第4章 权限管理与jwt鉴权

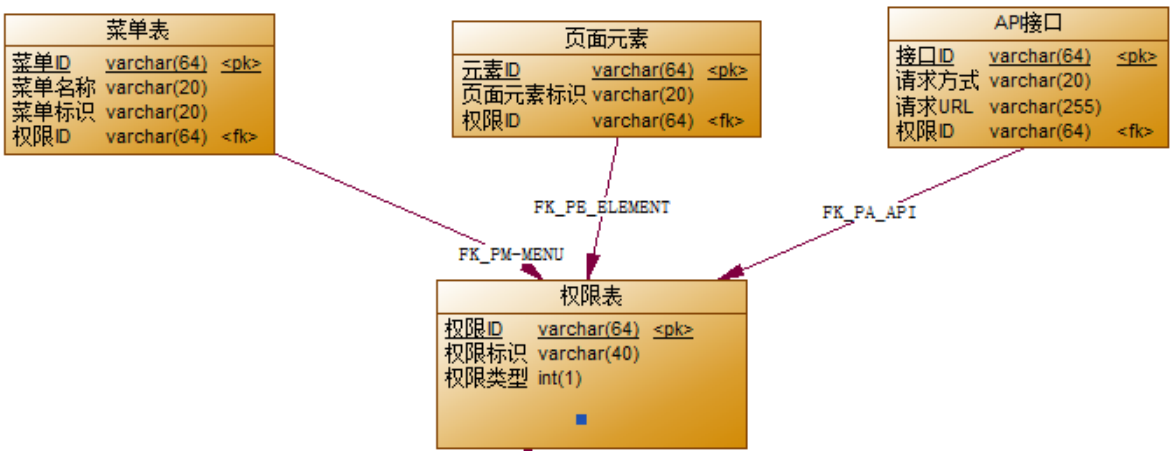
学习目标：

- 理解权限管理的需求以及设计思路
- 实现角色分配和权限分配
- 理解常见的认证机制
- 能够使用JWT完成微服务Token签发与验证

1 权限管理

1.1 需求分析

完成权限（菜单，按钮（权限点），API接口）的基本操作



权限与菜单，菜单与按钮，菜单与API接口都是一对一关系。为了方便操作，在SAAS-HRM系统的表设计中，采用基于共享主键的形式实现一对一关系维护，并且数据库约束，一切的关系维护需要程序员在代码中实现。

1.2 后端实现

1.2.1 实体类

在系统微服务中创建权限，菜单，按钮（权限点），API对象的实体类

（1）权限实体类`Permission`

```
@Entity
@Table(name = "pe_permission")
@Getter
@Setter
@NoArgsConstructor
@DynamicInsert(true)
@DynamicUpdate(true)
```

```
public class Permission implements Serializable {
    private static final long serialVersionUID = -4990810027542971546L;
    /**
     * 主键
     */
    @Id
    private String id;
    /**
     * 权限名称
     */
    private String name;
    /**
     * 权限类型 1为菜单 2为功能 3为API
     */
    private Integer type;

    /**
     * 权限编码
     */
    private String code;

    /**
     * 权限描述
     */
    private String description;

    private String pid;

    //可见状态
    private String enVisible;

    public Permission(String name, Integer type, String code, String description) {
        this.name = name;
        this.type = type;
        this.code = code;
        this.description = description;
    }


}
```

（2）权限菜单实体类`PermissionMenu`

```
@Entity
@Table(name = "pe_permission_menu")
@Getter
@Setter
public class PermissionMenu implements Serializable {
    private static final long serialVersionUID = -1002411490113957485L;

    @Id
    private String id;      //主键
    private String menuIcon; //展示图标
    private String menuOrder; //排序号
}
```

（3）权限菜单（权限点）实体类`PermissionPoint`

```
@Entity
@Table(name = "pe_permission_point")
@Getter
@Setter
public class PermissionPoint implements Serializable {
    private static final long serialVersionUID = -1002411490113957485L;
    @Id
    private String id;
    private String pointClass;
    private String pointIcon;
    private String pointStatus;
}
```

（4）权限API实体类`PermissionApi`

```
@Entity
@Table(name = "pe_permission_api")
@Getter
@Setter
public class PermissionApi implements Serializable {
    private static final long serialVersionUID = -1803315043290784820L;

    @Id
    private String id;
    private String apiUrl;
    private String apiMethod;
    private String apiLevel;//权限等级，1为通用接口权限，2为需校验接口权限
}
```

1.2.2 持久层

（1）权限持久化类

```
/**
  * 权限数据访问接口
  */
public interface PermissionDao extends JpaRepository<Permission, String>,
JpaSpecificationExecutor<Permission> {
    List<Permission> findByTypeAndPid(int type,String pid);
}
```

（2）权限菜单持久化类

```
public interface PermissionMenuDao extends JpaRepository<PermissionMenu, String>,
JpaSpecificationExecutor<PermissionMenu> {
}
```

（3）权限按钮（点）持久化类

```
public interface PermissionPointDao extends JpaRepository<PermissionPoint, String>,
JpaSpecificationExecutor<PermissionPoint> {
}
```

（4）权限API持久化类

```
public interface PermissionApiDao extends JpaRepository<PermissionApi, String>,
JpaSpecificationExecutor<PermissionApi> {
}
```

