第1章 SAAS-HRM系统概述与搭建环境

学习目标：

- 理解SaaS的基本概念
- 了解SAAS-HRM的基本需求和开发方式
- 掌握Power Designer的用例图
- 完成SAAS-HRM父模块及公共模块的环境搭建
- 完成企业微服务中企业CRUD功能

1 初识SaaS

1.1 云服务的三种模式

1.1.1 IaaS（基础设施即服务）

IaaS（Infrastructure as a Service），即基础设施即服务。提供给消费者的服务是对所有计算基础设施的利用，包括处理CPU、内存、存储、网络和其它基本的计算资源，用户能够部署和运行任意软件，包括操作系统和应用程序。消费者不管理或控制任何云计算基础设施，但能控制操作系统的选择、存储空间、部署的应用，也有可能获得有限制的网络组件（例如路由器、防火墙、负载均衡器等）的控制

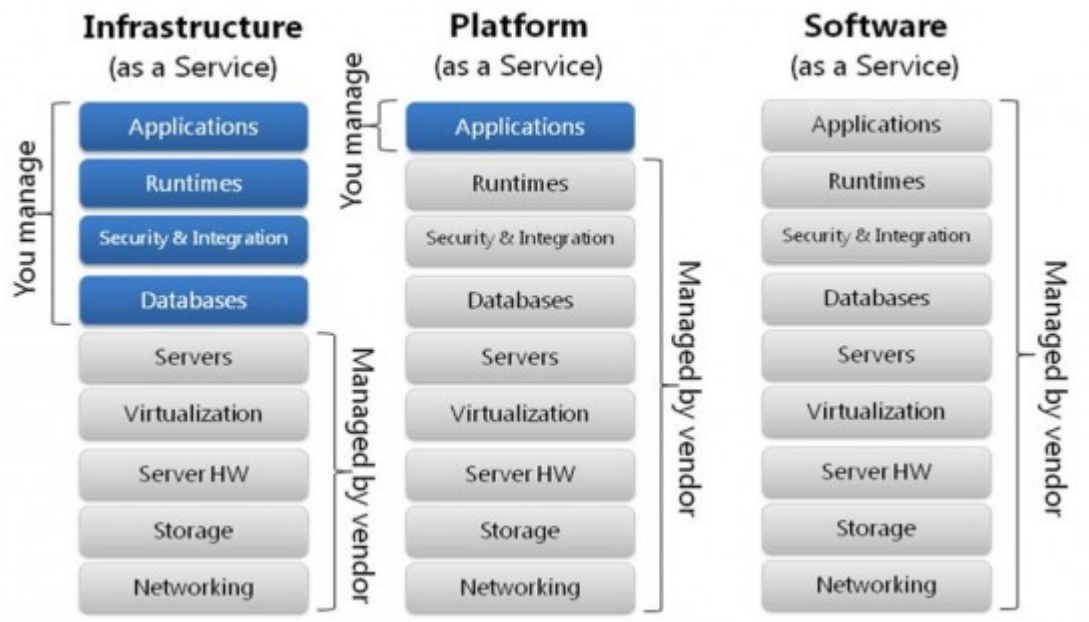
1.1.2 PaaS（平台即服务）

PaaS（Platform-as-a-Service），即平台即服务。提供给消费者的服务是把客户采用提供的开发语言和工具（例如Java，python, .Net等）开发的或收购的应用程序部署到供应商的云计算基础设施上去。客户不需要管理或控制底层的云基础设施，包括网络、服务器、操作系统、存储等，但客户能控制部署的应用程序，也可能控制运行应用程序的托管环境配置

1.1.3 SaaS（软件即服务）

SaaS（Software-as-a-Service），即软件即服务。提供给消费者完整的软件解决方案，你可以从软件服务商处以租用或购买等方式获取软件应用，组织用户即可通过 Internet 连接到该应用（通常使用 Web 浏览器）。所有基础结构、中间件、应用软件和应用数据都位于服务提供商的数据中心内。服务提供商负责管理硬件和软件，并根据适当的服务协议确保应用和数据的可用性和安全性。SaaS 让组织能够通过最低前期成本的应用快速建成投产。

1.1.4 区别与联系



1.2 SaaS的概述

1.2.1 Saas详解

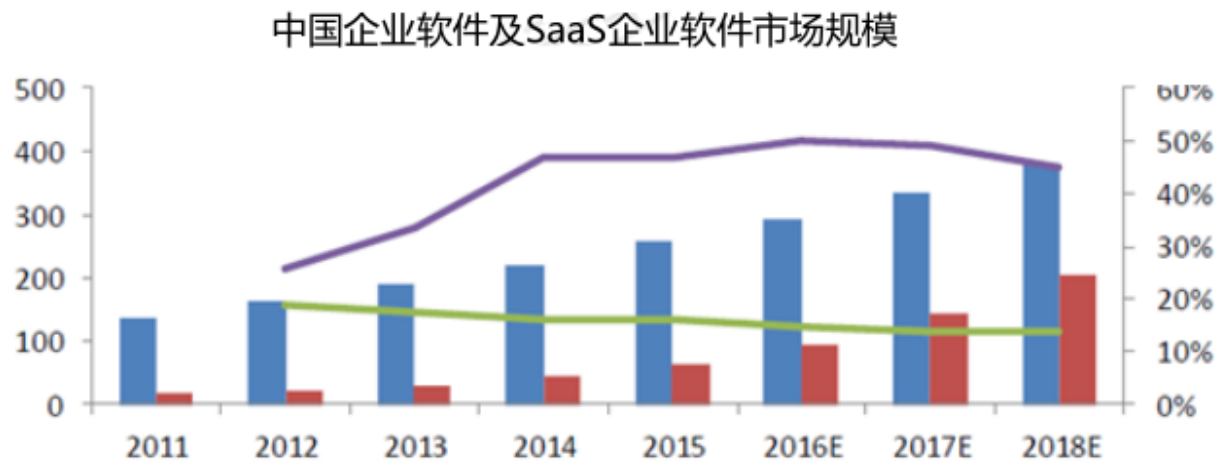
SaaS（Software-as-a-service）的意思是软件即服务。简单说就是在线系统模式，即软件服务商提供的软件在线服务。

