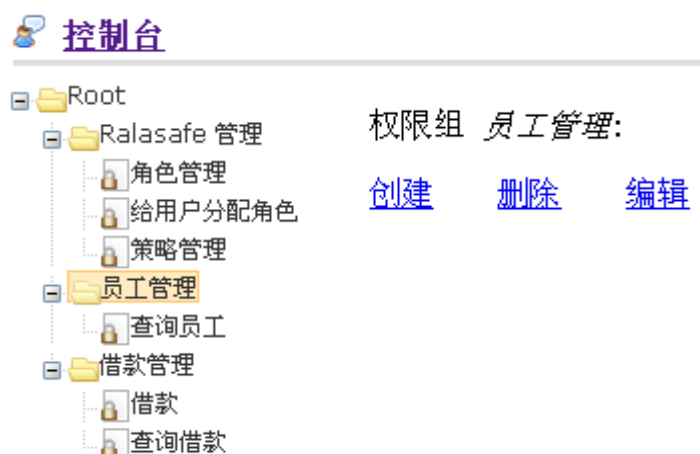
1. 创建权限
2. 做功能级权限控制
3. 做数据级权限策略定制

第一步是必须的，基础步骤；第二步和第三步可以分开。事实上，第二步一般在系统上线的时候，再进行角色定义、权限分配。

3.2. 权限创建

通过 Ralasafe 管理权限定义信息非常简单。打开 Ralasafe web 控制端（[http://localhost:8080/\\${context}/ralasafe/designer](http://localhost:8080/${context}/ralasafe/designer)），点击架构师下面的权限链接（[http://localhost:8080/\\${context}/ralasafe/privilege/privilegeMng](http://localhost:8080/${context}/ralasafe/privilege/privilegeMng)），进入权限管理界面。

权限以树形结构进行管理。点击每个节点，会在右边显示相应操作菜单。



点击组节点（非叶节点）时，会有创建、删除和编辑菜单。当我们需要创建新的权限或者权限组时，点击“创建”。

3.3. 权限信息填写

如果我们创建权限节点，我们需要填写以下信息：

权限组 员工管理:

[创建](#) [删除](#) [编辑](#)

组

☐ Yes ☒ No

名称

描述

常量名称

Target属性

URL

OK

1. 名称：权限名称

2. 描述：权限描述
3. 常量名称：给权限设置一个常量。权限常量的意义在于：调用 Ralasafe 各种权限方法时，不需要编写 int 数字（如 2），而该用常量引用（如 Constants.QUERY_EMPLOYEE）。Web 控制端可以集中导出所有常量，你可以将这些常量保存到你的常量类里面（比如 com.your.company.Constants）
4. Target 属性：指 html 链接的 target 属性
5. URL：指该权限对应的 web url，如果指定了 url，并配置了 org.ralasafe.webFilter.UrlAclFilter。那么该 Filter 会对这个 url 进行智能过滤，判断请求用户是否具有访问该 url 的功能权限

3.4. 权限组信息填写

如果要创建权限组，直接勾选组“yes”，然后输入名称和描述即可。

权限组 员工管理:

[创建](#) [删除](#) [编辑](#)

组

☒ Yes ☐ No

名称

描述

OK

权限组并没有任何权限意义，也并没有权限向上包含、向下包含等关系。它仅仅是方便权限树形结构管理的一个节点。

事实上，Ralasafe 也没有办法实现所谓向上包含、向下包含。因为，如你所知，这方面根本就没有什么标准，各种做法都有。而且这种包含关系极其容易让人混淆。

3.5. 权限删除

当删除权限时，系统会删除掉权限信息，权限与角色、与用户的关联信息，权限与数据级权

限的关联信息。

如果删除权限组，系统会级联对组下各个权限进行级联删除。

4. 功能级权限

功能级权限，Ralasafe 采用通用的 RBAC (基于角色的访问控制) 模型。Ralasafe 的功能级权限控制是可插拔的。你可以使用 Ralasafe 进行功能级权限控制，也可以不使用 Ralasafe 做功能级权限控制。

如果你不打算使用 Ralasafe 的功能级权限控制，那么你应该点击 Ralasafe web 控制端的非角色权限链接 ([http://localhost:8080/\\${context}/ralasafe/privilege/nonRolePrivilegeMng](http://localhost:8080/${context}/ralasafe/privilege/nonRolePrivilegeMng))。将所有权限设置为非角色权限，非角色权限只受数据级权限控制，不受功能级权限控制。

当你创建权限后，并且将角色付给用户。那么用户就拥有该权限了。你可以调用 Ralasafe API 判断用户是否对某功能是否有权限：

```
import org.ralasafe.Ralasafe;
import org.ralasafe.user.User;

boolean hasPrivilege=Ralasafe.hasPrivilege( privilegeId, user );
```

或者调用 WebRalasafe 进行功能权限判断：

```
import javax.servlet.http.HttpServletRequest;
import org.ralasafe.WebRalasafe;
import org.ralasafe.user.User;

boolean hasPrivilege=WebRalasafe.hasPrivilege( req, privilegeId );
```

当你调用 Ralasafe 的数据级权限 API (permit/query) 时，权限引擎会首先验证用户对该功能是否具有功能权限。因此调用 permit/query API 时，你无需显示调用 hasPrivilege 方法。

Ralasafe 引擎这么做的目的：**确保安全**，防止你的应用**忘记调用** hasPrivilege，就直接调用了数据级权限控制。

5. Ralasafe 与 WebRalasafe 区别

在进一步讨论之前，我们先探讨一下 Ralasafe 和 WebRalasafe 的区别。

它们的主要区别是：Ralasafe 传入 User 参数，而 WebRalasafe 传入 HttpServletRequest 参数。Ralasafe 权限引擎将自动从 HttpServletRequest 里面获得 Session 信息，然后从 Session 里面找

到当前请求用户信息。

所以，在你的应用程序的登录模块需要加入如下代码，将当前用户设置到 session 里面。

```
import javax.servlet.http.HttpServletRequest;
import org.ralasafe.WebRalasafe;
import org.ralasafe.user.User;

WebRalasafe.setCurrentUser( req, user );
```

