工作相关

启动事件系统

```
go run main.go --logfile /var/log/eventti/eventti.log --loglevel INFO --
listen-jrpc 0.0.0.0:8182 --listen-api 0.0.0.0:8184 --db_username eventti --
db_password r00tme --db_name eventti --db_addr 192.168.1.222:5432 --es_addr
192.168.1.146:9200 --grpc_addr 0.0.0.0:8188 --gway_addr 192.168.1.146:8185
```

windows 查看端口占用和服务占用

根据端口查询pid

```
netstat  -ano | findstr "8185"
```

根据pid查询服务

```
tasklist  | findstr   "16436"
```

google protobuf package import遇到的坑

在proto文件里，每个文件要定义**package**，一个包里的massage或者service要应用其他包下的massage时，需要将相应的包导入，这点类似与Golang语言，于此同时也面临了与Golang相同的问题，也就是包循环引用的问题。

需要将b的rpc service暴露出去，但是a 引用 b，b引用a ，这样是不能通过proto的编译的。

解决办法：引入第三方c c引用a ,c引用b 将c暴露出去

说实话 golang 的grpc服务端实现要比java服务端的优雅。

ip地址与long类型地址相互转换

```
  public static long ip2LongValue(String ip) {
        String[] split = ip.split("\\.");
        long l0 = Long.parseLong(split[0]);
        long l1 = Long.parseLong(split[1]);
        long l2 = Long.parseLong(split[2]);
        long l3 = Long.parseLong(split[3]);
        return l0 << 24 | l1 << 16 | l2 << 8 | l3;
    }
```

```
    public static String long2IP(long longIp) {
        StringBuffer sb = new StringBuffer("");
        // 直接右移24位
        sb.append(String.valueOf((longIp >>> 24)));
        sb.append(".");
        // 将高8位置0，然后右移16位
        sb.append(String.valueOf((longIp & 0x00FFFFFF) >>> 16));
        sb.append(".");
        // 将高16位置0，然后右移8位
        sb.append(String.valueOf((longIp & 0x0000FFFF) >>> 8));
        sb.append(".");
        // 将高24位置0
        sb.append(String.valueOf((longIp & 0x000000FF)));
        return sb.toString();
    }
```

java 通过位运算操作实现十进制转二进制

> 我们可以直接利用移位操作对一个十进制数进行移位操作，即：将最高位的数移至最低位（移31位），除过最低位其余位置清零，使用& 操作，可以使用和1相与（&），由于1在内存中除过最低位是1，其余31位都是零，然后把这个数按十进制输出；再移次高位，做相同的操作，直到最后一位，代码如下。可以说，这是我到目前为止见到的最简单的实现方式了。

```
public void binaryToDecimal(int n){
     for(int i = 31;i >= 0; i--) {
         System.out.print(n >>> i & 1);
     }
}
```

判断一个ip是否属于某个网段

```
ip a.b.c.d        cidr  e.d.g.a x.x.x.x

a.b.c.d 转成long  a
e.d.g.a 转成long c
mask 转成long  b

if(a&b ==c){
  //属于
}else{
//不属于
}
```

正掩码与反掩码相互转换

```
//正掩码
a.b.c.d  //long a
//反掩码
e.d.g.t    //long b
//255.255.255.255   long c


a ^ c   return b
b ^ c   return a
```

网络和IP地址计算

C网

192.168.1.*	网络
子网掩码是	255.255.255.0
子网数是	1
可用的主机数是	254
最大可容纳的主机数是	256
网络地址是	192.168.1.0
广播地址是	192.168.1.255
可用的IP是	192.168.1.1-192.168.1.254

通过 IP地址和子网掩码的运算得出网络地址。

以下面例子IP地址为192·168·100·5 子网掩码是255·255·255·0。计算出网络地址。

1、将IP地址和子网掩码换算为二进制，子网掩码连续全1的是网络地址，后面的是主机地址。 虚线前为网络地址，虚线后为主机地址。

192 • 168 • 100 • 5	11000000 • 10101000 • 01100100 • 00000101
255 • 255 • 255 • 0	11111111 • 11111111 • 11111111 • 00000000

2、IP地址和子网掩码进行与运算，结果是网络地址 。

	192 • 168 • 100 • 5	11000000 • 10101000 • 01100100 • 00000101
	255 • 255 • 255 • 0	11111111 • 11111111 • 11111111 • 00000000
与运算		
结果为:	192 • 168 • 100 • 0	11000000 • 10101000 • 01100100 • 00000000