1.2.2 应用领域与行业前景

SaaS软件就适用对象而言，可以划分为针对个人的与针对企业的

面向个人的SaaS产品：在线文档，账务管理，文件管理，日程计划、照片管理、联系人管理，等等云类型的服务

而面向企业的SaaS产品主要包括：CRM（客户关系管理）、ERP（企业资源计划管理）、线上视频或者与群组通话会议、HRM（人力资源管理）、OA(办公系统)、外勤管理、财务管理、审批管理等。



1.2.3 Saas与传统软件对比

降低企业成本：按需购买，即租即用，无需关注软件的开发维护。

软件更新迭代快速：和传统软件相比，由于saas部署在云端，使得软件的更新迭代速度加快

支持远程办公：将数据存储到云后，用户即可通过任何连接到 Internet 的计算机或移动设备访问其信息，

2 SaaS-HRM 需求分析

2.1 什么是SaaS-HRM

SaaS-HRM是基于saas模式的人力资源管理系统。他不同于传统的人力资源软件应用，使用者只需打开浏览器即可管理上百人的薪酬、绩效、社保、入职离职。

2.2 原型分析法

原型分析的理念是指在获取一组基本需求之后，快速地构造出一个能够反映用户需求的初始系统原型。让用户看到未来系统的概貌，以 便判断哪些功能是符合要求的，哪些方面还需要改进，然后不断地对这些需求进一步补充、细化和修改。依次类推，反复进行，直到用户满意为止并由此开发出完整 的系统。

> 简单的说，原型分析法就是在最短的时间内，以最直观的方式获取用户最真实的需求

2.3 UML的用例图

2.3.1 UML统一建模语言

Unified Modeling Language (UML)又称统一建模语言或标准建模语言，是始于1997年一个OMG标准，它是一个支持模型化和软件系统开发的图形化语言，为软件开发的所有阶段提供模型化和可视化支持，包括由需求分析到规格，到构造和配置。 面向对象的分析与设计(OOA&D，OOAD)方法的发展在80年代末至90年代中出现了一个高潮，UML是这个高潮的产物。它不仅统一了Booch、Rumbaugh和Jacobson的表示方法，而且对其作了进一步的发展，并最终统一为大众所接受的标准建模语言。UML中包含很多图形（用例图，类图，状态图等等），**其中用例图是最能体现系统结构的图形**

2.3.2 用例图

用例图（use case）主要用来描述用户与用例之间的关联关系。说明的是谁要使用系统，以及他们使用该系统可以做些什么。一个用例图包含了多个模型元素，如系统、参与者和用例，并且显示这些元素之间的各种关系，如泛化、关联和依赖。它展示了一个外部用户能够观察到的系统功能模型图。

2.3.3 需求分析软件

Power Designer 是Sybase公司的CASE工具集，使用它可以方便地对管理信息系统进行分析设计，他几乎包括了数据库模型设计的全过程。利用Power Designer可以制作数据流程图、概念数据模型、物理数据模型，还可以为数据仓库制作结构模型，也能对团队设计模型进行控制。