如果使用 org.ralasafe.webFilter.LoginFilter，就不需要调用上述代码了。因为 LoginFilter 在登录验证成功后，已经调用了上述方法。

6. 功能菜单树

用户成功登录系统后，一般都会显示功能菜单。功能菜单显示风格千差万别。Ralasafe 能够提供菜单数据，即告诉应用——当前用户具有哪些菜单权限；然后应用程序按照需求自行显示。

调用 Ralasafe API 获取菜单树非常简单，返回可以递归的权限定义。如下是代码示例：

```
import java.util.Collection;
import java.util.Iterator;

import org.ralasafe.Ralasafe;
import org.ralasafe.privilege.Privilege;
import org.ralasafe.user.User;

Privilege rootPvlg=Ralasafe.getBusinessPrivilegeTree( user );

Collection children=rootPvlg.getChildren();
if( children!=null ) {
    for( Iterator iter=children.iterator(); iter.hasNext(); ) {
        Privilege pvlg=(Privilege) iter.next();

        if( pvlg.getIsLeaf() ) {
            // 这是个权限
            pvlg.getName();
            pvlg.getUrl();
            pvlg.getTarget();
        } else {
            // 这是个权限组
            pvlg.getChildren();
        }
    }
}
```

同样，你也可以调用 WebRalasafe 获得权限树。

7. 数据级查询权限

7.1. 实践案例

我们以 Ralasafe demo 应用的查询员工需求为例。（Demo 的员工和用户是同一张表，不少人问我“员工和用户有什么区别”。其实，我做 demo 时没有想这么多。想不通的同学，把注意力放在怎样做查询权限上，不要考虑什么区别）

查询员工的需求是，典型的层级查询：

1. 总公司用户查询所有员工；
2. 分公司用户查询当前用户所在分公司及下属营业部员工；
3. 营业部用户查询当前用户所在营业部员工。

具体哪个用户查询员工权限，由功能级权限控制。既创建角色——人力专员，赋予人力专员查询员工权限；然后将人力专员权限赋给相关用户（demo 赋给了贾洪亮、王胜利、汪来三位用户）。

7.2. 准备工作

我们首先要去 Ralasafe web 控制端创建一个权限，然后通过导出权限常量，将该权限常量保存到常量类里面。

此处，我们假设导入到常量类：demo.Constants，该类代码如下：

```
package demo;

public class Constants {
    public static final int QUERY_EMPLOYEE=2; //这个2是假设的数字，具体数字以导出的为准
}
```

7.3. 编写业务代码

搞定上面准备工作后，我们就可以编写业务代码了！

看客可能问：哈哈，等等，我们还没有分析权限怎样实现？就开始写代码？

答：对滴！这充分体现了权限与业务分离。权限逻辑不需要在业务代码里面体现，完全转移到 Ralasafe 里面。

首先，我们创建业务模型 Javabeen，代码如下：

```
package demo;

import java.util.Date;

public class Employee {
    private int id;
    private int companyId;
    private int departmentId;
    private String loginName;
    private String name;
    private String password;
    private String companyName;
    private String departmentName;
    private int isManager;
    private Date hireDate;

    // 各个字段的 get、set 方法
}
```

然后，在你的 Servlet 或者 Struts Action 里面调用 Ralasafe 查询 API，代码如下：

```
Collection employees = WebRalasafe.query(req, Constants.QUERY_EMPLOYEE);
req.setAttribute("employees", employees);

RequestDispatcher rd = req.getRequestDispatcher("employee.jsp");
rd.forward(req, resp);
```

接下来，直接编写 employee.jsp 页面，显示查询到的员工。相关代码如下：

```
<%
Collection employees = (Collection) request.getAttribute("employees");
%>

<%
for (Iterator iter = employees.iterator(); iter.hasNext();) {
    Employee employee = (Employee) iter.next();
%>

<tr >
    <td><%=employee.getName() %></td>
    <td><%=employee.getHireDate() %></td>
    <td><%=employee.getCompanyName() %></td>
    <td><%=employee.getDepartmentName() %></td>
    <td><%= (employee.getIsManager() == 1 ? "YES" : "NO") %></td>
</tr>
<% } %>
```

至此，业务代码编程完成！就是调用 Ralasafe query API，然后到界面显示。Query API 以指定的 demo.Employee 业务模型格式返回查询结果。

是的，你不需要编写 ORM query 代码！如果你真的想累一点，你可以 hack ralasafe，把

ralasafe query 的 where 条件抽取出来，拼接到你的 ORM 工具里面。

7.4. 权限策略分析

按照《Ralasafe Cookbook》讲解的查询权限模型，我们需要定义用户分类和数据查询。根据上述需求，要定义如下用户分类：

1. 总公司用户
2. 分公司用户（不需要定义北京分公司、上海分公司等，只要定义一个分公司用户）
3. 营业部用户（同样，只要定义一个营业部用户，无需定义具体某个营业部）

还要定义如下数据查询：

1. 查询所有员工
2. 查询所在分公司及下属营业部员工
3. 查询我所在营业部员工

对于第 2、3 个查询，权限引擎会自动将用户相关信息钻取出来，并设置到查询参数里面。

将上述 3 个用户分类和数据查询分配配对，赋给查询员工权限，就满足了我们的权限需求。

7.5. 用户分类策略定义及在线测试

注：因为 Ralasafe 的策略名称必须是唯一的。如果你使用 Ralasafe demo 环境操作，请自行将以下用户分类名称后面追加 “_test” 等字符，以防 Ralasafe 提示名称存在错误。

7.5.1. 编辑策略

在 Ralasafe web 控制端点击用户分类链接，进入用户分类管理页面。点击 root，在 root 下创建用户分类组——按组织机构分类；然后在该分类组下面定义上述用户分类。

首先，我们定义总公司用户。

1. 点击按组织机构分类，在此下面创建用户分类节点——总公司用户
2. 点击总公司用户，然后点击右边显示的菜单项——编辑策略，如图示：

用户分类组 总公司用户:

[复制](#)

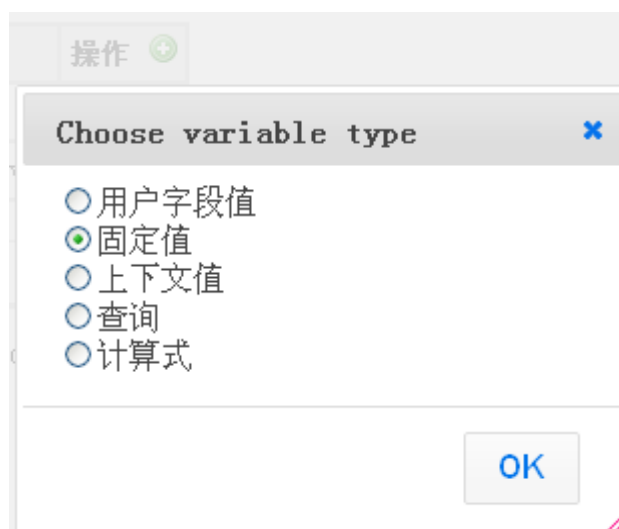
[删除](#)

[编辑](#)

[编辑策略](#)

[测试](#)

3. 在编辑策略页面的变量 tab 页，定义 2 个变量
4. 点击操作旁边的加号图片链接，在弹出的对话框中选择——固定值



5. 然后，在新弹出的对话框，输入如下值。表明变量名称是 headOfficeCompanyLevel，值等于 1，且是 java.lang.Integer 型



6. 点击 OK 按钮，关闭对话框

7. 点击操作旁边的加号图片链接，在弹出的对话框选择用户字段值
8. 输入变量名称 userCompanyLevel，值选择 companyLevel



9. 至此，变量定义完毕；下面我们定义表达式
10. 在表达式 tab 页，点击 Root Expr Group(AND)，在出现的菜单项中选择——新增二元表达式



11. 在弹出的对话框中，选择变量 1 headOfficeCompany，操作符 =，变量 2 userCompanyLevel



12. 点击 OK 按钮，关闭对话框

13. 至此，用户分类——总公司用户规则定义完毕，点击导航条的保存链接进行保存



7.5.2. 测试策略

通过 Ralasafe web 控制器设计好用户分类后，我们还可以在线测试用户分类策略正确与否。我们刚才已经点击了保存链接，对用户策略进行保存。Ralasafe web 控制器还支持保存之前进行测试。

下面，我们开始测试用户分类——总公司用户：

1. 点击导航条上面的测试链接，进入用户分类测试页面



2. 测试页面显示当前用户分类策略脚本（脚本是由引擎对刚才设计的表达式翻译），还有用户选择框。如果该用户分类策略的变量使用了上下文变量，测试页面还会自动显示输入上下文值的输入框
3. 点击选择，在弹出的用户选择框里面选择用户——贾洪亮，点击 OK 关闭对话框

4. 点击运行按钮，引擎将对贾洪亮用户进行解析计算，判断贾洪亮是否满足刚才定义的用户分类策略

脚本

```
1 | ( headOfficeCompanyLevel]
```

用户列:

☐
☐ 测试通过!

	姓名	公司
<input type="radio"/>	王朵朵	总部
<input type="radio"/>	王刚	总部
<input type="radio"/>	王胜利	上海分公司
<input type="radio"/>	齐明华	上海分公司
<input type="radio"/>	易斯	上海分公司
<input type="radio"/>	杨华	上海分公司
<input type="radio"/>	毕鑫鑫	合肥分公司

首页 上页 下页 末页 1~10 of 56 页: 1/6

5. 运行按钮下方的测试结果显示——测试通过，说明贾洪亮满足该用户分类策略，属于该用户分类——总公司用户
6. 你还可以选择一些分总公司用户测试，看看效果如何

7.5.3. 复制策略

因为用户分类——总公司用户、分公司用户和营业部用户非常类似，只是 companyLevel 分别等于 1、2、3 罢了。

下面，我们开始复制生成用户分类——分公司用户：

1. 点击刚才定制的用户分类——总公司用户

2. 在右边的菜单选项里面，选择复制链接



3. 在名称输入——分公司用户，点击 OK 按钮

名称

描述

OK

4. 在用户分类树会出现分公司用户节点，点击该节点，然后点击编辑策略链接

5. 在变量 tab 页面，点击 headOfficeCompanyLevel 变量后面的编辑图片链接

变量表达式

名称	类型	值	操作
headOfficeCompanyLevel	固定值	1	 
userCompanyLevel	用户字段值	companyLevel	

6. 将其值修改为 2

变量名

类型

Integer

值

OK

7. 至此，定义完毕可以点击导航条的保存链接，保存用户策略。但变量名使用

headOfficeCompanyLevel 不大合适，建议将变量名称修改为 branchCompanyLevel，这样更有业务意义。如果修改变量名称，我们还要对表达式进行重现编辑

8. 点击表达式 tab 页面，点击 headOfficeCompanyLeve=userCompanyLevel，点击编辑链接



9. 在弹出的对话框中，修改 branchCompanyLevel=userCompanyLevel

10. 点击导航条保存链接

按照以上步骤，你可以自行复制定义分公司用户分类。

7.6. 数据查询定义及在线测试

注：因为 Ralasafe 的策略名称必须是唯一的。如果你使用 Ralasafe demo 环境操作，请自行将以下数据查询名称后面追加 “_test” 等字符，以防 Ralasafe 提示名称存在错误。

7.6.1. 编辑策略

在 Ralasafe web 控制端点击查询链接，进入查询管理页面。点击 root，在 root 下创建查询组——查询员工；然后在该组下面定义上述数据查询。

我们先定义查询——查询所在分公司及下属营业部员工的查询策略。

1. 在组——查询员工下，创建数据查询——查询所在分公司及下属分公司员工
2. 点击查询树上的，查询所在分公司及下属分公司员工节点，然后选择右边菜单链接——编辑策略

