GEOVIS iCenter

空天大数据共享服务云平台

V1.1.2

二次开发教程

中科星图股份有限公司
2020 年 1 月

目 录

一、概述

（一）目的

本文档提供 GeoIoT 数据接口说明，服务扩展说明，数据扩展说明，主要给项目开发人员使用，帮助项目开发人员能在 iCenter 现有能力的基础上进行扩展。

该文档主要包括 GeoIoT 开发说明，服务扩展开发教程和数据扩展开发教程，每个部分均附带可运行的示例代码。该文档不包括 iCenter 所有接口的调用说明，具体接口调用说明，参见 iCenter 接口文档。

（二）适用范围

本文档适合 iCenter 的系统设计人员和程序开发人员使用。

（三）开源库版本约束

iCenter1.1.2 上的服务，都是基于 Spring Cloud 进行开发，用户也只能基于 Spring Cloud 进行二次开发，暂不支持其他框架的开发。所有开发的服务均需要遵循 Spring Cloud 的规范。在 iCenter1.1.2 上开发的服务需要使用的 Spring Cloud 的版本如下：

开源库	版本
jre	1.8
Spring Cloud	Greenwich.RC2
Spring Cloud Netflix	2.1.0.RC3
Spring Boot	2.1.1.RELEASE

注意：如果使用的库的版本和上述表格中的不一致，可能会导致一些不可预料的错误；一个典型的配置了指定版本的 Spring 的 pom 文件如下所示：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
```

```
        <groupId>com.geovis</groupId>
        <artifactId>demo</artifactId>
        <version>1.1.2-SNAPSHOT</version>
        <packaging>jar</packaging>
        <name>icenter112 demo project</name>
        <!-- 指定Spring Boot的版本为2.1.1.RELEASE -->
        <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>2.1.1.RELEASE</version>
        </parent>
        <properties>

<project.build.sourceEncoding>UTF-8<project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEnc
oding>
            <java.version>1.8</java.version>
        </properties>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
            </dependency>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-cloud-starter-config
    </artifactId>
            </dependency>
            <dependency>
```

```
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-boot-starter-actuator
</artifactId>
        </dependency>
    </dependencies>
    <dependencyManagement>
        <dependencies>
            <!-- 指定Spring Cloud的版本为Greenwich.RC2 -->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>Greenwich.RC2</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <!-- 指定netflix版本为2.1.0.RC3 -->
            <dependency>
                <groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-netflix-dependencies</artifactId>
                <version>2.1.0.RC3</version>
            </dependency>
        </dependencies>
    </dependencyManagement>
    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

服务中还可以使用 gdal，gdal 版本为 2.3.2。如果需要使用 gdal，需要在 pom 文件中添加如下依赖:

```
<dependency>
    <groupId>org.gdal</groupId>
    <artifactId>gdal</artifactId>
    <version>2.3.2</version>
</dependency>
```

二、基础环境及使用方式

（一）基础环境说明

iCenter1.1.2 在安装时，默认安装了以下基础服务，用户可以在自己的服务中使用。

名称	版本	说明
postgres	11.4	postgresql 数据库，已经安装的插件及版本有 postgis（2.5.2）、pg_pathman（1.5.3）、rum（1.3.1）、zhparser（0.2.0）
redis	5	分布式缓存系统
rabbitmq	3.7	分布式消息服务

（二）postgresql 使用方式

在用户服务启动时，系统会将 postgresql 配置相关的环境变量写入到用户服务所在的环境中，用户服务可以直接通过这些环境变量来配置使用 postgresql，如下：

环境变量	说明
pg_host	数据库地址
pg_port	数据库端口
pg_username	数据库用户名
pg_password	数据库密码

在 Spring Cloud 中配置示例如下：

```
spring:
```

```
datasource:
  driver-class-name: org.postgresql.Driver
  url: jdbc:postgresql://${pg_host}:${pg_port}/db_name
  username: ${pg_username}
  password: ${pg_password}
```

提示：按照示例方法配置完成之后，在添加服务时，系统会检查数据库名称，如果数据库中没有对应的数据库，则会按照默认方式创建。

（三）redis 使用方式

同 postgresql 一样，在用户服务启动时，系统会将 redis 配置相关的环境变量写入到用户服务所在的环境中，用户服务可以直接通过这些环境变量来配置使用 redis。环境变量如下：

环境变量	说明
redis_host	redis 数据库地址
redis_port	redis 数据库端口
redis_username	redis 数据库用户名
redis_password	redis 数据库密码

在 Spring Cloud 中配置示例如下：

```
spring:
  redis:
    host: ${redis_host}
    port: ${redis_port}
```

（四）rabbitmq 使用方式

用户服务启动时，系统会将 rabbitmq 配置相关的环境变量写入到用户服务所在的环境中，用户服务可以直接通过这些环境变量来配置使用 rabbitmq。环境变量如下：

环境变量	说明
rabbitmq_host	rabbitmq 数据库地址
rabbitmq_port	rabbitmq 数据库端口

环境变量	说明
rabbitmq_username	rabbitmq 数据库用户名
rabbitmq_password	rabbitmq 数据库密码

在 Spring Cloud 中配置示例如下：

```
spring:
  rabbitmq:
    host: ${rabbitmq_host}
    port: ${rabbitmq_port}
    username: ${rabbitmq_username}
    password: ${rabbitmq_password}
```

（五）用户验证接入说明

user-annotation 用于在 iCenter 中，以注解的方式快速的对使用 Springboot 开发的 web 服务接口进行用户拦截

添加 maven 依赖，在 pom 文件下添加如下依赖，引用 user-annotation

```
<dependency>
   <groupId>com.geovis</groupId>
   <artifactId>user-annotation</artifactId>
   <version>1.1.2-SNAPSHOT</version>
</dependency>
```

需要在起始类上使用 EnableAuth 注解开启用户拦截，如下所示

```
@EnableAuth
@SpringBootApplication
public class App {
   public static void main(String[] args) {
      SpringApplication.run(App.class);
   }
}
```

对需要验证的接口使用 VarifyAuth 注解可以指定拦截的角色 id，如下所示

```
@VarifyAuth(userRoleIds = {"2","3"})
@RestController
public class TestController {
```

```
    @GetMapping("/test")
    public String hello() {
        return "hello";
    }
    @GetMapping("/test2")
    public String hello2() {
        return "hello2";
    }
}
```

EnableAuth 注解默认全局验证，并且默认拦截角色 id 为 3 的用户，可以通过参数 pathPatterns 指定要验证的接口，也可以使用参数 excludePatterns 指定要忽略的接口。并且可以通过参数 userRoleIds 指定拦截的角色 id。如下所示，将会验证**/api 中的所有接口，但是会忽略/api/hello2 接口，并且拦截用户角色 id 为 2**、3

```
@EnableAuth(userRoleIds = {"2","3"},pathPatterns = {"/api/**"},excludePatterns = {"/api/hello2"})
@SpringBootApplication
public class App {
   public static void main(String[] args) {
      SpringApplication.run(App.class);
   }
}
```

也可以通过使用 IgnoreAuth，忽略对相关接口的验证，IgnoreAuth 可以用在接口上，也可以用在类上，如下所示，将不会对**/test2**进行验证

```
@VarifyAuth(userRoleIds = {"2","3"})
@RestController
public class TestController {
    @GetMapping("/test")
    public String hello() {
        return "hello";
    }
    @IgnoreAuth
```

```
    @GetMapping("/test2")
    public String hello2() {
        return "hello2";
    }
}
```

各注解优先级

IgnoreAuth > VarifyAuth > EnableAuth.excludePatterns > EnableAuth.pathPatterns

（六）其他

系统在启动用户服务的时候，除了向用户服务所处的环境添加上述三个基础服务的环境变量之外，还添加了其他的环境变量供用户服务使用，具体如下：

环境变量	说明
volume_in_docker	用户服务可以写入数据的位置
volume_in_host	用户服务写入的数据实际上在宿主机上存储的位置

三、实时服务开发说明

GeoIot 是 icenter 的实时数据服务，提供实时位置数据的接入、处理、存储、查询等功能。为了给用户提供统一的数据接入，我们提供了数据接入 SDK，用户可方便、快捷得将自有的数据接入至服务。目前，实时服务支持 http、tcp、udp 和 kafka 四类通信协议。下面以 Java 语言为例，介绍实时服务的二次开发实例。

（一）文件说明

GeoIOT 的二次开发相关代码都在 GeoIOTSDK.zip 文件内，其中包含 GeoiotSenderUtil 和 GeoiotSimulator。其中 GeoiotSenderUtil 为 SDK 代码，GeoiotSimulator 为使用 SDK 进行模拟发送的示例代码。