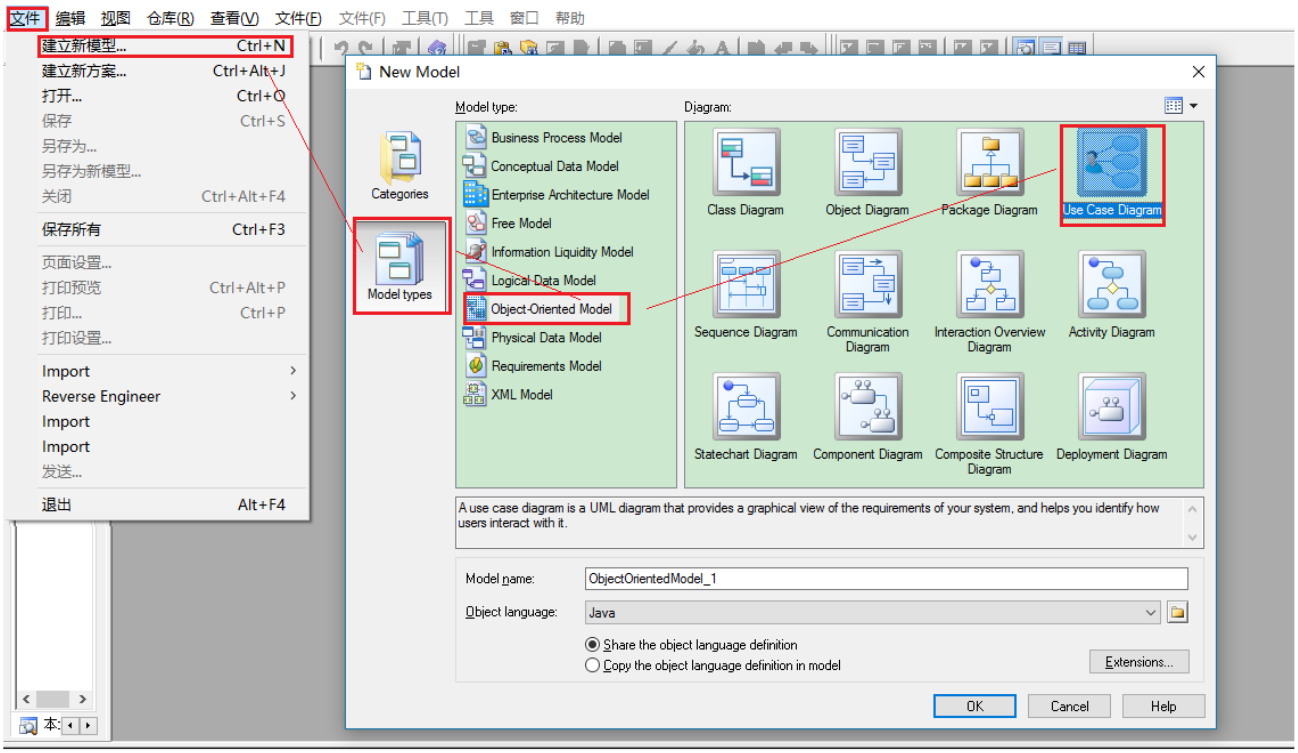
（1）下载安装

使用第一天资料中准备好的安装包安装Power Designer，安装过程略

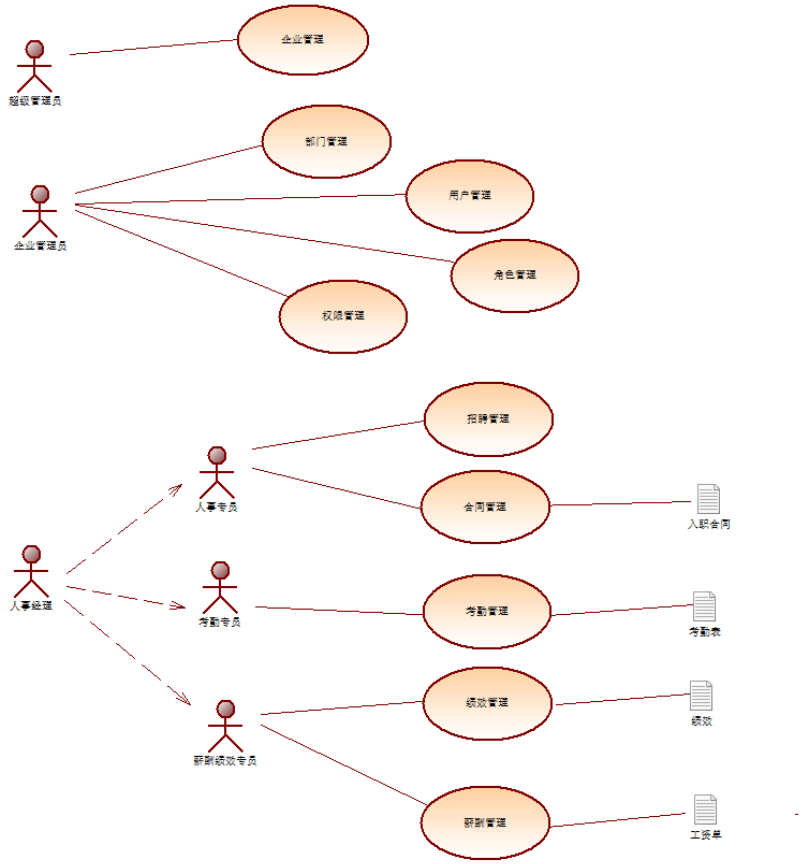
（2）使用Power Designer绘制用例图

绘制步骤：

文件=>建立新模型=>选择Modeltypes=>Use Case



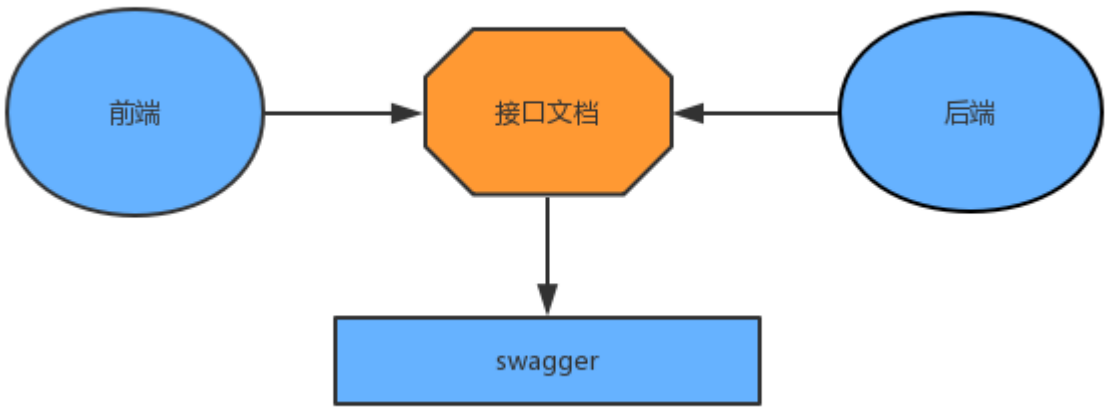
基本用例图：



3 系统设计

3.1 开发方式

SaaS-IHRM系统采用前后端分离的开发方式。



后端给前端提供数据,前端负责HTML渲染(可以在服务器渲染,也可以在浏览器渲染)和用户交互。双方通过文档的形式规范接口内容

3.2 技术架构

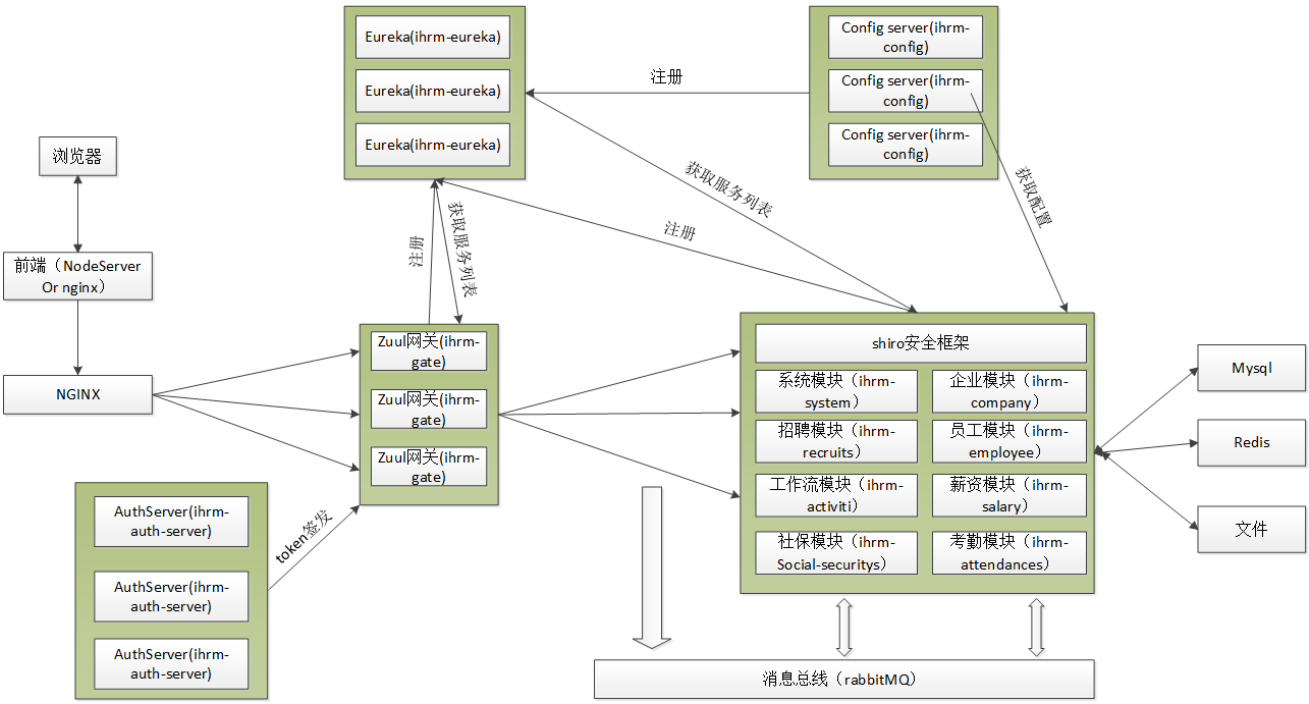
（1）前端技术栈

以Node.js为核心的Vue.js前端技术生态架构

（2）后端技术栈

SpringBoot+SpringCloud+SpringMVC+SpringData（Spring全家桶）

3.3 系统结构



3.4 API文档

课程提供了前后端开发接口文档（采用Swagger语言进行编写），并与Ngin进行了整合。双击Nginx执行文件启动后，在地址栏输入http://localhost:801 即可访问API文档

4 工程搭建

4.1 前置知识点的说明

Saas-HRM系统后端采用

SpringBoot+SpringCloud+SpringMVC+SpringData