查询 查询所在分公司及下属营业部员工:

[复制](#) [删除](#) [编辑](#) [编辑策略](#) [测试](#)

- 进入编辑策略后，在右边的数据源树上面，双击 3 张表——
demouser,company,department，查询员工是对这 3 张表进行联合查询



- 在右边编辑区域的映射类，输入前面我们定义的 demo.Employee，然后点击 OK

映射类

demo.Employee

- 然后我们点击 demouser 表的链接——选择所有，勾选 company 和 department 表的 name 数据列。看到效果了吗？Ralasafe 自动帮你把字段列和 demo.Employee 的 Java 属性做了匹配
- 我们修改 company 表的 name 数据列匹配关系（应该匹配 companyName，而不是 name），点击该列后面的编辑按钮

Schema: [Default], 数据库表: company, 别名: t1					
选择所有 取消全选 删除					
列	函数	Java 字段属性	只读	状态	操作
id <INT(11)>					<input type="checkbox"/>
name <VARCHAR(30)>		name <java.lang.String>	false		<input checked="" type="checkbox"/>
parentId <INT(11)>					<input type="checkbox"/> 编辑本字

- 在弹出的对话框中，选择 companyName，点击 OK 按钮

列
name <VARCHAR(30)>
函数

Java字段属性
companyName <java.lang.String>
只读
☐ True ☒ False
OK

8. 同样编辑 department 表的 name 数据列映射关系，使之与 departmentName 匹配
9. 下面，我们开始定义 where 条件，点击 where tab 页进入 where 编辑区域，我们要编辑的 where 条件伪代码是：where t0.companyId=t1.id **AND** t0.departmentId=t1.id **AND** (t1.id=\${user.id} **OR** t1.parentId=\${user.id})。第一层 where 的 3 个条件（t0.companyId=t1.id、t0.departmentId=t1.id 和 t1.id=\${user.id} **OR** t1.parentId=\${user.id}），以 AND 连接符连接；第 3 个条件是个条件组，使用 OR 连接符
10. 点击 Root expression group(AND)节点，然后点击菜单链接——新增二元表达式

Select Where Order Group

表达式

Root expression group (AND)

新增二元表达式
新增In表达式
新增Null判断
新增条件组
编辑

11. 在弹出的对话框中，上面选择 demouser[t0].companyId，下面选择 company[t1].id

列

demouser[t0].companyId

替换为上下文值
替换为用户属性值
替换为固定值

操作符

=

列

company[t1].id

替换为上下文值
替换为用户属性值
替换为固定值

OK

12. 重复上面步骤，定义 t0.departmentId=t2.id

13. 接下来，我们定义第三个表达式，这是一个嵌套的表达式组，连接符是 OR。点击 Root expression group(AND)节点，然后点击菜单链接——新增条件组

Select

Where

Order

Group

表达式

Root expression group (AND)
t0.companyId=t1.id
t0.departmentId=t2.id

新增二元表达式
新增In表达式
新增Null判断
新增条件组
编辑

14. 在弹出的对话框中，选择 OR 连接符，然后点击 OK 按钮

AND OR

OK

15. 接下来，我们点击刚才创建的 Expression group(OR)，在菜单项中选择——新增二元表达式链接



16. 在弹出的对话框中，上方选择 t1.id，下方先点击链接——替换用户属性值

列

company[t1].id

替换为上下文值

替换为用户属性值

替换为固定值

操作符

=

列

demouser[t0].id

替换为上下文值

替换为用户属性值

替换为固定值

17. 然后选择 companyId 字段，点击 OK，关闭对话框

18. 依照上面两步，定义 t1.parentId=\${user.companyId}

列

company[t1].id

替换为上下文值
替换为用户属性值
替换为固定值

操作符

=

用户字段值

companyId

替换为数据库列值
替换为上下文值
替换为固定值

OK

19. 完成以上步骤定义完成，我们可以立即保存数据查询，也可以先进行测试，测试通过后，再点击导航条的保存链接。

7.6.2. 测试策略

1. 点击导航条的测试链接，进入数据查询测试页面
2. 点击用户选择框，选择用户王胜利
3. 点击运行按钮，系统返回 27 条数据（真分页显示），下图是界面截图

SQL

```

1  SELECT  t0.id AS t0_id , t0.loginName AS t0_loginName , t
2  t0.password AS t0_password , t0.companyId AS t0_companyId
3  t0.departmentId AS t0_departmentId , t0.isManager AS t0_i
4  t0.hireDate AS t0_hireDate , t1.name AS t1_name , t2.name
5  FROM    demouser t0 , company t1 , department t2
6  WHERE   (    t0.companyId = t1.id    AND    t0.departmentId
7  t1.id = ?    OR    t1.parentId = ?    ) )

```

选择用户

王胜利

总记录数: 27

[下页](#)

id	loginName	name	password	companyId	departmentId	isManager	hireDate
9	杨华	杨华	password	2	1	0	2008-03-12
30	张希望	张希望	password	2	2	1	2008-09-23
31	刘恒士	刘恒士	password	2	2	0	2008-09-23

4. 点击导航条的保存链接，保存

7.6.3. 复制策略

和复制用户分类类似，点击刚才定制的查询——查询所在分公司和下属营业部员工，菜单项里面会有复制链接。点击后，提示输入查询名称和描述。

请读者自行复制出查询所有员工和查询我所在公司员工。（需要对 where 条件做适当调整）

7.7. 编辑权限策略

用户分类和数据查询已经定义好了，下面我们将其配对赋给权限查询员工。

1. 在 Ralasafe web 控制台点击权限链接，进入权限管理页面
2. 点击查询员工节点，然后点击菜单项中的编辑策略链接

权限 查询员工:

[删除](#) [编辑](#) [编辑策略](#) [测试](#)

