# 总体说明

新版IAM 的权限控制，不再基于角色与URL的映射进行权限检查，而是采用策略的模式。

策略就是在一定条件下（Conditions），某些服务/资源的某些操作的权限。

将策略与用户关联，则用户将拥有策略所代表的特定服务资源的特定操作权限。

将策略与角色关联，则拥有该角色的用户将拥有策略所代表的特定服务资源的特定操作权限。

将策略与用户组关联，则该用户组下的用户将拥有策略所代表的特定服务资源的特定操作权限。

将策略与Client关联，则该Client将拥有策略所代表的特定服务资源的特定操作权限。

策略中包含几个要素：resource、action、conditions

Resource：代表某些服务和资源。

Action：代表服务资源的操作

Conditions：表示限定条件，该条件成立时，上述才拥有上述服务资源的操作权限。

关于策略更多内容，参见[预设策略与授权](#_预设策略与授权)。

# 如何引入新版Adapter

## 去除对keycloak adapter的直接依赖

**1、在 build.gradle 文件中，删除所有包含 keycloak 的配置，如：**

|  |
| --- |
| buildscript {  ext {  **keycloak**Version = '4.2.0.Final'  }  }  dependencies {  compile('org.keycloak:**keycloak**-spring-boot-starter')  }  dependencyManagement {  imports {  mavenBom "org.keycloak.bom:**keycloak**-adapter-bom:${keycloakVersion}"  }  } |

**2、删除有关keycloak的一些配置类**

KeycloakClientRequestFactoryProd

KeycloakClientRequestFactoryDev

SecurityConfig

SecurityConfigDev

## 加入对IAM Adapter的直接依赖

在 build.gradle文件中，引入依赖：

compile('com.inspur.iam:iam-adapter-java:0.0.1-SNAPSHOT')

执行命令 gradlew cleanEclipse eclipse --refresh-dependencies 刷新本地依赖。

## 加入扫描

将com.inspur.iam加入到应用的扫描范围。

# IAM Adapter相关配置

以 **application.properties**为例

|  |
| --- |
| #dev prod，dev模式下，adapter会往上下文中放置模拟用户信息，并且所有接口都不受权限控制  spring.profiles.active=prod  #IAM API Server地址  iam.url=http://service.inspurtest.com/auth  #IAM ladon地址  iam.ladon.url=${iam.url}  #########################  #region 配置  common.region=cn-north-3  #是否禁用  keycloak.disable-trust-manager=true  #keycloak地址  keycloak.auth-server-url=https://iopdev.10.110.25.123.xip.io/auth  #使用的keycloak的Realm名称  keycloak.realm=picp  #使用的keycloak的clientId  keycloak.resource=iam-apiserver  #上面clientId对应的secret  keycloak.credentials.secret=49a66244-1d41-4605-a4e9-42c854489fc9  #是否纯后端微服务  keycloak.bearer-only=true  #是否接受跨域请求的相关配置  keycloak.cors=true  keycloak.cors-max-age=1728000  keycloak.cors-allowed-headers=Authorization, Origin, X-Requested-With, Content-Type, X-Custom-Header  keycloak.cors-allowed-methods=POST,PUT,DELETE,GET,OPTIONS,PATCH  #配置需要透传的到RestTemplate的cookie  resttemplate.cookie=realm\_roles,username  #应对防盗链检查  resttemplate.referer=https://console1.cloud.inspur.com/  spring.jackson.date-format=yyyy-MM-dd'T'HH:mm:ss'Z'  # Asia/Shanghai UTC ...  spring.jackson.time-zone=GMT  #true false  spring.jackson.serialization.write-dates-as-timestamps=false |