Saas-HRM系统前端采用

基于nodejs的vue框架完成编写使用element-ui组件库快速开发前端界面

学员应对以上前后端技术有初步的了解

4.2 开发环境要求

JDK1.8

数据库mysql 5.7

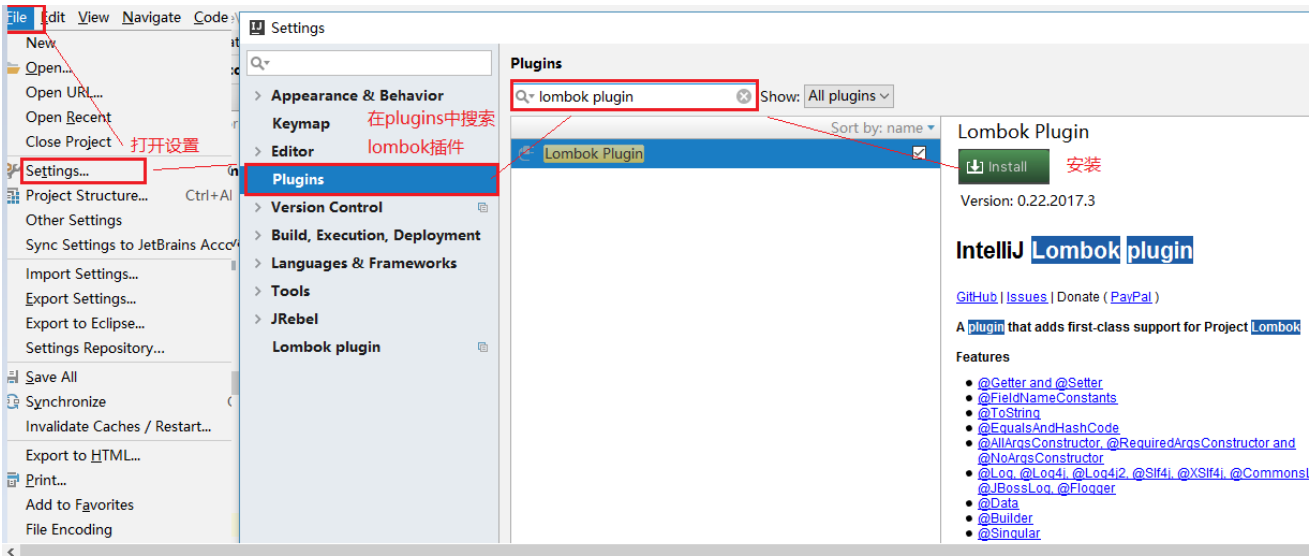
开发工具 idea 2017.1.2

maven版本3.3.9

4.2.1 lombok 插件

lombok是一款可以精减java代码、提升开发人员生产效率的辅助工具，利用注解在编译期自动生成setter/getter/toString()/constructor之类的代码

（1）idea中安装插件



（2）在pom文件中添加插件的依赖

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.16.16</version>
</dependency>
```

（3）常见注解

- @Data 注解在类上；提供类所有属性的 getting 和 setting 方法，此外还提供了equals、canEqual、hashCode、toString 方法
- @Setter ：注解在属性上；为属性提供 setting 方法
- @Setter ：注解在属性上；为属性提供 getting 方法
- @NoArgsConstructor ：注解在类上；为类提供一个无参的构造方法
- @AllArgsConstructor ：注解在类上；为类提供一个全参的构造方法

4.3 构建父工程

在IDEA中创建父工程ihrm_parent并导入相应的坐标如下：