3. 因为该权限还没有赋予过策略，系统会提示你这是查询权限还是决策权限。宣传查询权限，点击 OK 按钮

选择权限类型

☒ 查询权限 ☐ 决策权限

4. 在权限策略页面，点击操作旁边的加号图片链接
5. 在弹出对话框中，用户分类选择总公司用户，查询选择查询所有员工，点击 OK


增加策略 ✕
















用户分类

查询

描述

6. 重复上面 2 步，添加分公司用户与查询所在分公司及下属营业部配对策略；添加营业部用户与查询我所在公司员工配对策略。最后结果是

 [控制台](#) >> [权限](#) [保存](#) [测试](#)

用户分类	查询	描述	操作 
总公司用户	查询所有员工		    
分公司用户	查询所在分公司及下属营业部员工		    
营业部用户	查询我所在公司员工		    

7. 权限策略设置完成了，点击导航条的保存链接进行保存。
8. 你还可以点击测试链接进入测试页面，这里就不啰嗦了。

7.8. 总结

为了实现该数据级权限需求，我们在业务代码调用 Ralasafe query API 进行查询，然后直接在页面显示 demo.Employee 业务 Javabeen。所有权限逻辑在权限策略里面。

我们将权限策略分出 3 条，每条策略又有用户分类和数据查询组成。

定义好的用户分类，可以在多个权限策略里面复用；同样，数据查询也是可以在多个权限策略里面复用的。

将每条策略拆分成用户分类和数据查询，不仅提供了复用率，而去让策略更具有可读性！

8. 数据级决策权限

8.1. 实践案例

我们以 Ralasafe demo 的员工借款为例。借款的需求是：

1. 每次借款不能超过 5000 元
2. 每条借款总额不能超过 20000 元

8.2. 准备工作

在 Ralasafe web 控制端点击权限链接（[http://localhost:8080/\\${context}/ralasafe/privilege/privilegeMng](http://localhost:8080/${context}/ralasafe/privilege/privilegeMng)），进去权限管理页面。创建借款权限。

8.3. 编写业务代码

首先，我们编写业务 Javabean，代码如下：

```
package org.ralasafe.demo;

import java.util.Date;

public class LoanMoney {
    private int id;
    private int userId;
    private int money;
    private Date loanDate;

    //get和set方法
}
```

然后编写前台 jsp，让用户输入借款金额，代码如下：

```
<form id="appForm" method="post" action="loanMoney" />
<label>输入借款金额</label>
<input type="text" name="money" />
<input type="submit" />
</form>
```

然后在 servlet 层对用户是否具有借款权限做数据级判断，代码如下：

```
LoanMoney money=new LoanMoney();
money.setLoanDate( new Date() );

User user=WebRalasafe.getCurrentUser(req);
money.setUserId( ((Integer)user.get( User.idFieldName ) ).intValue() );

String strMoney=req.getParameter( "money" );
money.setMoney( Integer.parseInt( strMoney ) );

Map context=new HashMap();
context.put( "today", new java.sql.Date(System.currentTimeMillis()) );
if( WebRalasafe.permit( req, Privilege.LOAN, money, context ) ) {
    //调用你的dao或者service方法，将借款记录保存到数据库
}
```

支持业务代码编写完毕。如果你 WebRalasafe.permit API 如果返回 false 时，还会在 request 里面设置拒绝理由，你可以在页面显示这些拒绝理由。代码如下：

```
<%String denyReason=WebRalasafe.getDenyReason(request);
if ( denyReason!= null) { %>
<p><font color="red"><%=denyReason%></font></p>
<% } %>
```

8.4. 权限策略分析

按照《Ralasafe Cookbook》讲解的决策权限模型，我们需要定义用户分类和业务数据分类。然后将它们配对赋给借款权限。最终结果如下图示：

决策	用户分类	业务数据	拒绝理由	操作
拒绝	所有人	单笔借款大于5000	单笔借款上限是5000	    
允许	所有人	加上本笔当天借款额不超过20000	每天借款上限是20000	    

其中加上本笔当天借款额不超过 20000，还需要进行数据查询——查询用户当天已经借款总额。下面，我们开始策略定制：

8.5. 用户分类策略定义

所有人用户分类定义，策略非常简单。我们定义一个变量：固定值等于 a；表达式是：a=a。在数据级查询权限里面，已经详细探讨用户分类定义。此处，我们简单给出所有人用户分类相关截图：

<p>变量名</p> <div style="border: 1px solid #ccc; padding: 2px; width: 150px;">a</div> <p>类型</p> <div style="border: 1px solid #ccc; padding: 2px; width: 100px;">String ▼</div> <p>值</p> <div style="border: 1px solid #ccc; padding: 2px; width: 150px;">a</div>	<p>变量 1</p> <div style="border: 1px solid #ccc; padding: 2px; width: 50px;">a ▼</div> <p>操作符</p> <div style="border: 1px solid #ccc; padding: 2px; width: 50px;">= ▼</div> <p>变量 2</p> <div style="border: 1px solid #ccc; padding: 2px; width: 50px;">a ▼</div>
--	---

8.6. 数据查询定义——本人当天借款总额

对 loanmoney 表进行数据查询；只查询 money 数据列；映射类是 demo.LoanMoney；where 条件是 `userId=${user.id}` and `loanDate=${context.today}`。

其中有两点需要注意，相关操作如下：

1. 对 money 数据列做 sum 函数计算，获得总额。点击 money 数据列最右边的编辑图片链接

Schema: [Default], 数据库表: loan_money, 别名: t0
[选择所有](#) [取消全选](#) [删除](#)

列	函数	Java 字段属性	只读	状态	操作
id <INT(11)>					<input type="checkbox"/>
userId <INT(11)>					<input type="checkbox"/>
money <INT(11)>	sum	money <int>	false	<input checked="" type="checkbox"/>	