1.2.3 业务逻辑

```
package com.ihrm.system.service;

import com.ihrm.common.entity.ResultCode;
import com.ihrm.common.exception.CommonException;
import com.ihrm.common.utils.BeanMapUtils;
import com.ihrm.common.utils.IdWorker;
import com.ihrm.common.utils.PermissionConstants;
import com.ihrm.domain.system.*;
import com.ihrm.system.dao.*;
import com.ihrm.system.dao.PermissionDao;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.util.StringUtils;
import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Predicate;
import javax.persistence.criteria.Root;
import java.util.ArrayList;
```

```
import java.util.List;
import java.util.Map;

@Service
@Transactional
public class PermissionService {

    @Autowired
    private PermissionDao permissionDao;

    @Autowired
    private PermissionMenuDao permissionMenuDao;

    @Autowired
    private PermissionPointDao permissionPointDao;

    @Autowired
    private PermissionApiDao permissionApiDao;

    @Autowired
    private IdWorker idWorker;

    /**
     * 1.保存权限
     */
    public void save(Map<String,Object> map) throws Exception {
        //设置主键的值
        String id = idWorker.nextId()+"";
        //1.通过map构造permission对象
        Permission perm = BeanMapUtils.mapToBean(map,Permission.class);
        perm.setId(id);
        //2.根据类型构造不同的资源对象（菜单，按钮，api）
        int type = perm.getType();
        switch (type) {
            case PermissionConstants.PERMISSION_MENU:
                PermissionMenu menu = BeanMapUtils.mapToBean(map,PermissionMenu.class);
                menu.setId(id);
                permissionMenuDao.save(menu);
                break;
            case PermissionConstants.PERMISSION_POINT:
                PermissionPoint point =
BeanMapUtils.mapToBean(map,PermissionPoint.class);
                point.setId(id);
                permissionPointDao.save(point);
                break;
            case PermissionConstants.PERMISSION_API:
                PermissionApi api = BeanMapUtils.mapToBean(map,PermissionApi.class);
                api.setId(id);
                permissionApiDao.save(api);
                break;
            default:
                throw new CommonException(ResultCode.FAIL);
        }
```

```
        //3.保存
        permissionDao.save(perm);
    }

    /**
     * 2.更新权限
     */
    public void update(Map<String,Object> map) throws Exception {
        Permission perm = BeanMapUtils.mapToBean(map,Permission.class);
        //1.通过传递的权限id查询权限
        Permission permission = permissionDao.findById(perm.getId()).get();
        permission.setName(perm.getName());
        permission.setCode(perm.getCode());
        permission.setDescription(perm.getDescription());
        permission.setEnVisible(perm.getEnVisible());
        //2.根据类型构造不同的资源
        int type = perm.getType();
        switch (type) {
            case PermissionConstants.PERMISSION_MENU:
                PermissionMenu menu = BeanMapUtils.mapToBean(map,PermissionMenu.class);
                menu.setId(perm.getId());
                permissionMenuDao.save(menu);
                break;
            case PermissionConstants.PERMISSION_POINT:
                PermissionPoint point =
BeanMapUtils.mapToBean(map,PermissionPoint.class);
                point.setId(perm.getId());
                permissionPointDao.save(point);
                break;
            case PermissionConstants.PERMISSION_API:
                PermissionApi api = BeanMapUtils.mapToBean(map,PermissionApi.class);
                api.setId(perm.getId());
                permissionApiDao.save(api);
                break;
            default:
                throw new CommonException(ResultCode.FAIL);
        }
        //3.保存
        permissionDao.save(permission);
    }

    /**
     * 3.根据id查询
     *      //1.查询权限
     *      //2.根据权限的类型查询资源
     *      //3.构造map集合
     */
    public Map<String, Object> findById(String id) throws Exception {
        Permission perm = permissionDao.findById(id).get();
        int type = perm.getType();

        Object object = null;
```

```
        if(type == PermissionConstants.PERMISSION_MENU) {
            object = permissionMenuDao.findById(id).get();
        }else if (type == PermissionConstants.PERMISSION_POINT) {
            object = permissionPointDao.findById(id).get();
        }else if (type == PermissionConstants.PERMISSION_API) {
            object = permissionApiDao.findById(id).get();
        }else {
            throw new CommonException(ResultCode.FAIL);
        }

        Map<String, Object> map = BeanMapUtils.beanToMap(object);

        map.put("name",perm.getName());
        map.put("type",perm.getType());
        map.put("code",perm.getCode());
        map.put("description",perm.getDescription());
        map.put("pid",perm.getPid());
        map.put("enVisible",perm.getEnVisible());


        return map;
    }

    /**
     * 4.查询全部
     * type       : 查询全部权限列表type：0：菜单 + 按钮（权限点） 1：菜单2：按钮（权限点）3：API接
口
     * enVisible : 0：查询所有saas平台的最高权限，1：查询企业的权限
     * pid  ：父id
     */
    public List<Permission> findAll(Map<String, Object> map) {
        //1.需要查询条件
        Specification<Permission> spec = new Specification<Permission>() {
            /**
             * 动态拼接查询条件
             * @return
             */
            public Predicate toPredicate(Root<Permission> root, CriteriaQuery<?>
criteriaQuery, CriteriaBuilder criteriaBuilder) {
                List<Predicate> list = new ArrayList<>();
                //根据父id查询
                if(!StringUtils.isEmpty(map.get("pid"))) {
                    list.add(criteriaBuilder.equal(root.get("pid").as(String.class),
(String)map.get("pid")));
                }
                //根据enVisible查询
                if(!StringUtils.isEmpty(map.get("enVisible"))) {

 list.add(criteriaBuilder.equal(root.get("enVisible").as(String.class),
(String)map.get("enVisible")));
                }
                //根据类型 type
                if(!StringUtils.isEmpty(map.get("type"))) {
```

```
                    String ty = (String) map.get("type");
                    CriteriaBuilder.In<Object> in =
criteriaBuilder.in(root.get("type"));
                    if("0".equals(ty)) {
                        in.value(1).value(2);
                    }else{
                        in.value(Integer.parseInt(ty));
                    }
                    list.add(in);
                }
                return criteriaBuilder.and(list.toArray(new Predicate[list.size()]));
            }
        };
        return permissionDao.findAll(spec);
    }

    /**
     * 5.根据id删除
     *  //1.删除权限
     *  //2.删除权限对应的资源
     *
     */
    public void deleteById(String id) throws Exception {
        //1.通过传递的权限id查询权限
        Permission permission = permissionDao.findById(id).get();
        permissionDao.delete(permission);
        //2.根据类型构造不同的资源
        int type = permission.getType();
        switch (type) {
            case PermissionConstants.PERMISSION_MENU:
                permissionMenuDao.deleteById(id);
                break;
            case PermissionConstants.PERMISSION_POINT:
                permissionPointDao.deleteById(id);
                break;
            case PermissionConstants.PERMISSION_API:
                permissionApiDao.deleteById(id);
                break;
            default:
                throw new CommonException(ResultCode.FAIL);
        }
    }
}
```

1.2.4 控制器

```
package com.ihrm.system.controller;

import com.ihrm.common.entity.PageResult;
import com.ihrm.common.entity.Result;
import com.ihrm.common.entity.ResultCode;
```

```
import com.ihrm.domain.system.Permission;
import com.ihrm.domain.system.User;
import com.ihrm.system.service.PermissionService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.Map;

//1.解决跨域
@CrossOrigin
//2.声明restContoller
@RestController
//3.设置父路径
@RequestMapping(value="/sys")
public class PermissionController {
    @Autowired
    private PermissionService permissionService;
    /**
     * 保存
     */
    @RequestMapping(value = "/permission", method = RequestMethod.POST)
    public Result save(@RequestBody Map<String,Object> map) throws Exception {
        permissionService.save(map);
        return new Result(ResultCode.SUCCESS);
    }

    /**
     * 修改
     */
    @RequestMapping(value = "/permission/{id}", method = RequestMethod.PUT)
    public Result update(@PathVariable(value = "id") String id, @RequestBody
Map<String,Object> map) throws Exception {
        //构造id
        map.put("id",id);
        permissionService.update(map);
        return new Result(ResultCode.SUCCESS);
    }

    /**
     * 查询列表
     */
    @RequestMapping(value = "/permission", method = RequestMethod.GET)
    public Result findAll(@RequestParam Map map) {
        List<Permission> list =  permissionService.findAll(map);
        return new Result(ResultCode.SUCCESS,list);
    }

    /**
     * 根据ID查询
     */
    @RequestMapping(value = "/permission/{id}", method = RequestMethod.GET)
```

```
    public Result findById(@PathVariable(value = "id") String id) throws Exception {
        Map map = permissionService.findById(id);
        return new Result(ResultCode.SUCCESS,map);
    }



    /**
     * 根据id删除
     */
    @RequestMapping(value = "/permission/{id}", method = RequestMethod.DELETE)
    public Result delete(@PathVariable(value = "id") String id) throws Exception {
        permissionService.deleteById(id);
        return new Result(ResultCode.SUCCESS);
    }
}
```