```
    <packaging>pom</packaging>
```

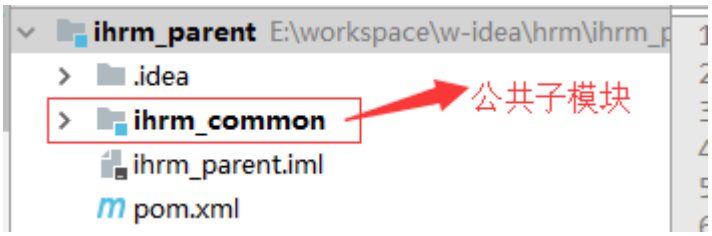
```
    <name>ihrm_parent</name>
    <description>IHRM-黑马程序员</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.5.RELEASE</version>
        <relativePath/>
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
        <fastjson.version>1.2.47</fastjson.version>
    </properties>


    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Finchley.SR1</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>fastjson</artifactId>
            <version>${fastjson.version}</version>
        </dependency>
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>
    </dependencies>

    <repositories>
```

```
        <repository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </repository>
    </repositories>

    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
            <snapshots>
                <enabled>false</enabled>
            </snapshots>
        </pluginRepository>
    </pluginRepositories>
    <build>
        <plugins>
            <!--编译插件-->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.1</version>
                <configuration>
                    <source>${java.version}</source>
                    <target>${java.version}</target>
                </configuration>
            </plugin>

            <!--单元测试插件-->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-plugin</artifactId>
```

```
                <version>2.12.4</version>
                <configuration>
                    <skipTests>true</skipTests>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

4.4 构建公共子模块

4.4.1 构建公共子模块ihrm-common



4.4.2 创建返回结果实体类

（1）新建com.ihrm.common.entity包，包下创建类Result，用于控制器类返回结果

```
package com.ihrm.common.entity;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.databind.annotation.JsonSerialize;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;


@Data
@NoArgsConstructor
//非空数据不显示
@JsonInclude(JsonInclude.Include.NON_NULL)
public class Result {

    private boolean success;//是否成功
    private Integer code;// 返回码
    private String message;//返回信息
    private Object data;// 返回数据

    public Result(ResultCode code) {
        this.success = code.success;
        this.code = code.code;
        this.message = code.message;
    }

    public Result(ResultCode code,Object data) {
        this.success = code.success;
        this.code = code.code;
```

```
        this.message = code.message;
        this.data = data;
    }

    public Result(Integer code,String message,boolean success) {
        this.code = code;
        this.message = message;
        this.success = success;
    }

    public static Result SUCCESS(){
        return new Result(ResultCode.SUCCESS);
    }

    public static Result ERROR(){
        return new Result(ResultCode.SERVER_ERROR);
    }

    public static Result FAIL(){
        return new Result(ResultCode.FAIL);
    }
}
```

（2）创建类PageResult ，用于返回分页结果

```
package com.ihrm.common.entity;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class PageResult<T> {
    private Long total;
    private List<T> rows;
}
```

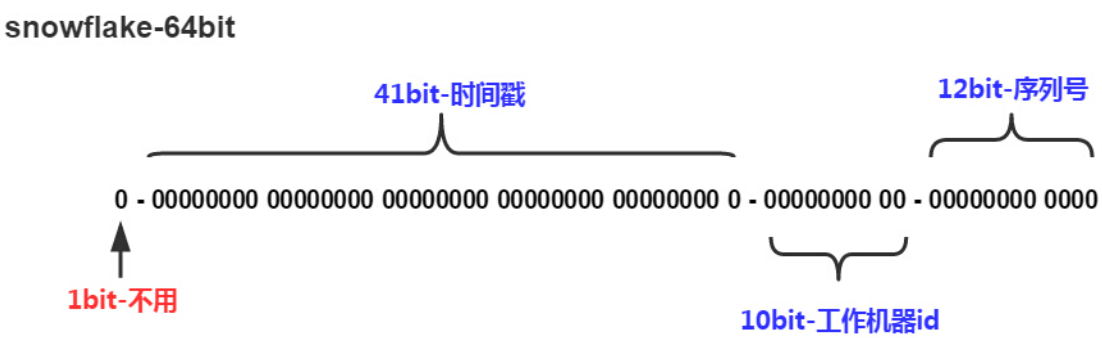
4.4.3 返回码定义类

```
public enum ResultCode {

    SUCCESS(true,10000,"操作成功！"),
    //---系统错误返回码-----
    FAIL(false,10001,"操作失败"),
    UNAUTHENTICATED(false,10002,"您还未登录"),
    UNAUTHORISE(false,10003,"权限不足"),
```

```
    SERVER_ERROR(false,99999,"抱歉，系统繁忙，请稍后重试！");

    //---用户操作返回码----
    //---企业操作返回码----
    //---权限操作返回码----
    //---其他操作返回码----

    //操作是否成功
    boolean success;
    //操作代码
    int code;
    //提示信息
    String message;

    ResultCode(boolean success,int code, String message){
        this.success = success;
        this.code = code;
        this.message = message;
    }

    public boolean success() {
        return success;
    }

    public int code() {
        return code;
    }

    public String message() {
        return message;
    }

}
```

4.4.4 分布式ID生成器

目前微服务架构盛行，在分布式系统中的操作中都会有一些全局性ID的需求,所以我们不能使用数据库本身的自增功能来产生主键值，只能由程序来生成唯一的主键值。我们采用的是开源的twitter(非官方中文惯称：推特.是国外的一个网站，是一个社交网络及微博客服务) 的snowflake （雪花）算法。



各个段解析：

分段	作用	说明
1bit	保留（不用）	---
41bit	时间戳，精确到毫秒	最多可以支持69年的跨度
5bit	机器id	最多支持2的5次方（32）个节点
5bit	业务编码	最多支持2的5次方（32）个节点
12bit	毫秒内的计数器	每个节点每毫秒最多产生2的12次方（4096）个id