2. 在弹出的对话框里面，函数文本框输入 sum

列

money <INT(11)>

函数

sum

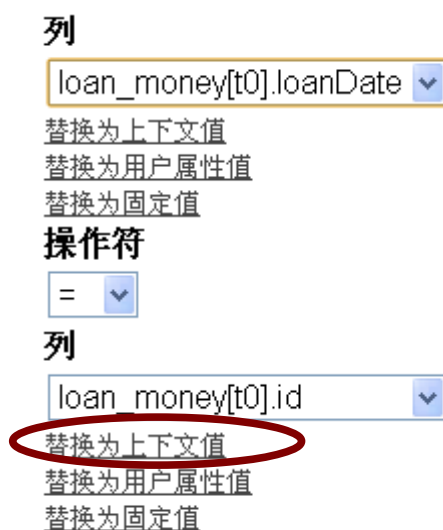
Java 字段属性

money <int> ▼

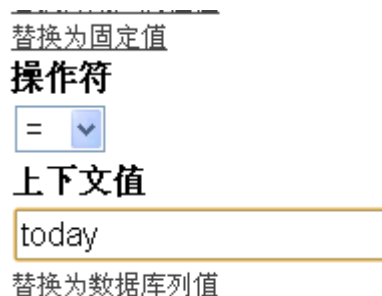
只读

☐ True ☒ False

3. where 条件中**上下文变量 today 的设置**，在 where tab 页面点击 Root expression group 链接后，点击二元表达式，然后在显示对话框中下方列点击**替换为上下文值**链接



4. 在输入框中输入 today



8.7. 业务数据分类策略定义及在线测试

下面我们开始定义业务数据分类——单笔借款大于 5000 元

8.7.1. 编辑策略

在变量 tab 页定义两个变量：

1. 固定值，变量名称（如果你觉得固定值和变量冲突，难以理解。你就将变量名称理解为固定值名称）limit；值 5000。相当于伪代码：int limit=5000;
2. 业务数据属性，变量名称 loanMoney，值 money。相当于伪代码：int loanMoney={\$businessData}.getMoney();（其中{\$businessData}是 permit 方法传入的参数）

名称	类型	值	操作
limit	固定值	5000	 
loanMoney	业务数据属性	money	 

在表达式 tab 页，定义表达式：loanMoney>limit。

8.7.2. 测试策略

点击导航条的测试链接，进入测试页面。

脚本

```
1 | ( loanMoney.compareTo(limit) > 0 )
```

业务数据

Class

属性	类型	值
money	null	<input type="text"/>

1. 在业务数据 Class 输入框，输入业务类名称：demo.LoanMoney，这就是前面业务代码里面构建的 javabean 实体类名称
2. 点击 load 按钮，money 的属性由 null 变成 int，同时值输入框变成可编辑状态

业务数据

Class

属性	类型	值
money	int	<input type="text" value="455"/>

3. 输入 455，点击运行。系统提示测试失败，money=455 的 javabean LoanMoney 不属于该业务数据分类。这显然是符合我们既定目标的。

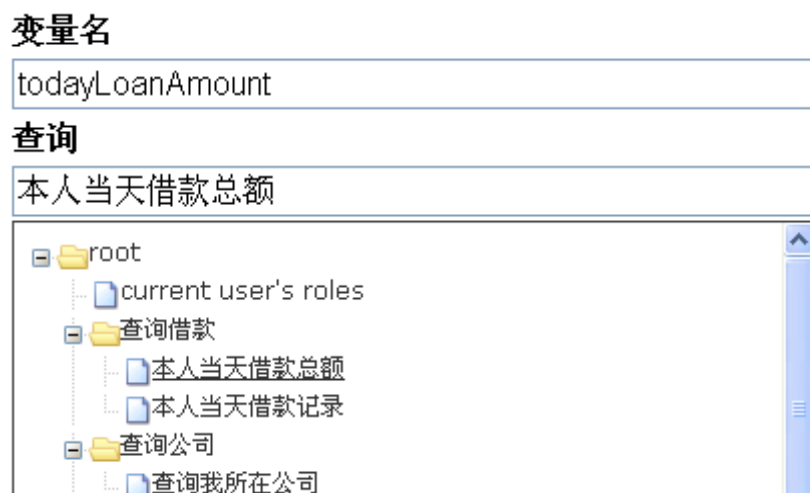
8.7.3. 其他策略定制注意事项

当我们定义业务数据分类——加上本笔当天借款总额不超过 20000 元时，这个好像有点复杂，不好定制。先来分析一下该分类的策略伪代码：

```
// 当天借款总上限
int limit=20000;
// 今天已借款总额
int todayLoanAmount=select sum(money) from loanmoney
                        where userid=${user.id} and loanDate=${context.today};
// 今天余下可借款最高额
int todayLeftLimit=limit-todayLoanAmount;
// 当前要借款的额度
int money=businessData.getMoney();

return money<=todayLeftLimit;
```

第二个变量就是前面定义的数据查询——本人当天借款总额。因此，在变量 tab 页面，定义数据查询类型变量，在查询输入框选择——本人当天借款总额。如下图所示：



第三个变量属于计算式，也可以在变量 tab 页面定义。选择计算式类型，然后选择第一个变量名称：limit，第二个变量名称：todayLoanAmount，计算符选择减号。如图所示：

变量名

todayLeftLimit

变量 1

limit

操作符

-

变量 2

todayLoanAmount

返回类型

integer

8.8. 编辑权限策略

至此，所有策略定义工作完成，下面我们将它们配对赋给借款权限。操作与给查询权限赋予权限策略类似，在此不做详述。最后权限策略结果，如下图所示：

决策	用户分类	业务数据	拒绝理由	操作
拒绝	所有人	单笔借款大于5000	单笔借款上限是5000	<div><div></div><div></div><div></div><div></div><div></div><div></div></div>
允许	所有人	加上本笔当天借款额不超过20000	每天借款上限是20000	<div><div></div><div></div><div></div><div></div><div></div><div></div></div>

8.9. 总结

为了实现该数据级权限需求，我们在业务代码调用 Ralasafe permit API 进行决策判断，如果判断通过调用 dao 或者 service 将数据同步到后台；如果拒绝，业务代码进行流转，同时 Ralasafe 也告知拒绝理由（使用 Ralasafe permit 返回 Decision 含有 denyReason 属性；使用 WebRalasafe permit，则调用 WebRalasafe.getDenyReason(request)获得拒绝理由）。所有权限逻辑在权限策略里面。