1.3 前端实现

1.3.1 引入权限管理模块

将今日提供的资料`module-permissions`引入到工程的`/src`文件夹下，在`/src/main.js`完成模块注册

```
import permissions from '@/module-permissions/' // 权限管理
Vue.use(permissions, store)
```

1.3.2 配置API

在`/src/api/base/`目录下创建`permissions.js`

```
import {createAPI} from '@/utils/request'

const api = "/sys/permission"
export const list = data => createAPI(`${api}`, 'get', data)
export const add = data => createAPI(`${api}`, 'post', data)
export const update = data => createAPI(`${api}/${data.id}`, 'put', data)
export const remove = data => createAPI(`${api}/${data.id}`, 'delete', data)
export const detail = data => createAPI(`${api}/${data.id}`, 'get', data)
export const saveOrUpdate = data => {return data.id?update(data):add(data)}
```

1.3.3 实现权限页面

```
<template>
  <div class="dashboard-container">
    <div class="app-container">
      <el-card shadow="never">
        <el-button class="filter-item fr" size="small" style="margin-left: 10px;"
@click="handleCreate(null,1);setPid(1,'0')" type="primary" icon="el-icon-edit">添加菜单
</el-button>
```

```
            <el-table :data="dataList" fit style="width: 100%;" highlight-current-row>
                <el-table-column fixed prop="name" label="菜单名称" width="200px">
                    <template slot-scope="scope">
                        <i :class="scope.row.type==1?'ivu-icon fa fa-folder-open-o fa-
fw':'ivu-icon  el-icon-view'"
                            :style="scope.row.type==1?'margin-left: 0px':'margin-left:
20px'"></i>
                        <span @click="show(scope.$index,scope.row.id)">
{{scope.row.name}}</span>
                    </template>
                </el-table-column>
                <el-table-column fixed prop="code" label="权限标识" width="200"></el-
table-column>
                <el-table-column fixed prop="description" label="描述" width="200"></el-
table-column>
                <el-table-column fixed="right" label="操作">
                    <template slot-scope="scope">
                        <el-button v-if="scope.row.type==1"
@click="handleCreate(null,2);setPid(2,scope.row.id)" type="text" size="small">添加权限点
</el-button>
                        <el-button @click="handlerApiList(scope.row.id)" type="text"
size="small">查看api权限</el-button>
                        <el-button
@click="handleCreate(scope.row.id,scope.row.type);setPid(scope.row.type,scope.row.pid)"
type="text" size="small">查看</el-button>
                        <el-button @click="handleDelete(scope.row.id)" type="text"
size="small">删除</el-button>
                    </template>
                </el-table-column>
            </el-table>
        </el-card>
      </div>

      <el-dialog title="编辑权限" :visible.sync="dialogFormVisible"
style="hight:100px;line-height:1px">
          <el-form :model="formData" label-width="90px" style="margin-top:20px">
            <el-form-item label="权限名称">
              <el-input v-model="formData.name" autocomplete="off" style="width:90%">
</el-input>
            </el-form-item>
            <el-form-item label="权限标识">
              <el-input v-model="formData.code" autocomplete="off" style="width:90%">
</el-input>
            </el-form-item>
            <el-form-item label="权限描述">
              <el-input v-model="formData.description" autocomplete="off"
style="width:90%"></el-input>
            </el-form-item>
            <div v-if="type==1">
              <el-form-item label="菜单顺序">
                <el-input v-model="formData.menuOrder" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
```

```
              <el-form-item label="菜单icon">
                <el-input v-model="formData.menuIcon" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
            </div>
            <div v-else-if="type==2">
              <el-form-item label="按钮样式">
                <el-input v-model="formData.pointClass" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
              <el-form-item label="按钮icon">
                <el-input v-model="formData.pointIcon" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
              <el-form-item label="按钮状态">
                <el-input v-model="formData.pointStatus" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
            </div>
            <div v-else-if="type==3">
              <el-form-item label="api请求地址">
                <el-input v-model="formData.apiUrl" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
              <el-form-item label="api请求方式">
                <el-input v-model="formData.apiMethod" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
              <el-form-item label="api类型">
                <el-input v-model="formData.apiLevel" autocomplete="off"
style="width:90%"></el-input>
              </el-form-item>
            </div>

          </el-form>
          <div slot="footer" class="dialog-footer">
            <el-button @click="dialogFormVisible = false">取 消</el-button>
            <el-button type="primary" @click="saveOrUpdate">确 定</el-button>
          </div>
      </el-dialog>

      <el-dialog  title="API权限列表" :visible.sync="apiDialogVisible"
style="hight:400px;line-height:1px">
            <el-button class="filter-item fr" size="small" style="margin-left: 10px;"
@click="handleCreate(null,1);setPid(3,pid)" type="primary" icon="el-icon-edit">添加api权
限</el-button>
            <el-table :data="apiList" fit style="width: 100%;" max-height="250" >
                <el-table-column fixed prop="name" label="菜单名称" width="120px"></el-
table-column>
                <el-table-column fixed prop="code" label="权限标识" width="200"></el-
table-column>
                <el-table-column fixed prop="description" label="描述" width="200"></el-
table-column>
```

```
                <el-table-column fixed="right" label="操作" width="200">
                    <template slot-scope="scope">
                        <el-button
@click="handleCreate(scope.row.id,scope.row.type);setPid(scope.row.type,scope.row.pid)"
type="text" size="small">查看</el-button>
                        <el-button
@click="handleDelete(scope.row.id);handlerApiList(pid)" type="text" size="small">删除
</el-button>
                    </template>
                </el-table-column>
            </el-table>
      </el-dialog>
  </div>
</template>

<script>
import {saveOrUpdate,list,detail,remove} from "@/api/base/permissions"
export default {
  name: 'permissions-table-index',
  data() {
    return {
      MenuList: 'menuList',
      type:0,
      pid:"",
      dialogFormVisible:false,
      apiDialogVisible:false,
      formData:{},
      dataList:[],
      apiList:[],
      pointEnable:{}
    }
  },
  methods: {
    setPid(type,pid){
      this.pid = pid;
      this.type = type
    },
    handleCreate(id) {
      if(id && id !=undefined) {
        detail({id}).then(res => {
          this.formData = res.data.data
          this.dialogFormVisible=true
        })
      }else{
        this.formData = {}
        this.dialogFormVisible=true
      }
    },
    saveOrUpdate() {
      this.formData.type = this.type
      this.formData.pid = this.pid
      saveOrUpdate(this.formData).then(res => {
```