> 默认情况下41bit的时间戳可以支持该算法使用到2082年，10bit的工作机器id可以支持1024台机器，序列号支持1毫秒产生4096个自增序列id . SnowFlake的优点是，整体上按照时间自增排序，并且整个分布式系统内不会产生ID碰撞(由数据中心ID和机器ID作区分)，并且效率较高，经测试，SnowFlake每秒能够产生26万ID左右

```
//雪花算法代码实现
public class IdWorker {
    // 时间起始标记点，作为基准，一般取系统的最近时间（一旦确定不能变动）
    private final static long twepoch = 1288834974657L;
    // 机器标识位数
    private final static long workerIdBits = 5L;
    // 数据中心标识位数
    private final static long datacenterIdBits = 5L;
    // 机器ID最大值
    private final static long maxWorkerId = -1L ^ (-1L << workerIdBits);
    // 数据中心ID最大值
    private final static long maxDatacenterId = -1L ^ (-1L << datacenterIdBits);
    // 毫秒内自增位
    private final static long sequenceBits = 12L;
    // 机器ID偏左移12位
    private final static long workerIdShift = sequenceBits;
    // 数据中心ID左移17位
    private final static long datacenterIdShift = sequenceBits + workerIdBits;
    // 时间毫秒左移22位
    private final static long timestampLeftShift = sequenceBits + workerIdBits +
datacenterIdBits;

    private final static long sequenceMask = -1L ^ (-1L << sequenceBits);
    /* 上次生产id时间戳 */
    private static long lastTimestamp = -1L;
    // 0，并发控制
    private long sequence = 0L;

    private final long workerId;
    // 数据标识id部分
    private final long datacenterId;

    public IdWorker(){
        this.datacenterId = getDatacenterId(maxDatacenterId);
```

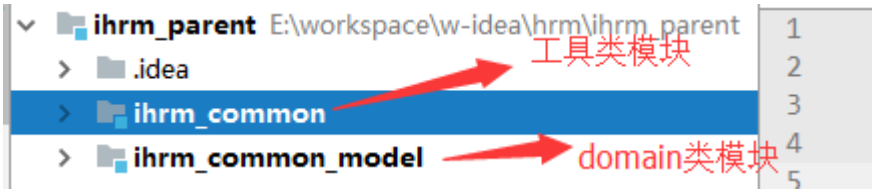
```
        this.workerId = getMaxWorkerId(datacenterId, maxWorkerId);
    }
    /**
     * @param workerId
     *            工作机器ID
     * @param datacenterId
     *            序列号
     */
    public IdWorker(long workerId, long datacenterId) {
        if (workerId > maxWorkerId || workerId < 0) {
            throw new IllegalArgumentException(String.format("worker Id can't be
greater than %d or less than 0", maxWorkerId));
        }
        if (datacenterId > maxDatacenterId || datacenterId < 0) {
            throw new IllegalArgumentException(String.format("datacenter Id can't be
greater than %d or less than 0", maxDatacenterId));
        }
        this.workerId = workerId;
        this.datacenterId = datacenterId;
    }
    /**
     * 获取下一个ID
     *
     * @return
     */
    public synchronized long nextId() {
        long timestamp = timeGen();
        if (timestamp < lastTimestamp) {
            throw new RuntimeException(String.format("Clock moved backwards.  Refusing
to generate id for %d milliseconds", lastTimestamp - timestamp));
        }

        if (lastTimestamp == timestamp) {
            // 当前毫秒内，则+1
            sequence = (sequence + 1) & sequenceMask;
            if (sequence == 0) {
                // 当前毫秒内计数满了，则等待下一秒
                timestamp = tilNextMillis(lastTimestamp);
            }
        } else {
            sequence = 0L;
        }
        lastTimestamp = timestamp;
        // ID偏移组合生成最终的ID，并返回ID
        long nextId = ((timestamp - twepoch) << timestampLeftShift)
                | (datacenterId << datacenterIdShift)
                | (workerId << workerIdShift) | sequence;

        return nextId;
    }

    private long tilNextMillis(final long lastTimestamp) {
        long timestamp = this.timeGen();
```

```
        while (timestamp <= lastTimestamp) {
            timestamp = this.timeGen();
        }
        return timestamp;
    }

    private long timeGen() {
        return System.currentTimeMillis();
    }

    /**
     * <p>
     * 获取 maxWorkerId
     * </p>
     */
    protected static long getMaxWorkerId(long datacenterId, long maxWorkerId) {
        StringBuffer mpid = new StringBuffer();
        mpid.append(datacenterId);
        String name = ManagementFactory.getRuntimeMXBean().getName();
        if (!name.isEmpty()) {
         /*
          * GET jvmPid
          */
            mpid.append(name.split("@")[0]);
        }
      /*
       * MAC + PID 的 hashcode 获取16个低位
       */
        return (mpid.toString().hashCode() & 0xffff) % (maxWorkerId + 1);
    }

    /**
     * <p>
     * 数据标识id部分
     * </p>
     */
    protected static long getDatacenterId(long maxDatacenterId) {
        long id = 0L;
        try {
            InetAddress ip = InetAddress.getLocalHost();
            NetworkInterface network = NetworkInterface.getByInetAddress(ip);
            if (network == null) {
                id = 1L;
            } else {
                byte[] mac = network.getHardwareAddress();
                id = ((0x000000FF & (long) mac[mac.length - 1])
                        | (0x0000FF00 & (((long) mac[mac.length - 2]) << 8))) >> 6;
                id = id % (maxDatacenterId + 1);
            }
        } catch (Exception e) {
            System.out.println(" getDatacenterId: " + e.getMessage());
        }
        return id;
```

```
    }
}
```

4.5 搭建公共的实体类模块

（1）构建公共子模块ihrm_common_model



（2）引入坐标

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>com.ihrm</groupId>
        <artifactId>ihrm_common</artifactId>
        <version>1.0-SNAPSHOT</version>
    </dependency>
</dependencies>
```

5 企业微服务-企业CRUD

5.1 模块搭建

（1）搭建企业微服务模块ihrm_company, pom.xml引入依赖

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<dependency>
    <groupId>com.ihrm</groupId>
    <artifactId>ihrm_common</artifactId>
    <version>1.0-SNAPSHOT</version>
</dependency>
```