**测试环境集群内配置**

iam.url=http://iam-apiserver.iam.svc.cluster.local:9101/auth

iam.ladon.url=http://iam-auth-ladon.iam.svc.cluster.local:8090/auth

**测试环境本地开发配置**

iam.url= http://service.inspurtest.com/auth

iam.ladon.url= http://10.110.25.71:48156/auth

# 服务定义

**要不要与产品线编码保持一致？**

每个Springboot应用，

从业务功能划分，必定属于某个特定的服务，

所有的服务名都需要预先初始化到IAM系统中。

**服务名命名规范**

* 全小写
* 单词缩写

如 iam, rds, oss, iot 等。

服务定义完成，并评审通过后，纳入到IAM-ApiServer的初始化SQL。

数据准备：

INSERT INTO iam\_service (`id`, `description`, `display\_name`, `**name**` ) VALUES ( 'iam', 'iam', 'iam', **'iam'** );

属性说明：

"name": "oss", //服务名

"displayName": "对象存储服务", //展示名称

"description": "对象存储服务", //描述

# 资源类型定义

一般与Entity对应。

## 规范

* 全小写
* 单词缩写
* **同一个服务下的资源类型名称不允许重复**。

## 示例

rds服务下的资源类型可能包括：mysql、oracle、db2

iam服务下的资源类型可能包括：user、role、usergroup

## 数据准备

INSERT INTO iam\_resource\_type (`id`, `**name**`, `display\_name`, `description`, `service\_id`, `context` ) VALUES ( 'user', **'user'**, '用户', '用户', 'iam', 'name' );

属性说明：

context：指定当前资源类型的那些属性可以作为鉴权上下文使用，定义策略condition和权限检查时使用，可以基于context中的服务属性授权。

# 操作定义

可以为每个服务定义一系列操作，

每个服务的操作也需要预先初始化到IAM系统中。

**操作名命名规范**

* 参照Java的类命名规范，首字母大写，每个单词的首字母大写。
* 动宾结构，动词 + 名词
* **同一个服务下的操作名称不允许重复**。

**可选动词：**

CreateXXX 新建XXX

UpdateXXX 更新XXX

DeleteXXX 删除XXX

GetXXX 查看XXX

ListXXX 展示XXX列表

StartXXX 启动XXX

StopXXX 停止XXX

BindXXX 绑定XXX

UnbindXXX 解绑XXX

SyncXXX 同步XXX

ImportXXX 导入XXX

ExportXXX 导出XXX

优先使用上述预定义的动词，

如不能满足需要，可自行定义。

**示例：**

CreateImage 创建镜像

DeleteImages 删除镜像

操作定义完成，并评审通过后，纳入到IAM-ApiServer的初始化SQL

数据准备

INSERT INTO iam\_action (`id`, `description`, `display\_name`, `**name**`, `resource\_type\_id`, `service\_id` ) VALUES ( 'iam\_create\_user', null, '创建用户', 'CreateUser', 'user', 'iam' );

# 预设策略与授权

## 预设策略

每个服务可以根据权限控制需求和粒度，预先定义一些策略，

预设授权，以及租户对子用户授权，都是基于策略进行授权。

**策略格式如下：**

|  |
| --- |
| {http://www.kjson.com/static/js/jsonfmt/images/Expanded.gif  "**resources**": [http://www.kjson.com/static/js/jsonfmt/images/Expanded.gif  **"irn:{服务名}:{Region}:{所属账号}:{资源}",**  ],  "**actions**": [http://www.kjson.com/static/js/jsonfmt/images/Expanded.gif  "{服务名}:{操作名}",  "{服务名}:{操作名}",  ……  ],  "**conditions**": {http://www.kjson.com/static/js/jsonfmt/images/Expanded.gif  "{鉴权上下文中的key}": {http://www.kjson.com/static/js/jsonfmt/images/Expanded.gif  "type": "StringEqualCondition",  "options": {http://www.kjson.com/static/js/jsonfmt/images/Expanded.gif  "equals": "{某个固定值}"  }  },  "{鉴权上下文中的key}": {  "type": "ContextEqualCondition",  "options": {  "key": "{鉴权上下文中的另外一个key}"  }  }  }  } |