基于 SDK 进行二次开发需要将 GeoiotSenderUtil 引入依赖。

（二）引入依赖

```
<dependency>
    <groupId>cn.com.geovis</groupId>
    <artifactId>geoiot-sender-util</artifactId>
```

```
    <version>1.0.0-snapshot</version>
</dependency>
```

如果无法下载依赖 jar 包，则在 GeoiotSenderUtil 目录下执行 mvn install，将 GeoiotSenderUtil 安装至本地 maven 仓库。

（三）对象说明

✓ GvEvent

GvEvent 为实时服务位置信息的数据结构体，其字段结构如下：

字段名	数据类型	说明
source_id	String	数据源 Id
target_type	String	目标类型 Id
target_id	String	目标 Id
target_name	String	目标名称
target_time	Long	目标位置时间
lon	Double	经度
lat	Double	纬度
alt	Double	海拔高度
heading	Double	目标朝向（方位角）
extend	String	扩展字段（json 样式字符串）

注：所有字段不能为空，否则无法通过校验。

✓ Sender

一个接口类，对数据接入进行了抽象，其中的接口如下：

```
    /**
 * 初始化 Sender，如创建连接、构造对象等
 */
void init() throws Exception;
/**
 * 实时数据发送
 */
void send(GvEvent event) throws Exception;
/**
 * 销毁 Sender，如删除连接、释放对象等
```

```
 */
void destory() throws Exception;
```

✓ KafkaSender

是 Sender 的实现类，用于通过 Kafka 向 GeoIot 发送实时数据。

```
    /**
 * KafkaSender 构造方法
 * @param servers kafka 集群服务连接
 * @param topic kafka 的 topic
 * @param properties kafka 其他参数，可为空
 */
public KafkaSender(String servers,String topic,Properties properties){
    this(servers,topic);
    kafkaProps = properties;
}
```

✓ TcpSender

是 Sender 的实现类，用于通过 tcp 向 GeoIot 发送实时数据。

```
    /**
 * TcpSender 构造方法
 * @param ip tcp Server 端 ip
 * @param port tcp Server 端口号
 * @param channelId 对应 Siddhi 脚本中@Source 中的 context 值
 */
public TcpSender(String ip,Integer port,String channelId){
    this.ip = ip;
    this.port = port;
    this.channelId = channelId;
}
```

✓ UdpSender

是 Sender 的实现类，用于通过 udp 向 GeoIot 发送实时数据。

```
    /**
 * UdpSender 构造方法
 * @param ip udp Server 端 ip
 * @param port udp Server 端口号
 * @param channelId 对应 Siddhi 脚本中@Source 中的 context 值
 */
public UcpSender(String ip,Integer port,String channelId){
    this.ip = ip;
    this.port = port;
    this.channelId = channelId;
}
```

✓ HttpPostSender

是 Sender 的实现类，用于通过 http 以 post 方式向 GeoIot 发送实时数据。

```
/**
 * HttpPostSender 构造方法
 * @param url url 的指定格式为 http://{ip}:{port}/api/v1/input/http
 */
public HttpPostSender(String url){
    this.url = url;
}
```

（四）使用说明

通过一个模拟器程序说明数据发送流程

```
public class Simulator {
       public static void main(String[] args) throws Exception {

       /**模拟 10 万目标 */
       if(targetCount==null||targetCount<=0){
           targetCount=100000;
       }
       /**初始化目标 */
```

```
for(int i=0;i<targetCount;++i){
    String name = "si_"+i;
    targetSet.add(TargetInfo.InitTarget(name));
}
/**根据配置文件确定构造 kafkaSender 还是 TcpSender */
if(tcpOrKafka.equals("kafka")){
    sender =  new KafkaSender(kafkaServers,kafkaTopic);
}else {
    sender = new TcpSender(tcpIp,tcpPort,tcpChannelId);
}
/**初始化 Sender */
sender.init();

while(true){
    for(TargetInfo target:targetSet){
        /**GvEvent 对象 */
        GvEvent data = new GvEvent();
        data.setSource_id("99");
        data.setTarget_id(target.getName());
        data.setTarget_name(target.getName());
        data.setAlt(0.0);
        data.setLon(target.getLon());
        data.setLat(target.getLat());
        data.setHeading(0.0);
        data.setExtend("");
        data.setTarget_type("00");
        data.setTarget_time(System.currentTimeMillis());
        /**发送消息 */
        sender.send(data);
        System.out.println( target.Move());
    }
    Thread.sleep(testSleep);
}
sender.destory();
```

```
    }
}
```

四、数据扩展开发说明

（一）原理说明

扩展数据功能是基于 SpringCloud 的 Eureka 为技术底层，通过使用 Eureka 的元数据注册和发现功能，将用户注册的服务接口发布到 icenter 的扩展数据功能中，扩展的数据类型按照指定的格式进行注册，能够在 icenter 中进行发现，通过用户的配置，添加到 icenter 平台中。

（二）接入说明

接入说明以 springBoot 框架开发来进行说明，其他的开发框架请参考此说明。

假设用户的服务名称是 DmDemoApplication，下面以 grib2 数据为例，提供 grib2 数据的解析功能，扩展的数据接口的路径为/api/v1/grib2/parser。

下面介绍详细的开发步骤

1.pom.xml 配置

扩展数据类型本身是一个 eureka 的客户端，需要在 pom.xml 中引入 Eureka 的依赖：

```
    <dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
</dependency>
```

2.增加启动注解

在启动项 appliaction 中添加 eureka 注释@EnableEurekaClient。

```
    @SpringBootApplication
@EnableEurekaClient
public class DmDemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DmDemoApplication.class, args);
   }
}
```

3.应用配置文件修改

添加 eureka 元数据注册所需的相关配置，需要在 application.yml(SpringBoot)或者 bootstratp.yml(SpringCloud)中进行配置。

```
    eureka:
  instance:
    prefer-ip-address: true
    instance-id:
${spring.application.name}(${spring.cloud.client.ip-address}:${server.port})
    lease-renewal-interval-in-seconds: 10


    #下面是需要注册到 eureka 元数据上的数据
    metadata-map:
      #字段说明
      #extend_datatype_grib2: 必须以 extend_datatype_为前缀来说明数据类型,后面的
grib2 是数据类型的标志
      #气象数据&grib2transfer&grib2&解析&parsefile&/parsefile&post
      #气象数据：数据类型的界面显示名称
      #.grib2：支持的数据类型的后缀名
      #解析：支持的方法的界面显示名称
      #parsefile：解析的字典名称，参考文档附录 1 的字典说明
      #/api/v1/grib2/parser:方法对应的路径名称
      #post:方法的调用方式
      extend_datatype_grib2:      气      象      数      据      &.grib2&      解      析
&parsefile&/api/v1/grib2/parser&post
```

4.接口样例代码

以 list 类型返回的解析完成的数据。

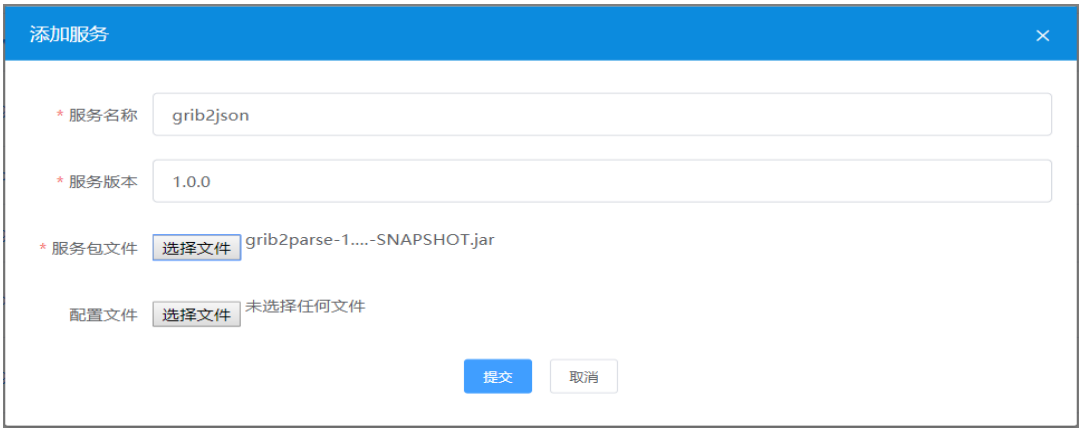
```
@ApiOperation(value = "grib2 解析", notes = "grib2 解析")
@PostMapping("/api/v1/grib2/parser")
public List< > grib2Parse(@Valid @RequestBody ParserParams params) {
    List<Map> list = new ArrayList<Map>();
            grib2ParseResult g2p = new grib2ParseResult();
        g2p.setData(data2);
        g2p.setGeo(geo);
```

```
        g2p.setLa1(String.valueOf(map.get("la1")));
        g2p.setLa2(String.valueOf(map.get("la2")));
        List<grib2ParseResult> result = new ArrayList<>();
        result.add(g2p);
        return result;
    }
```