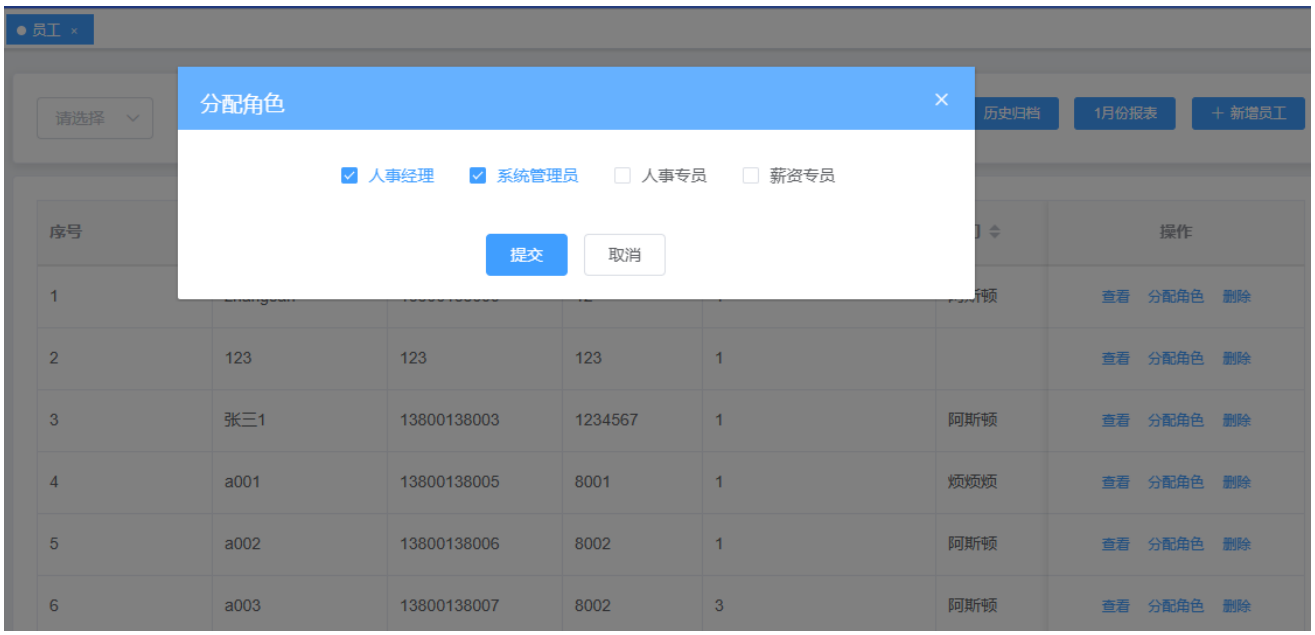
```
this.$message({message:res.data.message,type:res.data.success?"success":"error"});
        if(res.data.success){
          this.formData={};
          this.dialogFormVisible=false;
        }
        if(this.type ==3){
          this.handlerApiList(this.pid);
        }else{
          this.getList();
          this.pointEnable = {}
        }
      })
    },
    handleDelete(id) {
      remove({id}).then(res=> {

this.$message({message:res.data.message,type:res.data.success?"success":"error"});
      })
    },
    getList() {
      list({type:1,pid:0}).then(res=> {
          this.dataList = res.data.data
      })
    },
    show(index,id) {
        if(!this.pointEnable[id] == null || this.pointEnable[id]==undefined){
            list({type:2,pid:id}).then(res=> {
                for(var i = 0 ; i <res.data.data.length;i++) {
                    this.dataList.splice(index+1,0,res.data.data[i]);
                }
                this.pointEnable[id] = res.data.data.length;
                console.log(this.dataList)
            })
        }else{
            this.dataList.splice(index+1,this.pointEnable[id])
            this.pointEnable[id] = null;
        }
    },
    handlerApiList(id) {
      this.pid = id;
      list({type:3,pid:id}).then(res=> {
          this.apiList = res.data.data
          this.apiDialogVisible = true;
      })
    }
  },
  created () {
    this.getList();
  }
}
```

2 分配角色

2.1 需求分析

由于使用了RBAC模型对权限进行统一管理，所以每个SAAS-HRM平台的用户都应该具有角色的信息。进而通过角色完成对权限的识别。众所周知，一个用户可以具有很多的角色，一个角色可以被分配给不同的用户。所以用户和角色之间是多对多关系。



2.2 服务端代码实现

（1）改造用户实体类，添加角色的id集合属性，表明一个用户具有的多个角色id

在`com.ihrm.system.domain.User`用户实体类中添加与角色的多对多关系并进行JPA的配置

```
@ManyToMany
@JsonIgnore
@JoinTable(name="pe_user_role",joinColumns=
{@JoinColumn(name="user_id",referencedColumnName="id")},
   inverseJoinColumns={@JoinColumn(name="role_id",referencedColumnName="id")}
)
private Set<Role> roles = new HashSet<Role>();//用户与角色    多对多
```

在com.ihrm.system.domain.Role角色实体类中配置角色与用户的多对多关系并进行JPA配置

```
@JsonIgnore
@ManyToMany(mappedBy="roles")
private Set<User> users = new HashSet<User>(0);//角色与用户    多对多
```

（2）在`com.ihrm.system.controller.UserController`添加分配角色的控制器方法实现

```
    /**
     * 分配角色
     */
    @RequestMapping(value = "/user/assignRoles", method = RequestMethod.PUT)
    public Result assignRoles(@RequestBody Map<String,Object> map) {
        //1.获取被分配的用户id
        String userId = (String) map.get("id");
        //2.获取到角色的id列表
        List<String> roleIds = (List<String>) map.get("roleIds");
        //3.调用service完成角色分配
        userService.assignRoles(userId,roleIds);
        return new Result(ResultCode.SUCCESS);
    }
```

（3）业务逻辑层添加分配角色的业务方法

```
    /**
     * 分配角色
     */
    public void assignRoles(String userId,List<String> roleIds) {
        //1.根据id查询用户
        User user = userDao.findById(userId).get();
        //2.设置用户的角色集合
        Set<Role> roles = new HashSet<>();
        for (String roleId : roleIds) {
            Role role = roleDao.findById(roleId).get();
            roles.add(role);
        }
        //设置用户和角色集合的关系
        user.setRoles(roles);
        //3.更新用户
        userDao.save(user);
    }
```

2.3 前端代码实现

（1）`\src\module-employees` 添加分配角色的组件

```
<template>
  <div class="add-form">
    <el-dialog title="分配角色" :visible.sync="roleFormVisible" style="height:300px">
      <el-form  :model="formBase"  label-position="left" label-width="120px"
style='margin-left:120px; width:500px;'>
          <el-checkbox-group
            v-model="checkedCities1">
            <el-checkbox v-for="(item,index) in cities" :label="item.id" :key="index">
{{item.name}}</el-checkbox>
          </el-checkbox-group>
      </el-form>
      <div slot="footer" class="dialog-footer">
        <el-button type="primary" @click="createData">提交</el-button>
```

```
        <el-button @click="roleFormVisible=false">取消</el-button>
      </div>
    </el-dialog>
  </div>
</template>
<script>
import {findAll} from "@/api/base/role"
import {assignRoles} from "@/api/base/users"
export default {
    data () {
        return {
            roleFormVisible:false,
            formBase:{},
            checkedCities1:[],
            data:[],
            cities:[],
            id:null
        }
    },
    methods: {
        toAssignPrem(id) {
            findAll().then(res => {
                this.id = id;
                this.cities = res.data.data
                this.roleFormVisible=true
            })
        },
        createData() {
            assignRoles({id:this.id,ids:this.checkedCities1}).then(res => {

 this.$message({message:res.data.message,type:res.data.success?"success":"error"});
                this.roleFormVisible=false
            })
        }
    }
}
</script>
```