* **resources**：资源

具体格式为五段式：**"irn:{服务名}:{Region}:{所属账号}:{具体资源}"**

示例：

irn:iam:<.\*>:account0001:user 用户类操作。（不限地区）

irn:iam:<.\*>:account0001:user/user01 对资源用户user01的操作

* **actions：操作类型，格式为：**

**服务名:action**

示例：

iam:createUser iam服务的创建用户操作

* **conditions：条件，支持两种：**

1. StringEqualCondition

表示鉴权上下文中须包含该key，并且该key的值等于某个特定值。

1. ContextEqualCondition

表示鉴权上下文中须包含该key，并且该key的值等于鉴权上下文中另外一个key的值。

## 预设授权

所谓授权，就是将策略与授权对象（用户、角色、用户组、client）进行关联。

预设授权主要是针对内置用户、内置角色、内置用户组。内置client进行授权。

该类授权在系统初始化时进行，后期不允许随意修改。

### 根用户的预设授权

根用户默认拥有其所属账号下的所有资源的权限。

{

"id": "preset001",

"subjects": [

"roles:ACCOUNT\_ADMIN"

],

"resources": [

"<.\*>"

],

"actions": [

"<.\*>"

],

"effect": "allow",

"conditions": {

"account\_resource": {

"type": "ContextEqualCondition",

"options": {

"key": "account\_user"

}

}

}

}

### 子用户的预设授权

子用户默认拥有其所属账号下，并且创建者是其本人的所有资源实例的权限。

{

"id": "preset002",

"subjects": [

"roles:ACCOUNT\_USER"

],

"resources": [

"<.\*>"

],

"actions": [

"<.\*>"

],

"effect": "allow",

"conditions": {

"creator\_resource": {

"type": "ContextEqualCondition",

"options": {

"key": "user"

}

},

"account\_resource": {

"type": "ContextEqualCondition",

"options": {

"key": "account\_user"

}

}

}

}

**默认拥有List操作的功能权限，**

**List操作的数据权限，由业务方法根据线程变量中的过滤条件自行控制。**

{

"id": "preset003",

"subjects": [

"roles:ACCOUNT\_USER"

],

"resources": [

"<.\*>"

],

"actions": [

"List<.\*>"

],

"effect": "allow"

}

**子用户默认能操作自己：**

{

"id": "preset004",

"subjects": [

"roles:ACCOUNT\_USER"

],

"resources": [

"<.\*>"

],

"actions": [

"<.\*>"

],

"effect": "allow",

"conditions": {

"id": {

"type": "ContextEqualCondition",

"options": {

"key": "user"

}

}

}

}

### RDS预设授权

{

"id": "preset-rds-001",

"subjects": [

"roles:ACCOUNT\_USER"

],

"effect": "allow",

"resources": [

"irn:rds-mysql:<.\*>:<.\*>:instance"

],

"actions": [

"CreateInstanceOrder",

"ComputeInstancePrice"

],

"conditions": {}

}

# 配置示例

关于注解PermissionContext、ResourceContext、ContextKey的使用说明参考[注解使用参考](#_注解使用参考)。

## 白名单Controller层方法配置

**@PermissionContext(whitelist=true)**

@RequestMapping(value = "/v1/services", method = RequestMethod.***GET***)

**public** List<Service> listServices() {

**return** serviceService.listServices();

}

## 登录后即可访问的Controller层方法配置

**@PermissionContext(loginAccess=true)**

@RequestMapping(value = "/v1/services", method = RequestMethod.***GET***)

**public** List<Service> listServices() {

**return** serviceService.listServices();

}

## 根据策略进行鉴权的Controller层方法配置

### 3.1查询类接口（针对特定资源实例的操作）

场景：查询指定用户已授权策略

|  |
| --- |
| @ResourceContext(  service=UserService.**class**,  method="getUserById")  @PermissionContext(  service="iam",  action="GetUserPoicies",  resourceType="user",  resource="{userId}")  **public** List<Policy> getUserPoicies(@PathVariable String userId) |