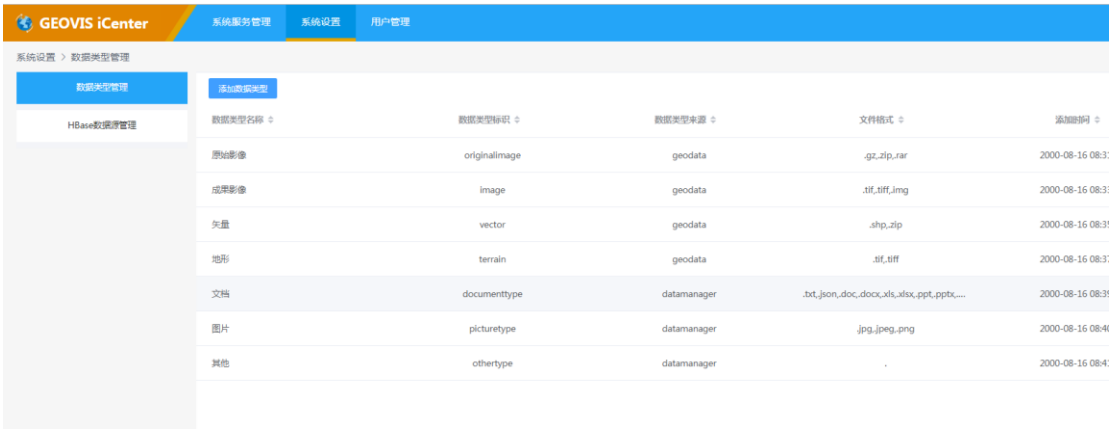
（三）扩展数据类型添加流程

添加扩展服务

选择“添加服务”，点击“提交”。



添加数据类型



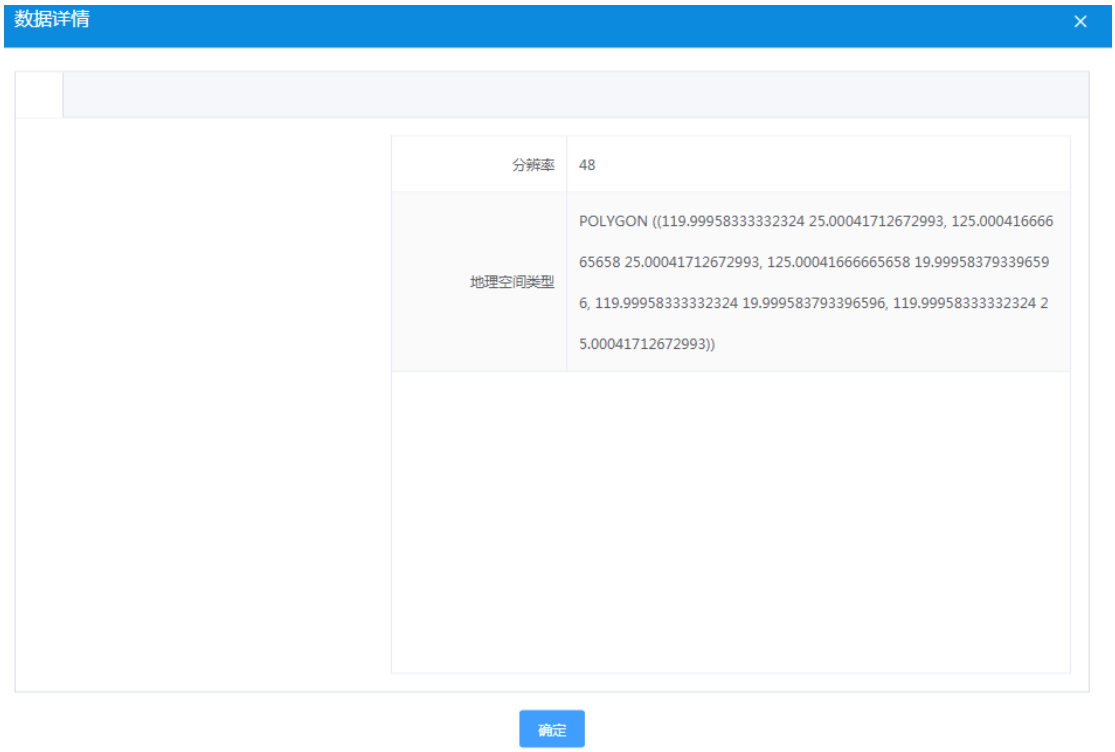
点击“添加数据类型”，选在服务来源，勾选需要添加的数据类型。

完成数据类型的添加，在数据类型列表中进行查看。

上传数据

选择气象数据，在详情中查看解析的结果。

查看解析结果



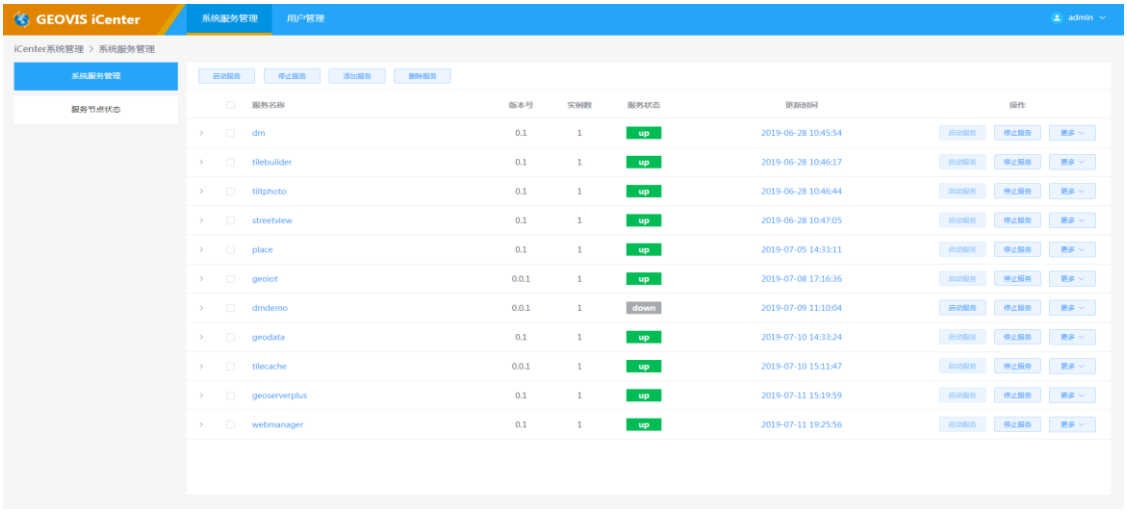
常用的扩展数据操作的标志

序号	操作类型	英文标识	
1	解析	parsefile	
2	发布	publishfile	
3	删除	deletefile	

五、远程调试

（一）开启调试模式

首先，进入服务管理页面，确保想要调试的服务处于停止状态，如果正在运行，点击“停止服务”将服务停止。



点击更多，在弹出的菜单中选择“调试”或“调试(启动时挂起)”