整个过程图形化，而且可在线验证！

9. 高级案例分析

前面探讨的 2 个案例，比较常规，主要是用来告诉大家怎样使用 Ralasafe API 编程，如何区分 Ralasafe 和业务编码界限。接下来，我们探讨几个复杂案例，重点阐述如何使用 Ralasafe

实践复杂权限管理需求。

9.1. 高级案例分析 1

9.1.1. 需求

某大型集团公司财务账单审核功能。正常情况下，是按照层级机构查询账单。但有的时候，会临时组成调查组，比如抽调广西省的张三、大连市的王五对河北省财务账单进行审核。如果按照正常逻辑，张三看广西省所有财务账单；王五看大连市所有财务账单。那么，如何实现查询财务账单功能呢？

9.1.2. 实践策略

实践准则依然是：权限与业务分离。权限方面：逐步梳理，要做到责权分离。

对案例进行分析，可以看出：

1. 如果当前用户临时被抽调，那么他/她按照抽调指定的地方查询财务账单
 1. 引出临时调查组用户分类（不要继续向下扩展，引出所谓北京调查组、深圳调查组等，除非有必要）
 2. 调查组用户分类应该执行的查询是：select * from where orgnId=指定的机构 id
2. 否则，按照层级机构查询财务账单
 1. 直接使用常规的层级查询即可，本文的数据级查询权限就实践了常规层级查询

总体来看，该需求的查询权限策略应该如下：

策略序号	用户分类	查询
1	临时调查组用户	执行调查组指定的机构账单
2	总公司用户	查询所有账单
3	分公司用户	查询所在分公司账单
4	营业部用户	查询营业部账单

9.1.3. 实践工作

所以，我们要创建一张表——临时调查组表；然后开发该表的基本信息管理功能，即——用户抽调管理功能——让用户通过界面将广西省的张三抽调到河北省，大连市的王五抽调到河北省。

临时调查组（SPEC_USER）表结构基本如下：

字段名称	字段意义
userId	用户 id，被抽调的用户 id 标识
organId	机构 id，用户被派遣到哪个机构 id，如前面说的河北省

临时调查组用户的权限策略伪代码：`${user.id} in ${select userId from SPEC_USER where userid=${user.id}}`

执行调查组指定的机构订单：

`select ... from BILL a, SPEC_USER b where a.orgId=b.orgId and b.userId=${user.id}`

9.2. 高级案例分析 2

实践准则依然是：权限与业务分离。权限方面：逐步梳理，要做到责权分离。

9.2.1. 初始需求及对策

我们先看看初始需求，订单审查权限是：

1. 普通审查员审查小于 100 万的订单
2. 高级审查员审查 100 万及以上的订单

很简单，我们创建这样的决策权限策略即可：

序号	决策	用户分类	业务数据分类	拒绝理由
1	允许	普通审查员	小于 100 万的订单	普通审查员只能审查小于 100 万的订单
2	允许	高级审查员	100 万及以上订单	高级审查员不能审查小于 100 万的订单

9.2.2. 需求变更及对策

因为地域经济差异，北京和陕西的订单审查界限都是 100 万不合理。北京的 100 万订单非常

多；陕西的 100 万订单又比较少。

客户将需求变更为：

1. 北京普通审查员审查 400 万以下订单
2. 北京高级审查员审查 400 万及以上订单
3. 陕西普通审查员审查 80 万以下订单
4. 陕西普通审查员审查 80 万及以上订单
5.

为了实现以上需求，我们无需更改业务代码，只需调整 Ralasafe 的权限策略即可：

序号	决策	用户分类	业务数据分类	拒绝理由
1	允许	北京普通审查员	小于 400 万的订单
2	允许	北京高级审查员	400 万及以上订单
3	允许	陕西普通审查员	小于 80 万订单
4	允许	陕西高级审查员	80 万及以上订单

9.2.3. 需求再变更及对策

以上实现方式，将 80 万、400 万固化到安全策略里面，由软件提供商或者企业 IT 管理员设置。客户认为 80 万、400 万，属于业务支撑概念。因此，需要需要变更为：业务管理员自行设置 80 万、400 万阈值。

为了实现以上需求，正好**实践了责权分离的原则**。非常简单，策略反而更加简单了！

首先，我们要创建一张表用来保存各个地区的阈值 (CREDIT_LINE) 表结构如下：

字段名称	字段意义
areaId	地区 id 标识
limit	阈值

然后，我们还要开发一个管理界面，供各个地区的业务管理员对阈值进行维护。

权限策略也要做相应变更：

序号	决策	用户分类	业务数据分类	拒绝理由
1	允许	普通审查员	小于 阈值表指定金额 的订单	普通审查员只能审查小于阈值表指定以下订单（如果嫌晦涩，就写：你无权审查该订单
2	允许	高级审查员	阈值表指定金额 及以上订单	高级审查员只能审查阈值表指定金额及以上订单（如果嫌晦涩，就写：你无权审查该订单

其中业务数据需要引用数据查询：

1. 小于阈值表指定金额的订单策略伪代码是：billAmount<(select limit from CREDIT_LINE where areaId=\${user.areaId})
2. 阈值表指定金额及以上订单策略伪代码是：billAmount>=(select limit from CREDIT_LINE where areaId=\${user.areaId})

9.2.4. 需求再再变更及对策

刚才谈到 80 万、400 万这样的阈值由业务管理员自行控制。总公司的领导认为：他们控制可以，但必须有个度，不能超出某个范围。比如北京的阈值应该在 300~500 万之间，陕西的阈值应该是 50~150 万之间。

需求变更如下：

1. 为每个地区增加额度限制权限，不能随意设置额度

这就出现了给权限系统再设置权限的问题了！看起来非常复杂，但用了 Ralasafe，并进行一些思维转换，一切变的简单。我们的策略应该调整为：

1. 原来的订单审核决策权限维持不变
2. 为订单阈值设置功能，增加一个权限控制，而且是决策权限。即当业务管理员输入某个数字时，Ralasafe 权限引擎给予判断是否运行该业务管理设置该数字。场景举例：
当北京业务管理员输入 100000 万时，Ralasafe 权限引擎将提示权限不足，该维护功能的 Servlet/Action 不予调用后台阈值增加/修改方法