上述配置，

如果要查询user01已授权的策略，

那么请求发起者需要具备resource(user01)的action(GetUserPolicies)权限

### 3.2新增类接口（非资源实例类操作）

场景：向iam中新增服务

|  |
| --- |
| @PermissionContext(  service="iam",  action="CreateService",  resourceType="service")  **public** Service createService(@RequestBody Service service, HttpServletResponse response) |

### 3.3 action或resourceId是入参内的某个属性

|  |
| --- |
| @ResourceContext(service=VmService.**class**, method="getVmById")  @PermissionContext(service="ecs",  action="{action.actionName}",  resourceType="vm  resource="{vmId}")  **public** String vmAction(@RequestBody Action action, @PathVariable String vmId) |

### 3.4列表类接口

**场景：**

云服务器实例列表，

用户user01拥有VM01、VM02的查看权限，

用户user02拥有VM02的查看权限

**预期查询结果：**

User01返回结果存在两条记录VM01、VM02

User02返回结果存在一条记录VM02

**Controller层方法添加注解如下：**

@PermissionContext(

service="ecs",

resourceType="server",

action="ListServerInstance"

)

**注意：action必须以“List”开头**

**列表类查询，数据的过滤，需要在业务代码中进行**

列表方法鉴权通过后，系统会查询该用户的所有策略，并将符合该服务、该资源类型的策略打包放入线程变量。（以本例为例，查询出当前用户拥有的所有ecs服务中所有云服务器实例的权限）。开发者可以通过访问线程变量获取策略列表，并通过策略筛选数据（**当用户拥有对某一实例的任一操作的权限，则用户拥有对该实例的查看权限**）。

筛选规则如下：

policy1 (condition11 and condition12 ) or policy2(.....)

**筛选方式分如下四种：**

**筛选方式一：**

对已有数据列表，通过代码进行筛选

参考java代码如下：

|  |
| --- |
| // 获取业务数据列表  List<CloudServer> sourceList = cloudServerService.getServerList();  // 通过ListFilterUtil工具类进行筛选—adapter中提供  // ListFilterUtil.filterListData(数据列表，业务实体类型)  List<CloudServer> dataList = ListFilterUtil.*filterListData*(sourceList, CloudServer.**class**); |

**筛选方式二：**

使用JPA查询数据时，通过过滤条件生成JPA查询条件。

参考java代码如下：

1. 在继承了JpaRepository的Dao层代码中声明带有Specification查询条件的find方法，如：

|  |
| --- |
| **public** List<CloudServer> findAll(Specification<CloudServer> specification); |

带有分页时：

|  |
| --- |
| **public** Page<CloudServer> findAll(Specification<CloudServer> specification, Pageable page); |

1. 使用上述findAll方法查询数据时，Specification类型参数声明匿名内部类，继承ListFilterSpecification类（adapter中提供）。构造方法传入过滤条件列表数据。

如：

|  |
| --- |
| Page<CloudServer> data = serverDao.findAll(  **new** ListFilterSpecification<CloudServer>() {  // 开发者自定义查询条件，若需其他查询条件，在此实现  @Override  **public** Predicate toCostomPredicate(Root<CloudServer> root, CriteriaQuery<?> query, CriteriaBuilder criteriaBuilder) {  // 自定义查询条件  **return** criteriaBuilder.equal(root.get("id").as(String.**class**), condition.getId());  }  }, PageRequest.*of*(1, 20)); |

**筛选方式三：**

使用ListFilterUtil工具类生成sql条件，再拼接自己的sql进行查询

参考java代码如下：

|  |
| --- |
| // 参数分别为，第一个为过滤条件对应的实体类，可对经过ContextKey注解标注的字段进行转义，第二个为sql中字段所属表的别名  String sqlCondition = ListFilterUtil.*filterListToSql*(CloudServer.**class,** "c");  // 该方法有两个重载方法*filterListToSql(Class clazz)*和*filterListToSql(String tableAlias)* |