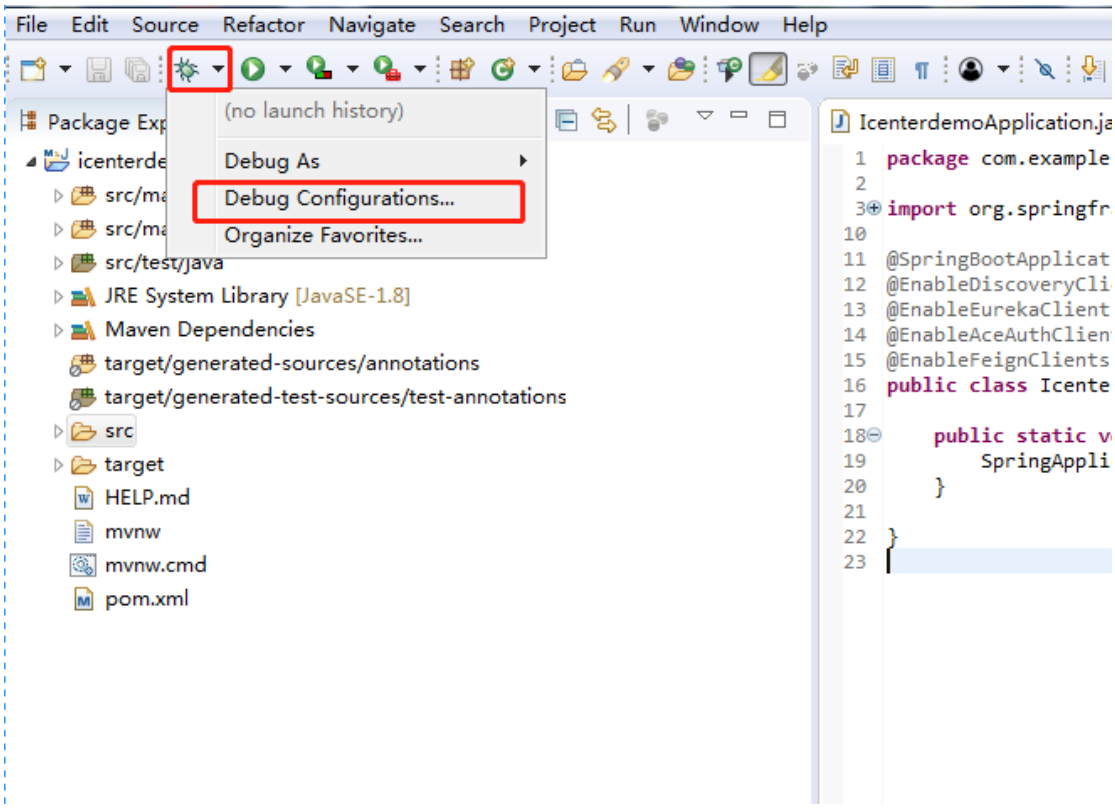
说明：如果选择“调试(启动时挂起)”，服务启动之后将会被挂起，直到有调试器远程连接到服务，服务开始运行；这种模式适合从程序入口开始进行调试。

在弹出的窗口中输入端口号，点击“确定”，开始以调试模式启动服务

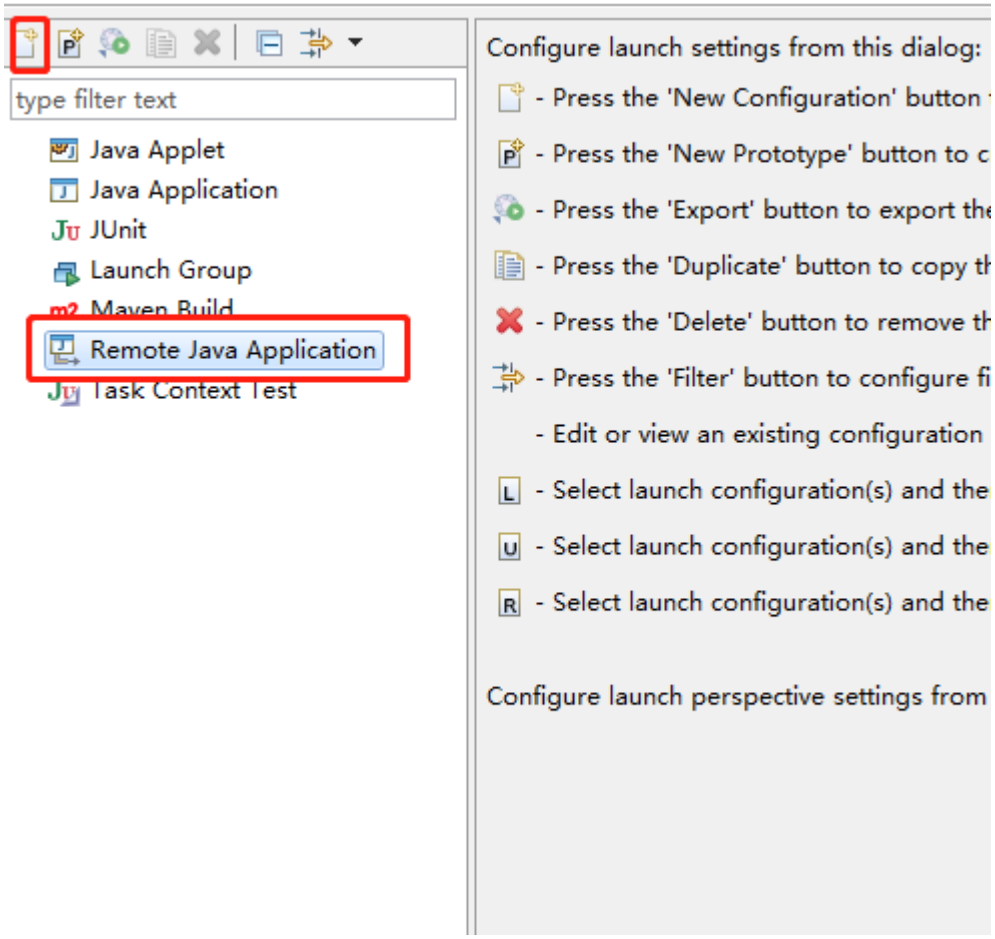
（二）在 IDE 中远程调试

1.eclipse 中的配置

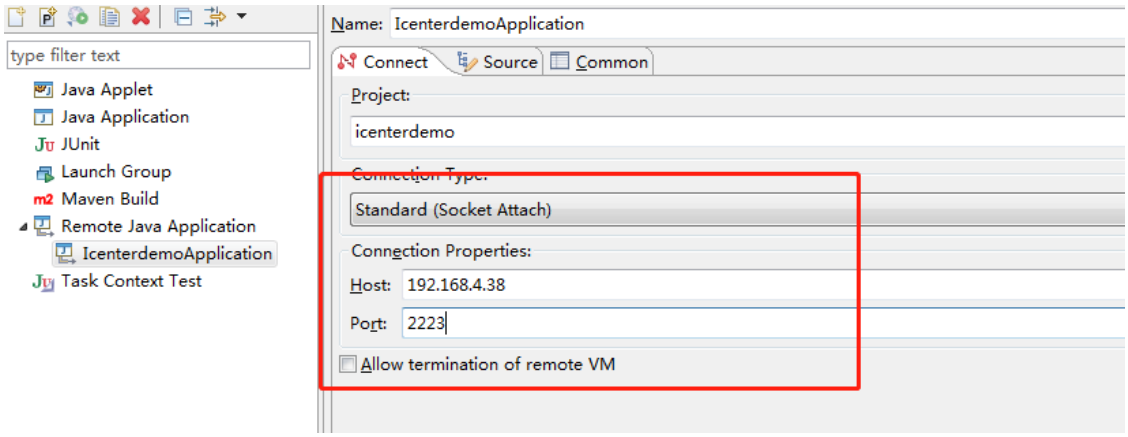
首先在 eclipse 中导入相应的工程代码，如图中所示，选择“Debug Configurations…”，添加调试配置



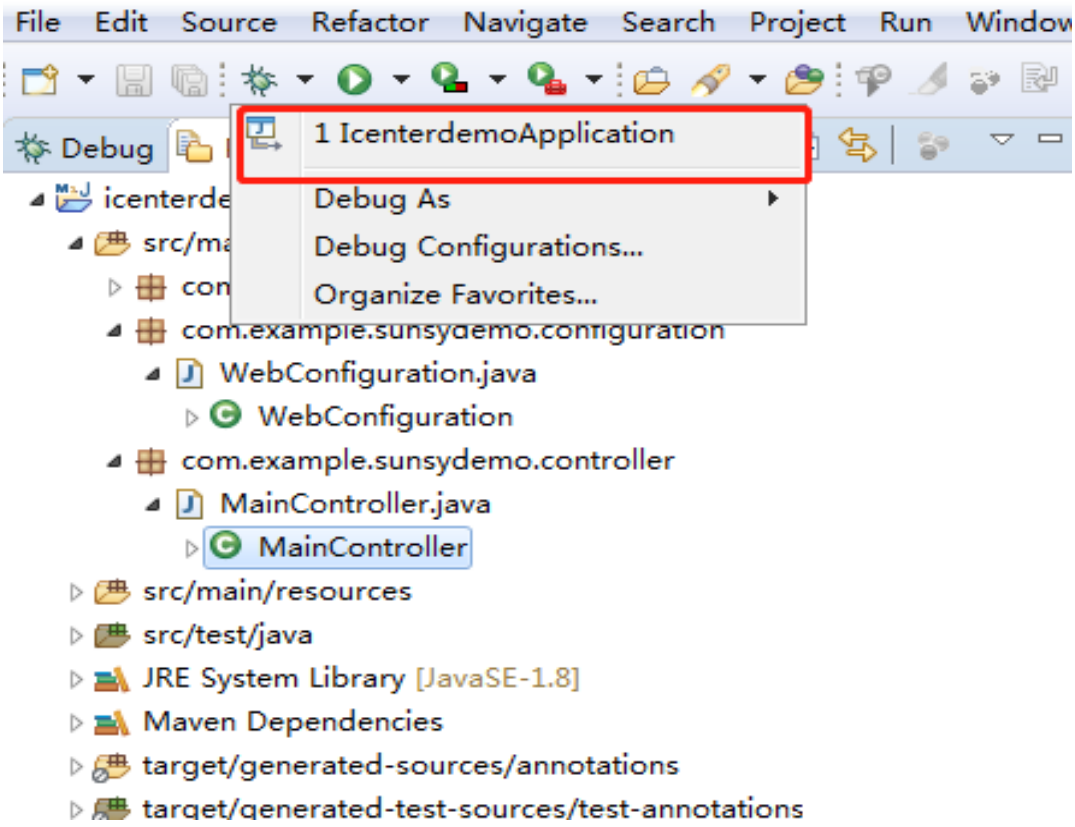
在弹出的对话框中，选择“Remote Java Application”，点击左上角的新建按钮，添加调试选项