阈值设置的决策权限策略应该是：

序号	决策	用户分类	业务数据分类	拒绝理由
1	允许	北京用户	区间在 300~500 万之间的阈值
2	允许	陕西用户	区间在 50 ~ 150 万之间的阈值

业务数据分类策略伪代码是：limit<=5000000 and limit>=3000000。

(你也可以将 300、500、50、150 万这样的阈值从策略里面抽取出来，放到数据库里面。这和上一步的需求变更比较类似。)

10. 安全策略命中机制

10.1. 查询权限命中机制

先讨论查询权限的安全策略命中机制。假设查询订单权限，分配有如下安全策略：

查询订单安全策略：

1. 用户分类 A —— 数据查询 1
2. 用户分类 B —— 数据查询 2
3. 用户分类 C —— 数据查询 3

Ralasafe 权限引擎会逐条对安全策略进行检验，直到命中某条策略。首先检验当前用户是否满足用户分类 A 的规则，如果满足，则执行数据查询 1，并返回查询结果；否则，检验当前用户是否满足用户分类 B 的规则，如果满足，则执行数据查询 2，并返回查询结果；否则，检验当前用户是否满足用户分类 C 的规则，如果满足，则执行数据查询 3，并返回查询结果。如果当前用户均不满足以上用户分类规则 A/B/C，则不执行任何数据查询，直接返回空集合 (size=0 的集合，不是 NULL)。

如果，当前用户既满足用户分类 A 的规则，也满足用户分类 B 的规则。因为，用户分类 A 对应的安全策略在用户分类 B 对应的安全策略前面，被权限引擎优先执行，所以返回数据查询 1 的查询结果。

10.2. 决策权限命中机制

再来讨论决策权限的安全策略命中机制。假设修改订单权限，分配有如下安全策略：

1. 用户分类 A ———— 订单类型 1 ———— 允许 ———— “你不允许修改类型 1 订单”
2. 用户分类 B ———— 订单类型 2 ———— 拒绝 ———— “你不允许修改类型 2 订单”
3. 用户分类 C ———— 订单类型 3 ———— 允许 ———— “你不允许修改类型 3 订单”

Ralasafe 权限引擎会逐条对安全策略进行检验，直到命中某条策略。首先检验当前用户是否满足用户分类 A 的规则，和当前订单是否满足订单类型 1 的规则，如果都满足，则返回允许，如果有一个不满足，继续检验下一条策略；检验当前用户是否满足用户分类 B 的规则，和当前订单是否满足订单类型 2 的规则，如果都满足，则返回拒绝，并将如果有一个不满足，继续检验下一条策略；检验当前用户是否满足用户分类 C 的规则，和当前订单是否满足订单类型 3 的规则，如果都满足，则返回允许。因为没有其他安全策略了，直接返回拒绝。

当权限引擎返回拒绝时，还会告知拒绝理由。

如果当前用户和当前订单，同时满足某条行为为“拒绝”的策略（如上面第 2 条），那么拒绝理由就是该条策略的拒绝理由；否则，将用当前用户满足的用户分类规则，但不满足业务数据分类规则的安全策略拒绝理由做为拒绝理由返回（可能有多条理由）；如果当前用户不满足任何一条安全策略，直接返回拒绝理由“您没有操作权限”。

例如：

1. 如果当前用户满足用户分类 A，当前订单满足订单类型 1，返回允许
2. 如果当前用户满足用户分类 A，当前订单不满足订单类型 1，且当前用户满足用户分类 C，也不满足订单类型 3，则返回“拒绝”，拒绝理由是“你不允许修改类型 1 订单”和“你不允许修改类型 3 订单”；
3. 如果当前用户满足用户分类 B，当前订单满足订单类型 2，则直接返回“拒绝”，拒绝理由是“你不允许修改类型 2 订单”。

看起来比较复杂，道理其实非常简单。如果当前用户满足某条用户分类的规则且当前业务数据满足对应的业务数据分类的规则，则直接返回该策略的行为：允许或者拒绝。如果是拒绝，返回该策略的“拒绝理由”。

当前用户和当前业务数据不能命中任何任何策略时，返回拒绝。拒绝理由分 2 种情况：

1. 如果当前用户不满足任何策略用户分类的规则，则直接返回拒绝，拒绝理由是“您没有操作权限”；
2. 否则，返回这些安全策略的拒绝理由：当前用户能满足该安全策略的用户分类的规则。

11. 登录控制

org.ralasafe.webFilter.LoginFilter 能够实现登录控制。详见 JavaDoc。（过几天详细写）

12. Url 权限过滤

org.ralasafe.webFilter.UrlAclFilter 能够实现 URL 权限过滤，如果用户没有权限访问对应页面，会重定向到拒绝页面。详见 JavaDoc。（过几天详细写）

13. 让 Ralasafe web 控制端安全

1.1 版本的 Ralasafe 各个功能基本都可以使用 URL 访问到。在系统上线时，需要对各个 URL 做权限控制，禁止未授权用户访问。

所以，用 URL 权限过滤就能实现 Ralasafe 控制端安全！

实践步骤：

1. 为各个 URL 创建权限
2. 创建角色，赋予对应权限
3. 然后将权限赋给指定用户

这样，就实现了 Ralasafe web 控制端的安全！

（过几天详细写）

备注

1. Ralasafe web 控制端地址：[http://localhost:8080/\\${context}/ralasafe/designer](http://localhost:8080/${context}/ralasafe/designer)
2. 一般没有特意标注下，点击指左击。Ralasafe 1.1 版本起，几乎只有左击和双击。
3. 安全策略：用户分类策略、业务数据分类策略、数据查询策略和权限策略，我们习惯通称为安全策略。有时，我们会不经意地将权限策略称为安全策略。