（2）添加配置文件application.yml

```
server:
  port: 9001
spring:
  application:
    name: ihrm-company #指定服务名
  datasource:
    driver-class-name: com.mysql.jdbc.Driver
    url: jdbc:mysql://localhost:3306/ihrm?useUnicode=true&characterEncoding=utf8
    username: root
    password: 111111
  jpa:
    database: MySQL
    show-sql: true
    open-in-view: true
```

（3）配置启动类

```
@SpringBootApplication(scanBasePackages = "com.ihrm")
@EntityScan("com.ihrm")
public class CompanyApplication {

    public static void main(String[] args) {
        SpringApplication.run(CompanyApplication.class, args);
    }

    @Bean
    public IdWorker idWorkker() {
        return new IdWorker(1, 1);
    }
}
```

5.2 企业管理-CRUD

5.2.1 表结构分析

```
CREATE TABLE `co_company` (
  `id` varchar(40) NOT NULL COMMENT 'ID',
  `name` varchar(255) NOT NULL COMMENT '公司名称',
  `manager_id` varchar(255) NOT NULL COMMENT '企业登录账号ID',
  `version` varchar(255) DEFAULT NULL COMMENT '当前版本',
  `renewal_date` datetime DEFAULT NULL COMMENT '续期时间',
  `expiration_date` datetime DEFAULT NULL COMMENT '到期时间',
  `company_area` varchar(255) DEFAULT NULL COMMENT '公司地区',
  `company_address` text COMMENT '公司地址',
  `business_license_id` varchar(255) DEFAULT NULL COMMENT '营业执照-图片ID',
  `legal_representative` varchar(255) DEFAULT NULL COMMENT '法人代表',
  `company_phone` varchar(255) DEFAULT NULL COMMENT '公司电话',
  `mailbox` varchar(255) DEFAULT NULL COMMENT '邮箱',
  `company_size` varchar(255) DEFAULT NULL COMMENT '公司规模',
  `industry` varchar(255) DEFAULT NULL COMMENT '所属行业',
  `remarks` text COMMENT '备注',
```

```
  `audit_state` varchar(255) DEFAULT NULL COMMENT '审核状态',
  `state` tinyint(2) NOT NULL DEFAULT '1' COMMENT '状态',
  `balance` double NOT NULL COMMENT '当前余额',
  `create_time` datetime NOT NULL COMMENT '创建时间'
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

5.2.2 完成企业增删改查操作

（1）实体类（domain）

```
@Entity
@Table(name = "co_company")
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Company implements Serializable {
    private static final long serialVersionUID = 594829320797158219L;
    //ID
    @Id
    private String id;
    /**
     * 公司名称
     */
    private String name;
    /**
     * 企业登录账号ID
     */
    private String managerId;
    /**
     * 当前版本
     */
    private String version;
    /**
     * 续期时间
     */
    private Date renewalDate;
    /**
     * 到期时间
     */
    private Date expirationDate;
    /**
     * 公司地区
     */
    private String companyArea;
    /**
     * 公司地址
     */
    private String companyAddress;
    /**
     * 营业执照-图片ID
     */
    private String businessLicenseId;
    /**
```

```
     * 法人代表
     */
    private String legalRepresentative;
    /**
     * 公司电话
     */
    private String companyPhone;
    /**
     * 邮箱
     */
    private String mailbox;
    /**
     * 公司规模
     */
    private String companySize;
    /**
     * 所属行业
     */
    private String industry;
    /**
     * 备注
     */
    private String remarks;
    /**
     * 审核状态
     */
    private String auditState;
    /**
     * 状态
     */
    private Integer state;
    /**
     * 当前余额
     */
    private Double balance;
    /**
     * 创建时间
     */
    private Date createTime;
}
```

（2）持久层（dao）

```
/**
 * 企业数据访问接口
 */
public interface CompanyDao extends JpaRepository<Company, String>,
JpaSpecificationExecutor<Company> {
}
```

JpaRepository提供了基本的增删改查 JpaSpecificationExecutor用于做复杂的条件查询

（3）业务逻辑层（service）

```
@Service
public class CompanyService {
    @Autowired
    private CompanyDao companyDao;
    @Autowired
    private IdWorker idWorker;

    /**
     * 添加企业
     *
     * @param company 企业信息
     */
    public Company add(Company company) {
        company.setId(idWorker.nextId() + "");
        company.setCreateTime(new Date());
        company.setState(1);    //启用
        company.setAuditState("0"); //待审核
        company.setBalance(0d);
        return companyDao.save(company);
    }

    public Company update(Company company) {
        return companyDao.save(company);
    }

    public Company findById(String id) {
        return companyDao.findById(id).get();
    }

    public void deleteById(String id) {
        companyDao.deleteById(id);
    }

    public List<Company> findAll() {
        return companyDao.findAll();
    }
}
```

（4）控制器

```
@RestController
@RequestMapping("/company")
public class CompanyController{
    @Autowired
    private CompanyService companyService;
    /**
    * 添加企业
    */
    @RequestMapping(value = "", method = RequestMethod.POST)
    public Result add(@RequestBody Company company) throws Exception {
        companyService.add(company);
```

```
        return Result.SUCCESS();
    }

    /**
    * 根据id更新企业信息
    */
    @RequestMapping(value = "/{id}", method = RequestMethod.PUT)
    public Result update(@PathVariable(name = "id") String id, @RequestBody Company
company) throws Exception {
        Company one = companyService.findById(id);
        one.setName(company.getName());
        one.setRemarks(company.getRemarks());
        one.setState(company.getState());
        one.setAuditState(company.getAuditState());
        companyService.update(company);
        return Result.SUCCESS();
    }

    /**
    * 根据id删除企业信息
    */
    @RequestMapping(value = "/{id}", method = RequestMethod.DELETE)
    public Result delete(@PathVariable(name = "id") String id) throws Exception {
        companyService.deleteById(id);
        return Result.SUCCESS();
    }