编辑 Host 和 Port，Connection Type 选择 “Standard(Socket Attach)”，完成之后，点击“Applay”按钮，添加调试配置



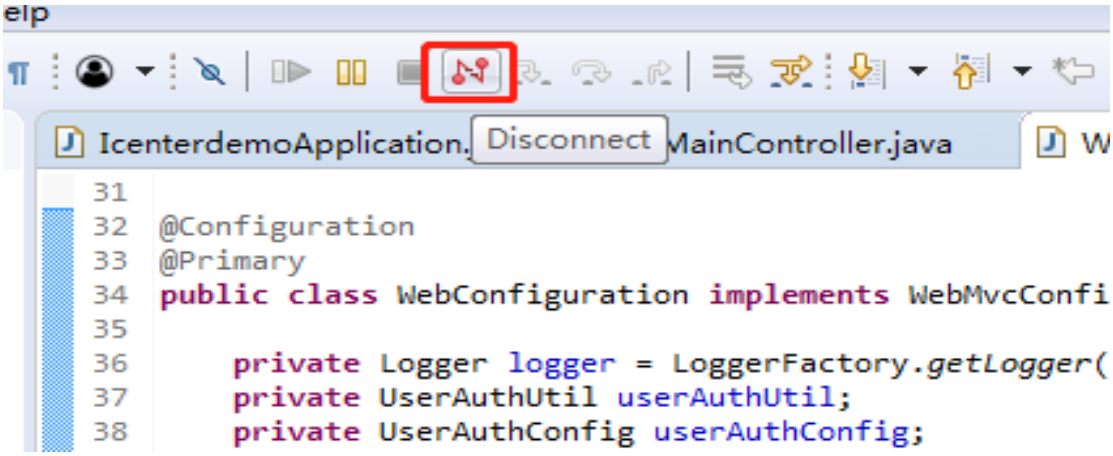
点击调试按钮边上的黑色三角形，选择刚才添加的配置，开始调试



选择调试配置，开始调试

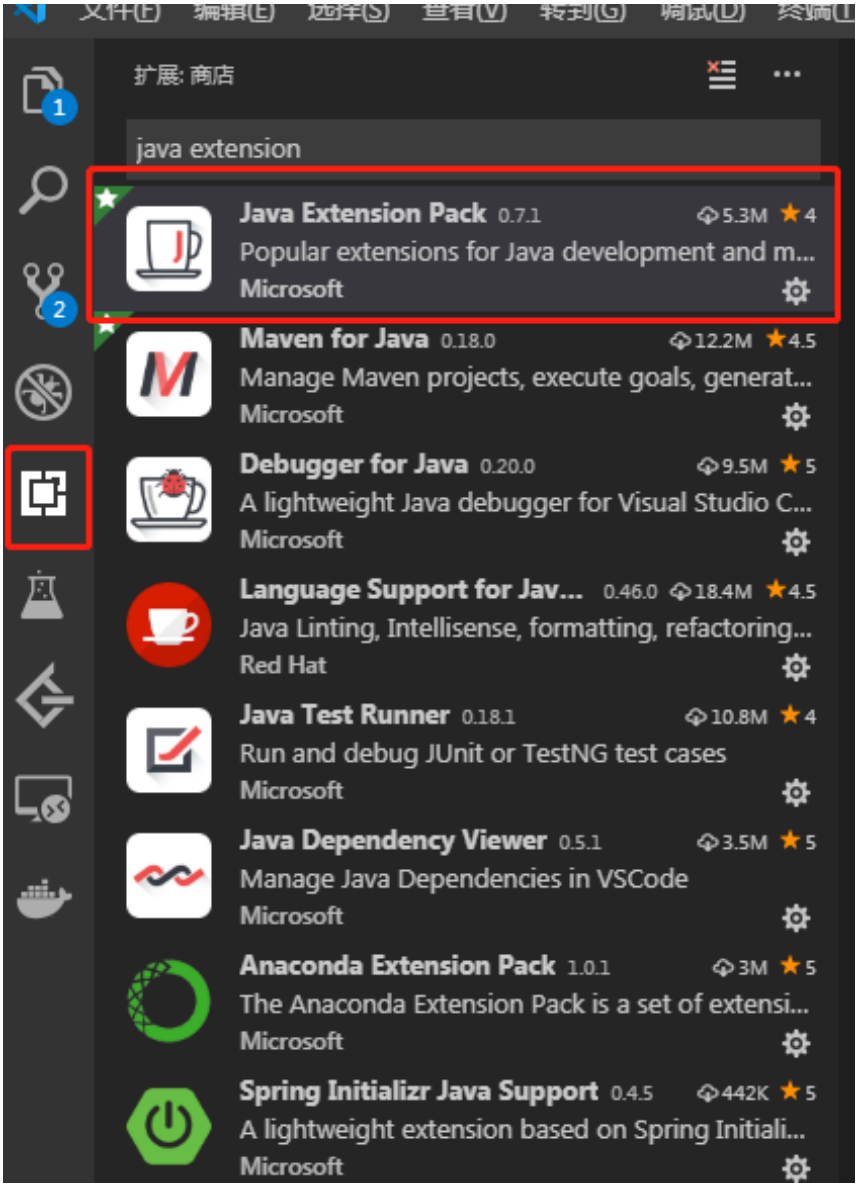
说明：开始调试时，eclipse 有可能没有自动跳转到调试页面，可以通过界面右上角的调试按钮切换过去

在调试界面，点击如图所示的图标，断开调试(断开之后，服务并没有停止)