（2）`\src\module-employees\pages\employees-list.vue` 引入组件

```
<!--分配角色组件 -->
<component v-bind:is="addRole" ref="addRole"></component>
```

3 分配权限

3.1 需求分析

完成对角色权限的分配。

3.2 服务端代码实现

（1）角色实体类中添加与权限的多对多关系并进行JPA配置

```
@JsonIgnore //忽略json转化
@ManyToMany
@JoinTable(name="pe_role_permission",
        joinColumns={@JoinColumn(name="role_id",referencedColumnName="id")},
        inverseJoinColumns=
{@JoinColumn(name="permission_id",referencedColumnName="id")})
private Set<Permission> permissions = new HashSet<Permission>(0);//角色与模块  多对多
```

（2）控制器类（`com.ihrm.system.controller.RoleController`）添加权限分配

```
    /**
     * 分配权限
     */
    @RequestMapping(value = "/role/assignPrem", method = RequestMethod.PUT)
    public Result assignPrem(@RequestBody Map<String,Object> map) {
        //1.获取被分配的角色的id
        String roleId = (String) map.get("id");
        //2.获取到权限的id列表
        List<String> permIds = (List<String>) map.get("permIds");
        //3.调用service完成权限分配
        roleService.assignPerms(roleId,permIds);
        return new Result(ResultCode.SUCCESS);
    }
```

（3）持久化类中添加分配权限方法

```
    /**
     * 分配权限
```

```
     */
    public void assignPerms(String roleId,List<String> permIds) {
        //1.获取分配的角色对象
        Role role = roleDao.findById(roleId).get();
        //2.构造角色的权限集合
        Set<Permission> perms = new HashSet<>();
        for (String permId : permIds) {
            Permission permission = permissionDao.findById(permId).get();
            //需要根据父id和类型查询API权限列表
            List<Permission> apiList =
permissionDao.findByTypeAndPid(PermissionConstants.PERMISSION_API, permission.getId());
            perms.addAll(apiList);//自定赋予API权限
            perms.add(permission);//当前菜单或按钮的权限
        }
        System.out.println(perms.size());
        //3.设置角色和权限的关系
        role.setPermissions(perms);
        //4.更新角色
        roleDao.save(role);
    }
```

3.3 前端代码实现

（1）在 `\src\module-settings\components\role-list.vue` 绑定权限按钮

```
<el-table-column fixed="right" label="操作" align="center" width="250">
   <template slot-scope="scope">
      <el-button @click="handlerPerm(scope.row)" type="text" size="small">分配权限</el-
button>
      <el-button @click="handleUpdate(scope.row)" type="text" size="small">修改</el-
button>
      <el-button @click="handleDelete(scope.row)" type="text" size="small">删除</el-
button>
   </template>
</el-table-column>
```

（2）在 `\\src\api\base\role.js` 中添加分配权限的API方法

```
export const assignPrem = data => createAPI(`/sys/role/assignPrem`, 'put', data)
```

（3）`\src\module-settings\components\role-list.vue` 使用Element-UI构造权限树

```
    <el-dialog :title="'为【'+formData.name+'】分配权限'" :visible.sync="permFormVisible"
style="hight:100px;line-height:1px">
      <el-tree
        :data="treeData"
        default-expand-all
        show-checkbox
        node-key="id"
        ref="tree"
        :default-checked-keys="checkNodes"
```

```
          :props="{label:'name'}">
        </el-tree>
        <div slot="footer" class="dialog-footer">
          <el-button @click="permFormVisible = false">取 消</el-button>
          <el-button type="primary" @click="assignPrem">确 定</el-button>
        </div>
      </el-dialog>
```

（4）完成添加权限

```
import {list,add,update,remove,detail,assignPrem} from "@/api/base/role"
import * as permApi from "@/api/base/permissions"
import commonApi from "@/utils/common"
import PageTool from './../../components/page/page-tool'
var _this = null
export default {
  name: 'roleList',
  components: {PageTool},
  props: ['objId'],
  data() {
    return {
      formData:{},
      treeData:[],
      checkNodes:[],
      dialogFormVisible: false,
      permFormVisible:false,
      dataList:[],
      counts:0,
      requestParameters:{
        page: 1,
        pagesize: 10
      }
    }
  },
  methods: {
    assignPrem() {

 assignPrem({roleId:this.formData.id,ids:this.$refs.tree.getCheckedKeys()}).then(res =>
{

this.$message({message:res.data.message,type:res.data.success?"success":"error"});
          this.permFormVisible=false
      })
    },
    handlerPerm(obj) {
       detail({id:obj.id}).then(res=>{
         this.formData = res.data.data;
         if(this.formData.menusIds != null) {
            this.checkNodes = this.formData.menusIds.split(",")
         }
         if(this.formData.pointIds != null) {
          this.checkNodes.push(this.formData.pointIds.split(","))
         }
```

```
          permApi.list({type:0,pid:null}).then(res => {
            this.treeData = commonApi.transformTozTreeFormat(res.data.data)
            this.permFormVisible=true
          })
       })
    },
    handlerAdd() {
      this.formData={}
      this.dialogFormVisible = true
    },
    handleDelete(obj) {
      this.$confirm(
        `本次操作将删除${obj.name},删除后角色将不可恢复，您确认删除吗？`
      ).then(() => {
          remove({id: obj.id}).then(res => {

this.$message({message:res.data.message,type:res.data.success?"success":"error"});
              this.doQuery()
          })
      })
    },
    handleUpdate(obj) {
      detail({id:obj.id}).then(res=>{
        this.formData = res.data.data;
        this.formData.id = obj.id;
        this.dialogFormVisible = true
      })
    },
    saveOrUpdate() {
      if(this.formData.id == null || this.formData.id == undefined) {
          this.save()
      }else{
          this.update();
      }
    },
    update(){
      update(this.formData).then(res=>{

this.$message({message:res.data.message,type:res.data.success?"success":"error"});
        if(res.data.success){
          this.formData={};
          this.dialogFormVisible=false;
          this.doQuery();
        }
      })
    },
    save() {
      add(this.formData).then(res=>{

this.$message({message:res.data.message,type:res.data.success?"success":"error"});
        if(res.data.success){
          this.formData={};
          this.dialogFormVisible=false;
```

```
            this.doQuery();
          }
        })
      },
      // 获取详情
      doQuery() {
        list(this.requestParameters).then(res => {
            this.dataList = res.data.data.rows
            this.counts = res.data.data.total
          })
      },
      // 每页显示信息条数
      handleSizeChange(pageSize) {
        this.requestParameters.pagesize = pageSize
        if (this.requestParameters.page === 1) {
          _this.doQuery(this.requestParameters)
        }
      },
      // 进入某一页
      handleCurrentChange(val) {
        this.requestParameters.page = val
        _this.doQuery()
      },
    },
    // 挂载结束
    mounted: function() {},
    // 创建完毕状态
    created: function() {
      _this = this
      this.doQuery()
    },
    // 组件更新
    updated: function() {}
}
</script>
```