引入entityManager：

|  |
| --- |
| @PersistenceContext  **private** EntityManager entityManager; |

查询数据：

|  |
| --- |
| // 自定义sql，可根据需求自行编写，加入sqlCondition条件即可  String querySql = "select \* from iam\_cloud\_server c where c.status='running' and " + sqlCondition;  Query query = entityManager.createNativeQuery(querySql, CloudServer.**class**);    List<CloudServer> data = query.getResultList(); |

**筛选方式四：**

使用ListFilterUtil工具类。根据自定义sql查询数据（可分页）

自定义列表数据查询sql：

|  |
| --- |
| String querySql=" select \* from iam\_cloud\_server c where c.status='running' "; |

引入entityManager：

|  |
| --- |
| @PersistenceContext  **private** EntityManager entityManager; |

调用ListFilterUtil工具方法：

|  |
| --- |
| Page<CloudServer> page =  ListFilterUtil.*filterPageDataBySql*(entityManager, querySql, PageRequest.*of*(1, 10), CloudServer.**class**); |

或：

|  |
| --- |
| List list = ListFilterUtil.*filterListDataBySql*(entityManager, querySql, CloudServer.**class**); |

直接查询出筛选过的数据

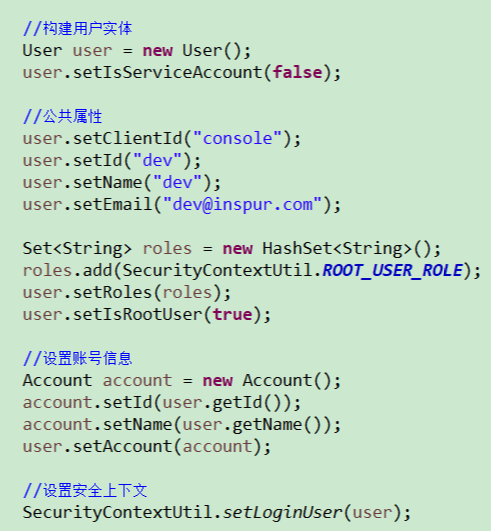
# 开发模式

配置文件 application.properties

配置项：spring.profiles.active

|  |
| --- |
| # 可选值（dev、prod）。  # dev模式下，adapter会往上下文中放置模拟用户信息，  # 并且所有接口都不受权限控制，可任意访问。  spring.profiles.active=**dev** |

dev模式下，线程变量中，会放入固定用户信息，如下：



# 注解使用参考

IAM Adapter默认拦截所有的Controller层的所有方法，**Controller层的所有方法**必须明确配置IAM Adapter提供的PermissionContext注解进行权限定义，否则，将无法被外界访问。

## 注解PermissionContext

**注解PermissionContext提供五个属性，分别如下：**

1. **whitelist**

是否为白名单，默认为false，当配置为true时表示该方法为白名单方法，不做权限检查。

1. **loginAccess**

是否登录后就可以访问，默认为false，当配置为true时表示该方法登录后就可以访问。

1. **service**

服务名，如iam、rds等，根据服务命名规范，应服务名应全部由小写字母组成。

1. **resourceType 资源类型**

举个例子：

rds 是服务名，rds服务下可能包括：mysql、oracle、db2

iam是服务名，iam服务下可能包括：user、role、usergroup

1. **resource 资源实例表达式**

**如果是针对特定资源实例的操作**，可使用 **资源实例类型/{资源实例ID}**。

如：resource=”**servers/{serverId}**”。大括弧中的变量名称来自于方法的入参。

如果大括弧中的变量来自于方法入参的内部某个属性，可以使用**点.**来表达。

如：resource=”**servers/{server.id}**

**如果不过针对特定资源实例的操作**，直接配置**resource=”资源实例类型”**即可。

如：resource=”**servers”**

1. **action操作名**

**如果要标注的Controller层方法的操作名是固定的**，可使用**action=”操作名”**进行标注，

如；**action=”CreateServer”**.