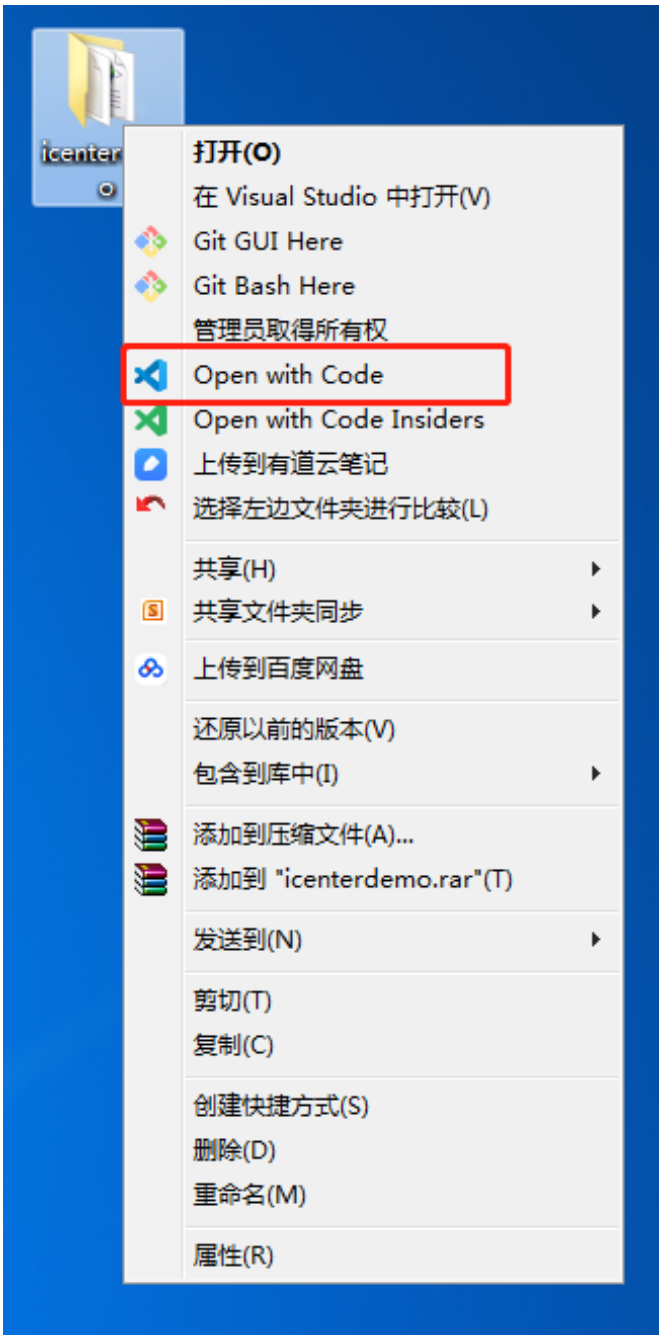


2.vscode 中的配置

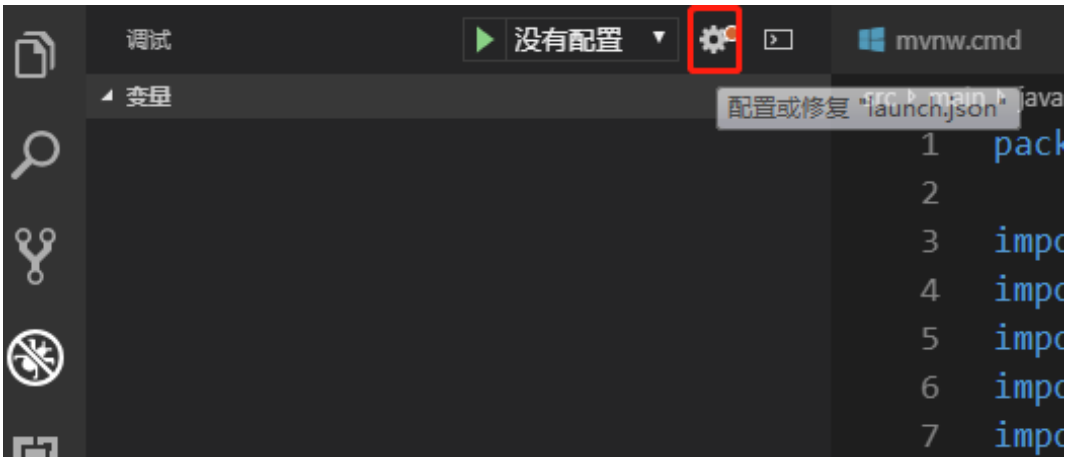
打开 vscode 的插件商店，搜索 java extension pack 并安装，等安装完成后根据提示重启 vscode。



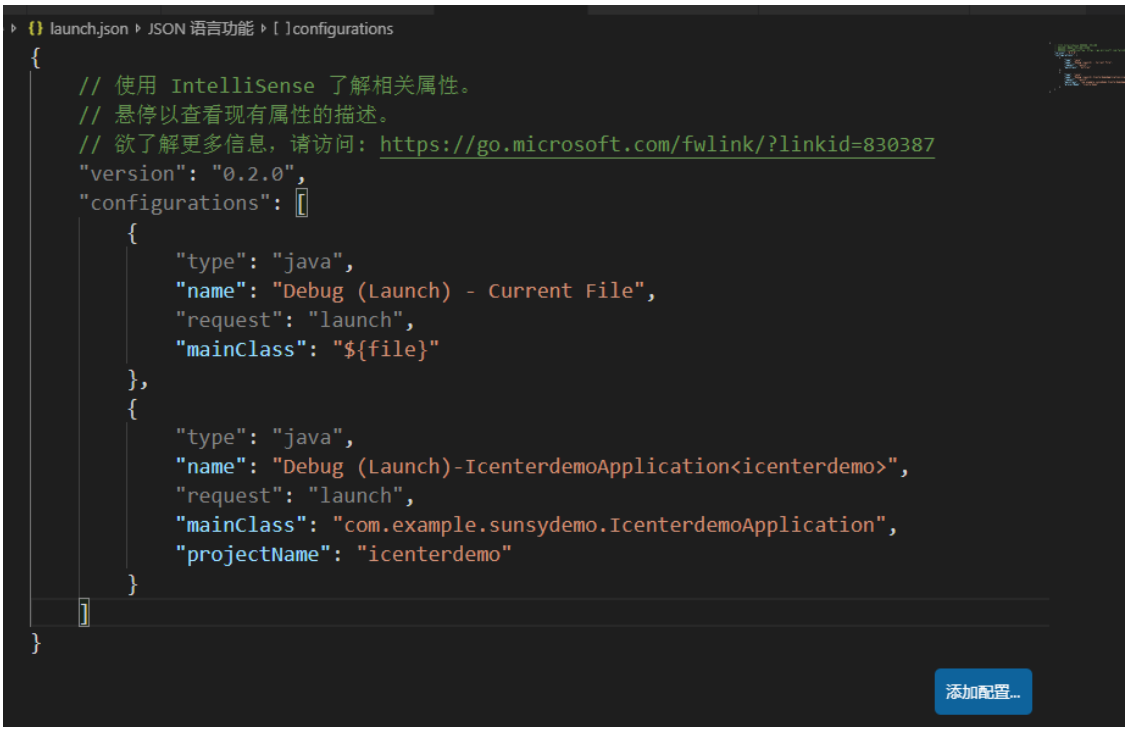
代码上单击右键，在弹出的菜单中选择“Open with Code”打开与要调试的服务匹配的代码。



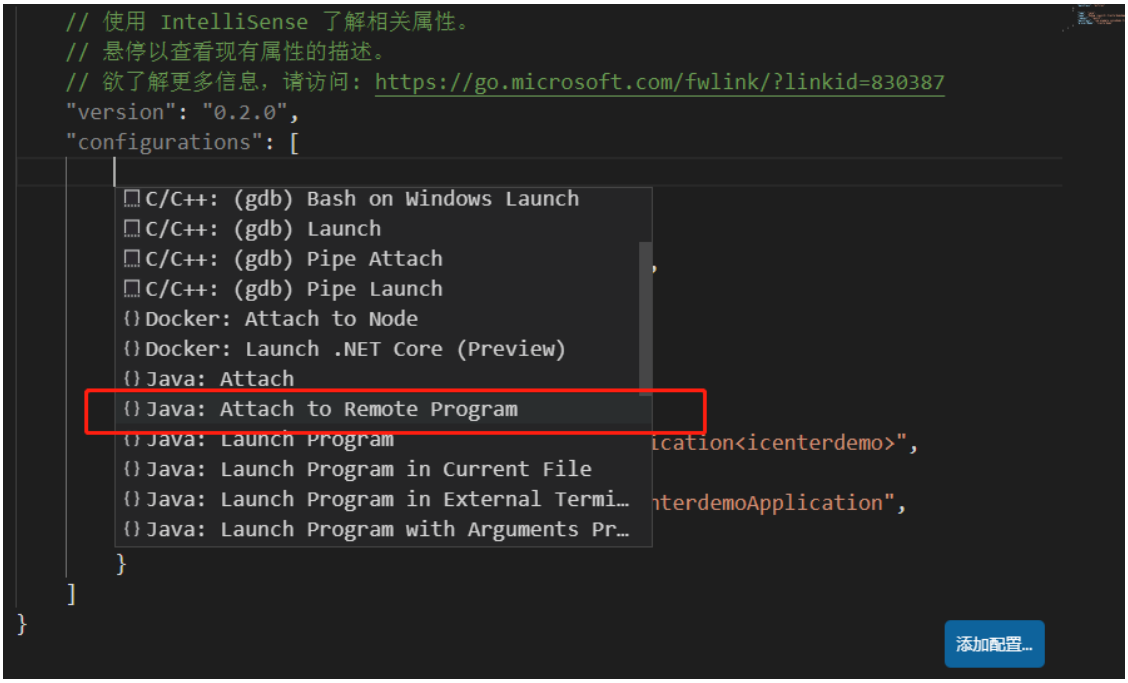
在主界面的左侧，选择调试标签  ，打开调试界面，点击上面的齿轮按钮，配置 launch.json