4 常见的认证机制

4.1 HTTP Basic Auth

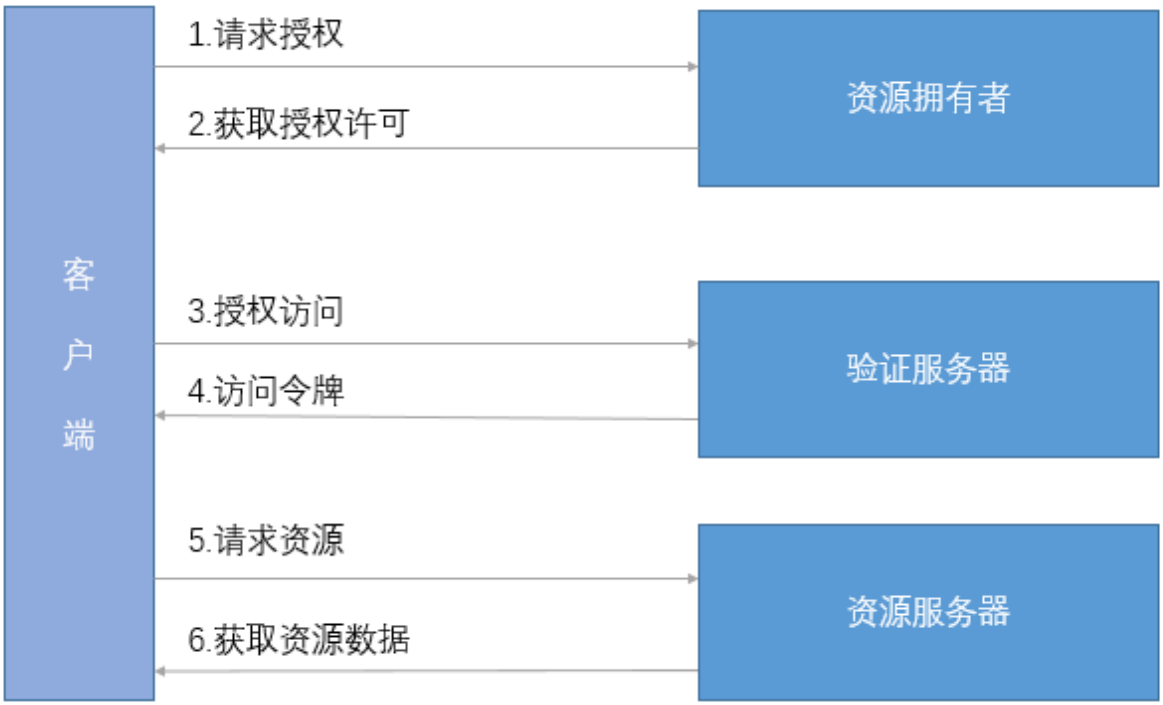
HTTP Basic Auth简单点说明就是每次请求API时都提供用户的username和password，简言之，Basic Auth是配合RESTful API 使用的最简单的认证方式，只需提供用户名密码即可，但由于有把用户名密码暴露给第三方客户端的风险，在生产环境下被使用的越来越少。因此，在开发对外开放的RESTful API时，尽量避免采用HTTP Basic Auth

4.2 Cookie Auth

Cookie认证机制就是为一次请求认证在服务端创建一个Session对象，同时在客户端的浏览器端创建了一个Cookie对象；通过客户端带上来Cookie对象来与服务器端的session对象匹配来实现状态管理的。默认的，当我们关闭浏览器的时候，cookie会被删除。但可以通过修改cookie 的expire time使cookie在一定时间内有效

4.3 OAuth

OAuth（开放授权）是一个开放的授权标准，允许用户让第三方应用访问该用户在某一web服务上存储的私密的资源（如照片，视频，联系人列表），而无需将用户名和密码提供给第三方应用。 OAuth允许用户提供一个令牌，而不是用户名和密码来访问他们存放在特定服务提供者的数据。每一个令牌授权一个特定的第三方系统（例如，视频编辑网站)在特定的时段（例如，接下来的2小时内）内访问特定的资源（例如仅仅是某一相册中的视频）。这样，OAuth让用户可以授权第三方网站访问他们存储在另外服务提供者的某些特定信息，而非所有内容

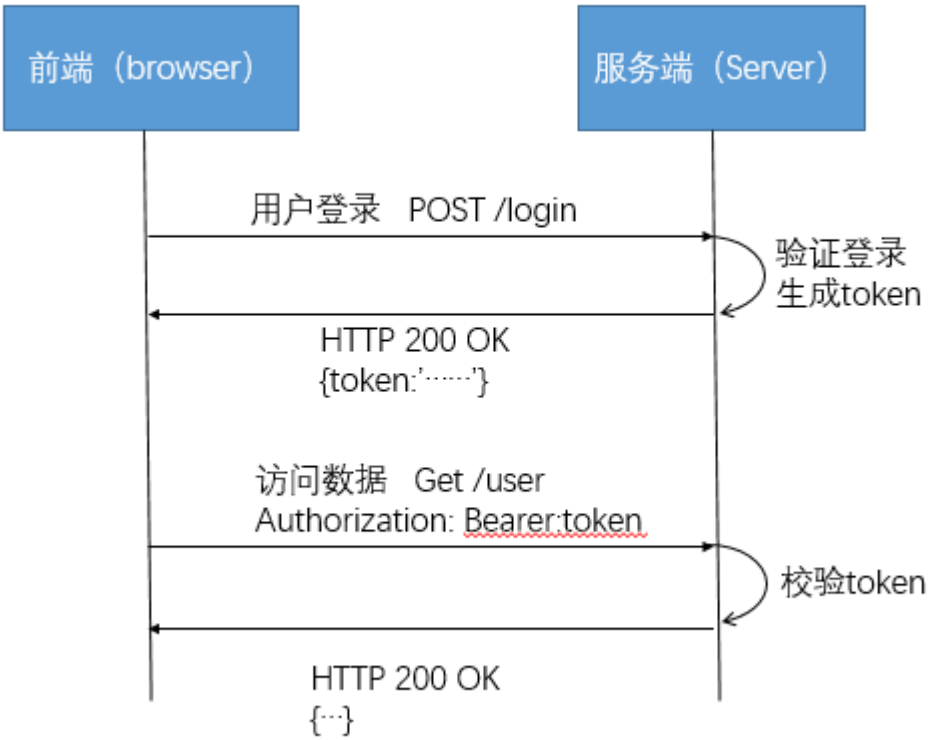


这种基于OAuth的认证机制适用于个人消费者类的互联网产品，如社交类APP等应用，但是不太适合拥有自有认证权限管理的企业应用。

4.4 Token Auth

使用基于 Token 的身份验证方法，在服务端不需要存储用户的登录记录。大概的流程是这样的：

1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端
4. 客户端收到 Token 以后可以把它存储起来，比如放在 Cookie 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的 Token
6. 服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据