    /**
     * 根据ID获取公司信息
     */
    @RequestMapping(value = "/{id}", method = RequestMethod.GET)
    public Result findById(@PathVariable(name = "id") String id) throws Exception {
        Company company = companyService.findById(id);
        return new Result(ResultCode.SUCCESS);
    }

    /**
     * 获取企业列表
     */
    @RequestMapping(value = "", method = RequestMethod.GET)
    public Result findAll() throws Exception {
        List<Company> companyList = companyService.findAll();
        return new Result(ResultCode.SUCCESS);
    }

}
```

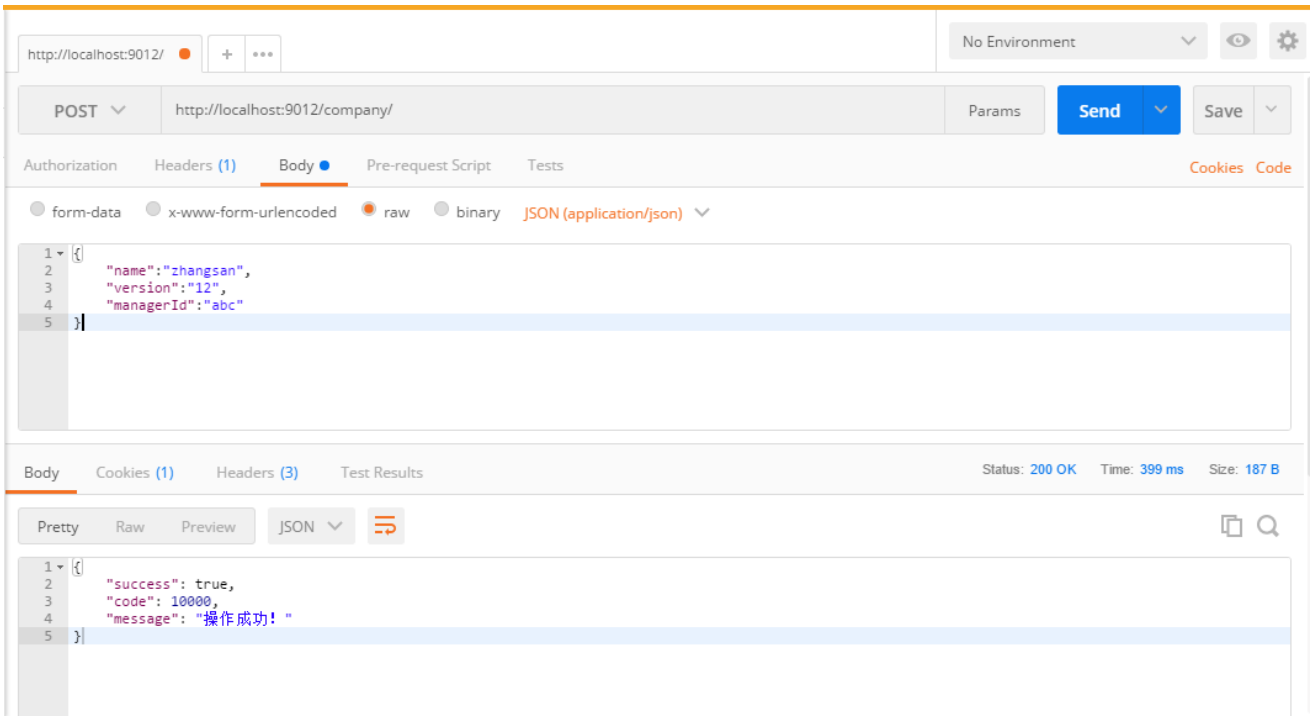
5.2.3 测试

（1）测试工具postman

Postman提供功能强大的Web API & HTTP 请求调试。软件功能非常强大，界面简洁明晰、操作方便快捷，设计得很人性化，能够发送任何类型的HTTP 请求 (GET, HEAD, POST, PUT..)，附带任何数量的参数。

使用资料中提供的postman安装包进行安装，注册成功之后即可使用

（2）使用postman测试企业接口



5.3 公共异常处理

为了使我们的代码更容易维护,同时给用户最好的用户体验，有必要对系统中可能出现的异常进行处理。spring提供了@ControllerAdvice注解和@ExceptionHandler可以很好的在控制层对异常进行统一处理

（1）添加自定义的异常

```
package com.ihrm.common.exception;

import com.ihrm.common.entity.ResultCode;
import lombok.Getter;

@Getter
public class CommonException extends RuntimeException {

    private static final long serialVersionUID = 1L;

    private ResultCode code = ResultCode.SERVER_ERROR;

    public CommonException(){}

    public CommonException(ResultCode resultCode) {
        super(resultCode.message());
        this.code = resultCode;
    }
}
```

（2）配置公共异常处理

```
package com.ihrm.common.exception;

import com.alibaba.fastjson.JSON;
import com.ihrm.common.entity.Result;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseBody;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

/**
 * 全局异常处理
 */
@ControllerAdvice
public class BaseExceptionHandler {


    @ResponseBody
    @ExceptionHandler(value = Exception.class)
    public Result error(HttpServletRequest request, HttpServletResponse response,
Exception e) throws IOException {
        e.printStackTrace();
        if (e.getClass() == CommonException.class) {
            CommonException ce = (CommonException) e;
            return new Result(ce.getCode());
        } else {
            return Result.ERROR();
        }
    }
}
```

5.4 跨域处理

跨域是什么？浏览器从一个域名的网页去请求另一个域名的资源时，域名、端口、协议任一不同，都是跨域 。我们是采用前后端分离开发的，也是前后端分离部署的，必然会存在跨域问题。 怎么解决跨域？很简单，只需要在controller类上添加注解@CrossOrigin 即可！这个注解其实是CORS的实现。 CORS(Cross-Origin Resource Sharing, 跨源资源共享)是W3C出的一个标准，其思想是使用自定义的HTTP头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功，还是应该失败。因此，要想实现CORS进行跨域，需要服务器进行一些设置，同时前端也需要做一些配置和分析。本文简单的对服务端的配置和前端的一些设置进行分析。