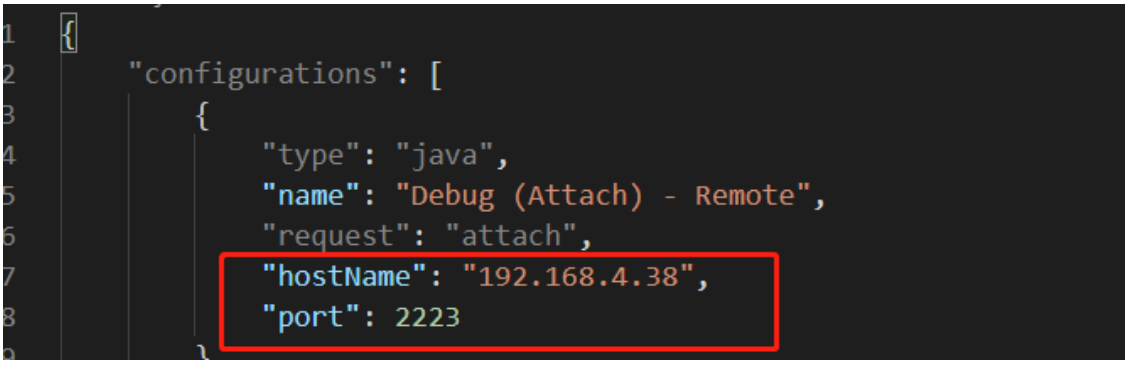
vscode 自动生成了默认的 launch



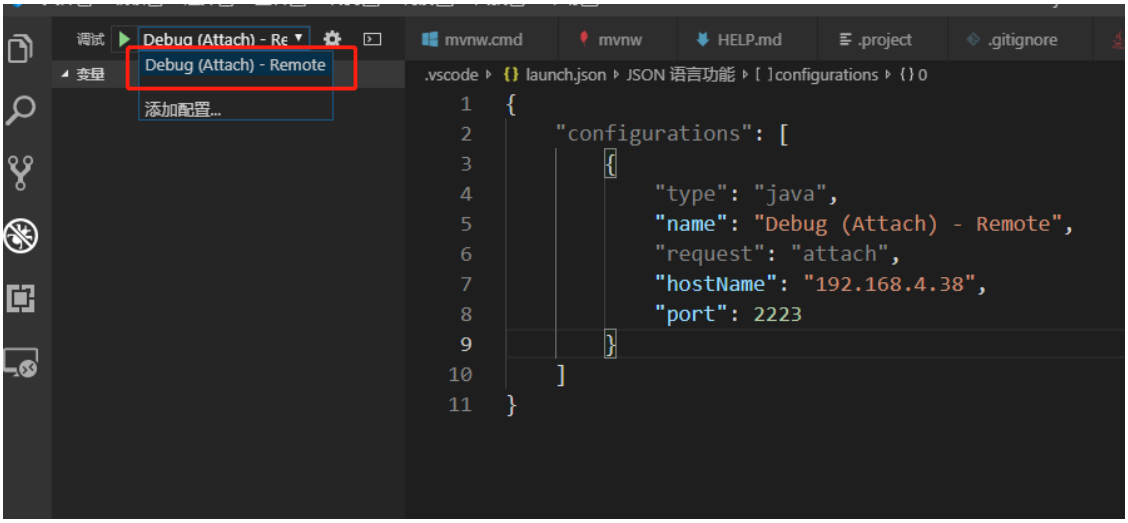
点击右下角的添加配置按钮，从弹出的菜单中选择 “Java: Attach to Remote Program”