Token Auth的优点

- 支持跨域访问: Cookie是不允许垮域访问的，这一点对Token机制是不存在的，前提是传输的用户认证信息通过HTTP头传输.
- 无状态(也称：服务端可扩展行):Token机制在服务端不需要存储session信息，因为Token 自身包含了所有登录用户的信息，只需要在客户端的cookie或本地介质存储状态信息.
- 更适用CDN: 可以通过内容分发网络请求你服务端的所有资料（如：javascript，HTML,图片等），而你的服务端只要提供API即可.
- 去耦: 不需要绑定到一个特定的身份验证方案。Token可以在任何地方生成，只要在你的API被调用的时候，你可以进行Token生成调用即可.
- 更适用于移动应用: 当你的客户端是一个原生平台（iOS, Android，Windows 8等）时，Cookie是不被支持的（你需要通过Cookie容器进行处理），这时采用Token认证机制就会简单得多。
- CSRF:因为不再依赖于Cookie，所以你就不需要考虑对CSRF（跨站请求伪造）的防范。
- 性能: 一次网络往返时间（通过数据库查询session信息）总比做一次HMACSHA256计算 的Token验证和解析要费时得多.
- 不需要为登录页面做特殊处理: 如果你使用Protractor 做功能测试的时候，不再需要为登录页面做特殊处理.
- 基于标准化:你的API可以采用标准化的 JSON Web Token (JWT). 这个标准已经存在多个后端库（.NET, Ruby, Java,Python, PHP）和多家公司的支持（如：Firebase,Google, Microsoft）.

5 HRM中的TOKEN签发与验证

5.1 什么是JWT

JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息。在Java世界中通过JJWT实现JWT创建和验证。

5.2 JJWT的快速入门

5.2.1 token的创建

（1）创建maven工程，引入依赖

```
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.6.0</version>
</dependency>
```

（2）创建类CreateJwtTest，用于生成token

```
public class CreateJwtTest {
    public static void main(String[] args) {
        JwtBuilder builder= Jwts.builder().setId("888")
                .setSubject("小白")
                .setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256,"itcast");
        System.out.println( builder.compact() );
    }
}
```

（3）测试运行，输出如下:

```
eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmbOiLCJpYXQiOjE1MjM0MTM0NTh9.gqOJ-cOM_qCNqU_s-d_IrRytaNenesPmqAIhQpYXHZk
```

5.2.2 token的解析

我们刚才已经创建了token ，在web应用中这个操作是由服务端进行然后发给客户端，客户端在下次向服务端发送请求时需要携带这个token（这就好像是拿着一张门票一样），那服务端接到这个token 应该解析出token中的信息（例如用户id）,根据这些信息查询数据库返回相应的结果。

创建ParseJwtTest

```
public class ParseJwtTest {
    public static void main(String[] args) {
        String
token="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmbOiLCJpYXQiO
jE1MjM0MTM0NTh9.gqOJ-cOM_qCNqU_s-d_IrRytaNenesPmqAIhQpYXHZk";
        Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(token).getBody();
        System.out.println("id:"+claims.getId());
        System.out.println("subject:"+claims.getSubject());
        System.out.println("IssuedAt:"+claims.getIssuedAt());
    }
}
```

试着将token或签名秘钥篡改一下，会发现运行时就会报错，所以解析token也就是验证token

5.2.3 自定义claims

我们刚才的例子只是存储了id和subject两个信息，如果你想存储更多的信息（例如角色）可以定义自定义claims

（1）创建CreateJwtTest3，并存储指定的内容

```
public class CreateJwtTest3 {
    public static void main(String[] args) {
        //为了方便测试，我们将过期时间设置为1分钟
        long now = System.currentTimeMillis();//当前时间
        long exp = now + 1000*60;//过期时间为1分钟
        JwtBuilder builder= Jwts.builder().setId("888")
                .setSubject("小白")
                .setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256,"itcast")
                .setExpiration(new Date(exp))
                .claim("roles","admin") //自定义claims存储数据
                .claim("logo","logo.png");
        System.out.println( builder.compact() );
    }
}
```

（2）修改ParseJwtTest，获取指定信息

```
public class ParseJwtTest {
    public static void main(String[] args) {
        String
compactJws="eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiI4ODgiLCJzdWIiOiLlsI_nmbOiLCJpYXQiOjE1MjM0MT
czMjMsImV4cCI6MTUyMzQxNzM4Mywicm9sZXMiOiJhZG1pbiIsImxvZ28iOiJsb2dvLnBuZyJ9.b11p4g4rE94r
qFhcfzdJTPCORikqP_1zJ1MP8KihYTQ";
        Claims claims =
Jwts.parser().setSigningKey("itcast").parseClaimsJws(compactJws).getBody();
        System.out.println("id:"+claims.getId());
        System.out.println("subject:"+claims.getSubject());
        System.out.println("roles:"+claims.get("roles"));
        System.out.println("logo:"+claims.get("logo"));
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");
```

```
        System.out.println("签发时间:"+sdf.format(claims.getIssuedAt()));
        System.out.println("过期时间:"+sdf.format(claims.getExpiration()));
        System.out.println("当前时间:"+sdf.format(new Date()) );
    }
}
```

5.3 JWT工具类

在ihrm_common工程中创建JwtUtil工具类

```
@ConfigurationProperties("jwt.config")
public class JwtUtil {

    private String key;
    private long ttl;

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public long getTtl() {
        return ttl;
    }

    public void setTtl(long ttl) {
        this.ttl = ttl;
    }

    /**
     * 签发 token
     */
    public String createJWT(String id, String subject,Map<String,Object> map){
        long now=System.currentTimeMillis();
        long exp=now+ttl;
        JwtBuilder jwtBuilder = Jwts.builder().setId(id)
                .setSubject(subject).setIssuedAt(new Date())
                .signWith(SignatureAlgorithm.HS256, key);
        for(Map.Entry<String,Object> entry:map.entrySet()) {
            jwtBuilder.claim(entry.getKey(),entry.getValue());
        }
        if(ttl>0){
            jwtBuilder.setExpiration( new Date(exp));
        }
        String token = jwtBuilder.compact();
        return token;
    }

    /**
```

```
     * 解析JWT
     * @param token
     * @return
     */
    public Claims parseJWT(String token){
        Claims claims = null;
        try {
            claims = Jwts.parser()
                    .setSigningKey(key)
                    .parseClaimsJws(token).getBody();
        }catch (Exception e){

        }
        return claims;
    }
}
```

（3）修改ihrm_common工程的application.yml, 添加配置

```
jwt:
 config:
    key: saas-ihrm
    ttl: 360000
```