**如果要标注的Controller层方法的操作名来自于入参，可通过大括弧来指定。**

如：**action=”{actionName}”**

**如果大括弧中的变量来自于方法入参的内部某个属性**，可以使用**点.**来表达.

如：**action=”{body.actionName}”。**

## 注解ResourceContext

**注解ResourceContext提供两个属性：**

1. service：获取资源实例的接口
2. method：获取资源实例的接口方法名

**使用说明：**

此注解用于获取资源实例，例如，当PermissionContext注解的resource配置为services/{serviceId}时，Adapter会读取方法入参中的serviceId信息，并反射调用本注解配置的接口及方法。通过serviceId查询该条service实例。并将获取service实例中的accountId和creator属性的值，作为鉴权context中的account\_resource与creator\_resource属性。（当service实例中没有accountId和creator属性时，可以用到下一个注解@ContextKey进行解决）。

例：

**@ResourceContext(service=UserService.class,method="getUserById")**

以userId为入参（resource="users/{userId}"），调用UserService类的getUserById方法获取user实例。

## 注解ContextKey

注解ContextKey只有一个属性：

**value**：授权上下文中的字段名

**使用方法：**

本注解用于解决资源实例对象中包含的公共鉴权属性名称与标准名称不一致的问题，。如上例中AOP根据userId及ResourceContext注解配置的类及方法获取到了user实例。AOP会从此实例中读取accountid和creator（声明的枚举）信息。这时，user实例的属性名并不一定与accountid（或creator）一致。这时我们可以使用本注解

例如：

**@ContextKey("accountId")**

**private String account;**

**本例中，为属性account添加ContextKey注解，值为accountid。在AOP获取授权上下文accountid时，就可以读取到user实例的account属性的值。（尽管user实例中并不存在accountid属性）**

# 公共鉴权属性和服务个性化鉴权属性

策略定义时，

Conditions中定义的key来源有三处：

1. 公共鉴权属性
2. 当前登录用户
3. 该策略所关联资源类型的个性化鉴权属性。

## 公共鉴权属性

公共鉴权属性包含以下三个：

String id; //资源实例ID

String accountId; //资源所属账号Id

String creator; //资源创建者ID

在Controller层方法上，

通过**@ResourceContext**标注的，根据实例ID查询实例属性的方法，

返回的实例属性需要包含以上三个属性。

## 服务个性化鉴权属性

在服务定义时，可以为该服务定义一些个性化鉴权属性，Adapter在组织鉴权数据时，会将公共鉴权属性和当前服务的个性化鉴权属性一同放入鉴权上下文，进行权限检查。

# RestTemplate

## 默认提供两个RestTemplate实例

keycloakRestTemplate 发送请求时会携带当前登录用户的token

keycloakRestTemplateForClient 发送请求时会携带当前服务的的token（由clientId+secret换取）

通过**@Autowired**注入RestTemplate实例时，不指定bean名称，默认注入的是keycloakRestTemplate

如果想使用keycloakRestTemplateForClient，可以通过**@Qualifier("keycloakRestTemplateForClient")** 来明确指定。

## 自定义dateFormat、timeZone

如果以上两个RestTemplate实例不能满足需求，

如自定义MessageConverter的dateFormat、timeZone，你还可以通过：

KeycloakRestTemplate com.inspur.iam.adapter.config.RestTemplateFactory.createKeycloakRestTemplate(String dateFormat, String timeZone, boolean writeDatesAsTimestamps, boolean useClientToken, String referer)

创建个性化的RestTemplate实例。

# Token验证



# 从Token中解析用户