在生成的配置中，修改 hostname 和 port 到相应的值，保存

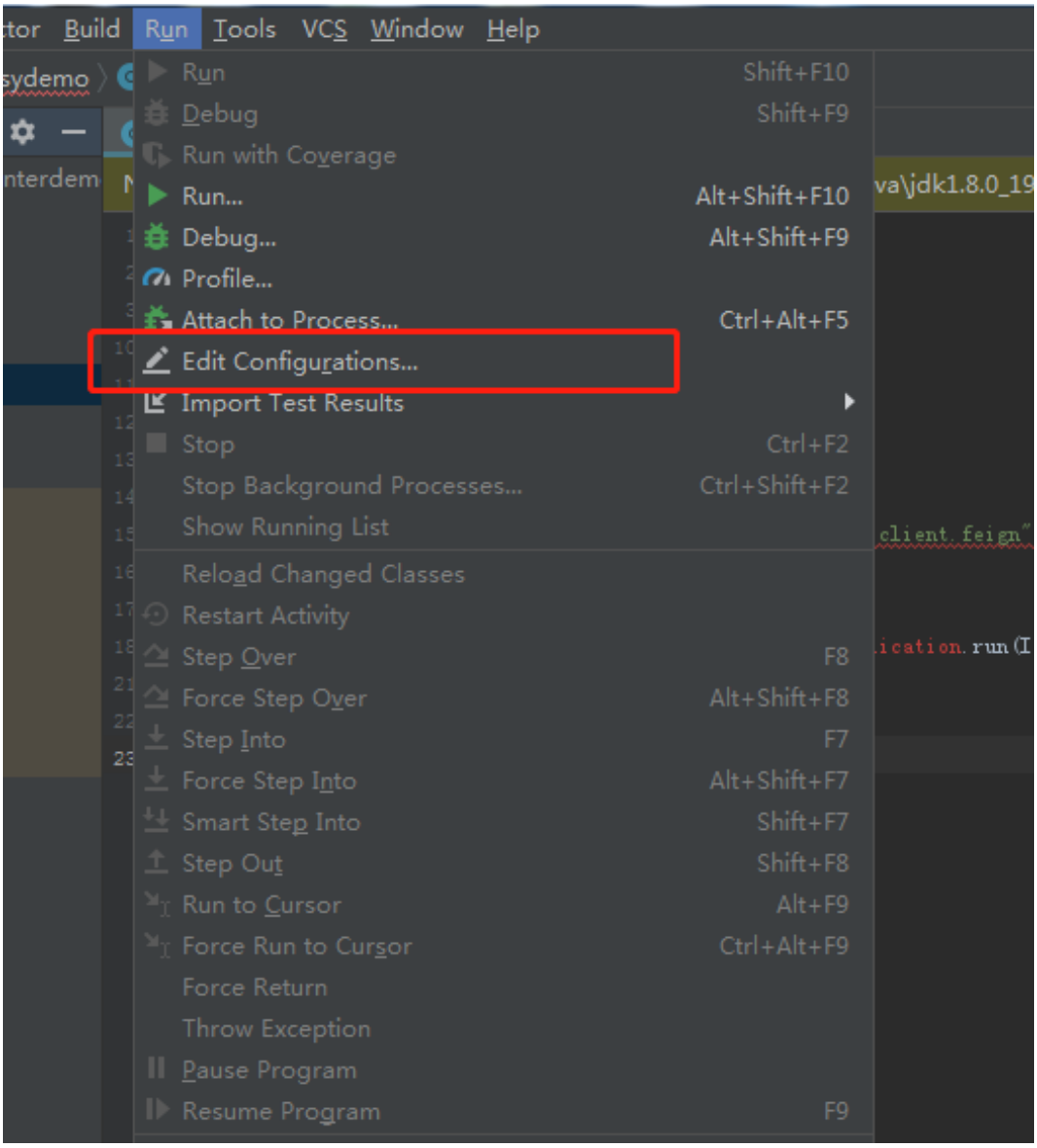


选中刚才设置的配置，按快捷键 F5 开始调试。

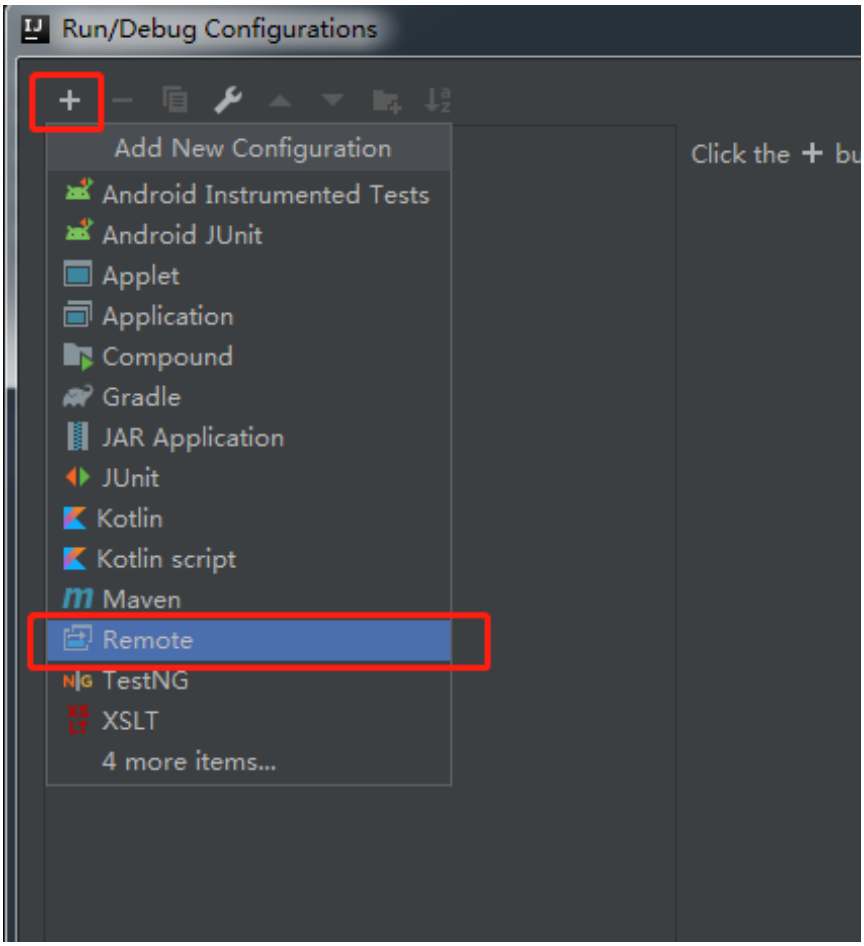


3.idea 中的配置

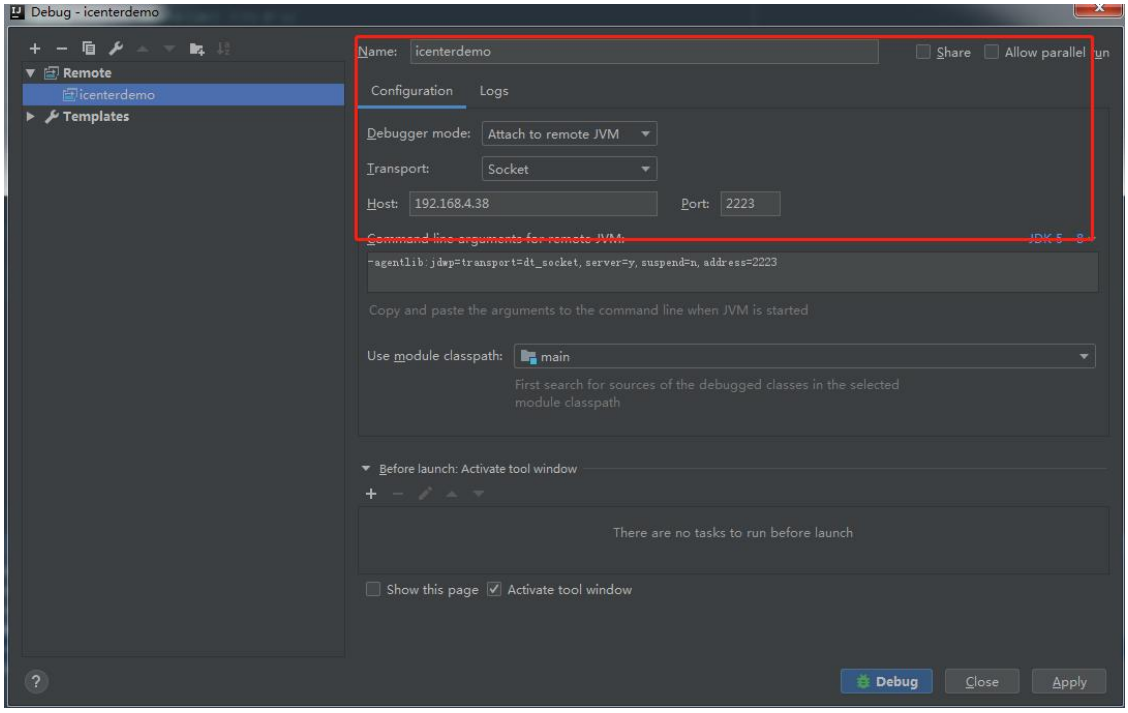
使用 idea 打开对应的项目，菜单中选择 “Run” -> “Edit Configurations…”



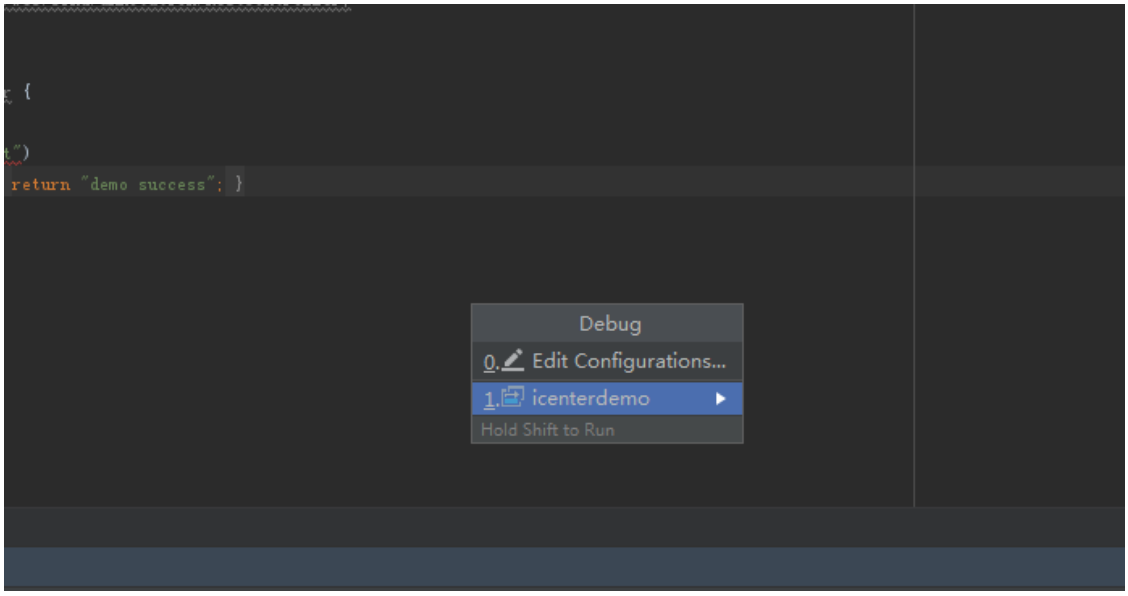
在弹出的对话框中，点击左上角的“+”号，选择“Remote”



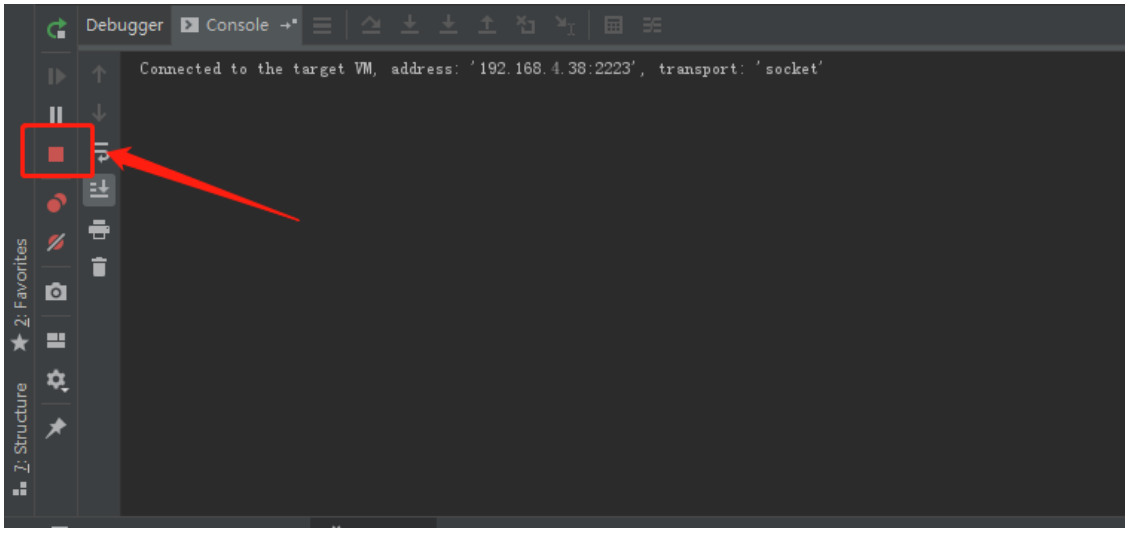
配置如图所示，注意 Debuger mode 和 Transport 的配置要和图中的相同，设置好 ip 和端口之后，点击“Applay”按钮保存配置



点击菜单中的“Run” -> “Debug…”，在弹出的菜单中选中刚才的配置，启动调试



调试完成之后，点击如图所示的按钮停止调试（只是断开调试连接，服务不会被停止）


Debugger
Console
Connected to the target VM, address: '192.168.4.38:2223', transport: 'socket'
2: Favorites
7: Structure