5.4 登录成功签发token

（1）配置JwtUtil。修改ihrm_system工程的启动类

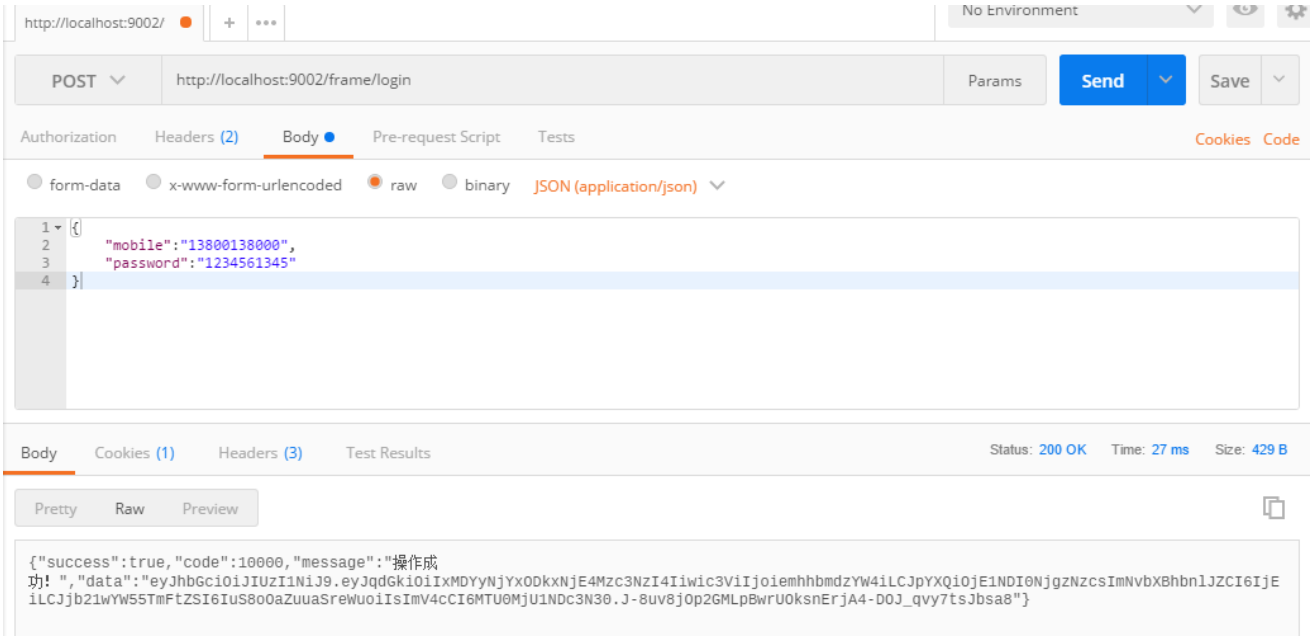
```
@Bean
public JwtUtil jwtUtil(){
    return new util.JwtUtil();
}
```

（2）添加登录方法

```
    /**
     * 用户登录
     *  1.通过service根据mobile查询用户
     *  2.比较password
     *  3.生成jwt信息
     *
     */
    @RequestMapping(value="/login",method = RequestMethod.POST)
    public Result login(@RequestBody Map<String,String> loginMap) {
        String mobile = loginMap.get("mobile");
        String password = loginMap.get("password");
        User user = userService.findByMobile(mobile);
        //登录失败
        if(user == null || !user.getPassword().equals(password)) {
            return new Result(ResultCode.MOBILEORPASSWORDERROR);
        }else {
```

```
        //登录成功
            Map<String,Object> map = new HashMap<>();
            map.put("companyId",user.getCompanyId());
            map.put("companyName",user.getCompanyName());
            String token = jwtUtils.createJwt(user.getId(), user.getUsername(), map);
            return new Result(ResultCode.SUCCESS,token);
        }
    }
```

（3）测试运行结果



使用postman验证登录返回：

```
{"success":true,"code":10000,"message":"操作成
功！","data":"eyJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIxMDYyNjYxODkxNjE4Mzc3NzI4Iiwic3ViIjoiemhhb
mdzYW4iLCJpYXQiOjE1NDI0NjgzNzcsImNvbXBhbnlJZCI6IjEiLCJjb21wYW55TmFtZSI6IuS8oOaZuuaSreWu
oiIsImV4cCI6MTU0MjU1NDc3N30.J-8uv8jOp2GMLpBwrUOksnErjA4-DOJ_qvy7tsJbsa8"}
```

5.5 获取用户信息鉴权

需求：用户登录成功之后，会发送一个新的请求到服务端，获取用户的详细信息。获取用户信息的过程中必须登录才能，否则不能获取。

前后端约定：前端请求微服务时需要添加头信息Authorization ,内容为Bearer+空格+token

（1）添加响应值对象

```
@Getter
@Setter
@NoArgsConstructor
public class ProfileResult {

    private String mobile;
    private String username;
```

```
    private String company;
    private Map roles;

    public ProfileResult(User user) {
        this.mobile = user.getMobile();
        this.username = user.getUsername();
        this.company = user.getCompanyName();
        //角色数据
        Set<String> menus = new HashSet<>();
        Set<String> points = new HashSet<>();
        Set<String> apis = new HashSet<>();
        Map rolesMap = new HashMap<>();
        for (Role role : user.getRoles()) {
            for (Permission perm : role.getPermissions()) {
                String code = perm.getCode();
                if(perm.getType() == 1) {
                    menus.add(code);
                }else if(perm.getType() == 2) {
                    points.add(code);
                }else {
                    apis.add(code);
                }
            }
        }
        rolesMap.put("menus",menus);
        rolesMap.put("points",points);
        rolesMap.put("apis",points);
        this.roles = rolesMap;
    }
}
```

（2）添加profile方法

```
/**
 * 获取个人信息
 */
@RequestMapping(value = "/profile", method = RequestMethod.POST)
public Result profile(HttpServletRequest request) throws Exception {
    //临时使用
    String userId = "1";
    User user = userService.findById(userId);
    return new Result(ResultCode.SUCCESS,new ProfileResult(user));
}
```

（3）验证token

思路：从请求中获取key为Authorization的token信息，并使用jwt验证，验证成功后获取隐藏信息。

修改profile方法添加如下代码

```
@RequestMapping(value = "/profile", method = RequestMethod.POST)
public Result profile(HttpServletRequest request) throws Exception {
    //请求中获取key为Authorization的头信息
```

```
    String authorization = request.getHeader("Authorization");
    if(StringUtils.isEmpty(authorization)) {
        throw new CommonException(ResultCode.UNAUTHENTICATED);
    }
    //前后端约定头信息内容以  Bearer+空格+token 形式组成
    String token = authorization.replace("Bearer ", "");
    //比较并获取claims
    Claims claims = jwtUtil.parseJWT(token);
    if(claims == null) {
        throw new CommonException(ResultCode.UNAUTHENTICATED);
    }
    String userId = claims.getId();
    User user = userService.findById(userId);
    return new Result(ResultCode.SUCCESS,new ProfileResult(user));
}
```