3、主机号：掩码2进制反取值和IP相与 ，得到主机号

4、广播地址减1得到最后一个可用地址

192.168.1.53/27 的相应掩码是

11111111.11111111.11111111.11100000

由于网络号数量不足，所以得向右的主机数借位，主机数的可用数量减少。

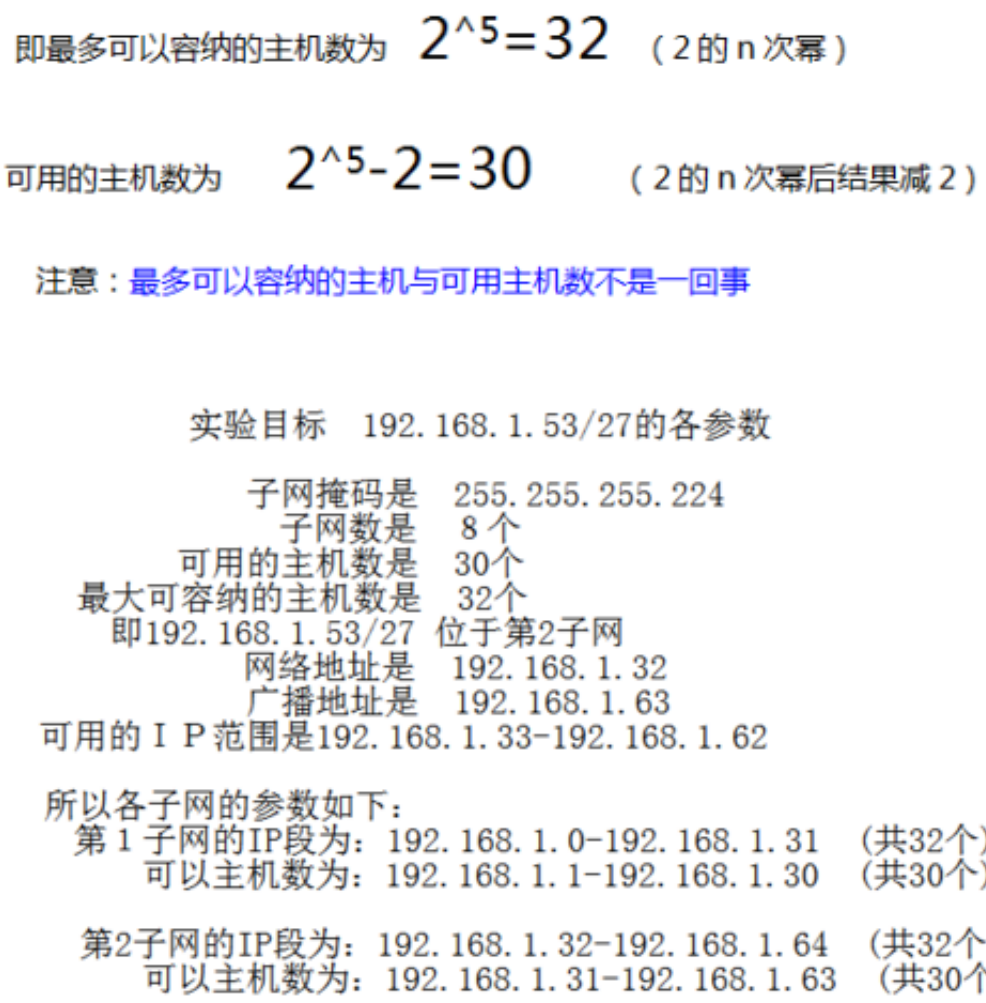
所有可用的主机数为主机号所剩下的5位掩码中算出。

注：8位主机号减少网络号借去的3位，所剩下5

即最多可以容纳的主机数为 32

可用的主机数为 30

计算公式如下图所示：



获取系统UUD

Linux: dmidecode -s system-uuid

windows: 在命令提示符下输入wmic 再输入csproduct 或 csproduct list full

maven项目转gradle项目

在pom.xml目录进入cmd 输入命令

```
gradle init --type pom
```

Intellij IDEA运行报Command line is too long

修改项目下 .idea\workspace.xml，找到标签 `<component name="PropertiesComponent">`， 在标签里加一行 `<property name="dynamic.classpath" value="true" />`

quartz问题记录-missed their scheduled fire-time

这里有3个原因：

1. 所有的woker thread(工作线程; 辅助线程)都在运行其他的job
2. scheduler(调度器)down了（关于这个down。我不太明确是shutdown了。。还是挂掉了。因此下文依旧用down。）
3. 任务被安排在过去的某一时刻启动(此可能为代码错误)

解决方法：修改quartz.properties文件中的org.quartz.threadPool.threadCount的值增大。（从原来的10增大到20），线程不够用，至此quartz挂掉。

VMware Workstation Pro 虚拟机打不开

> VMware Workstation 无法连接到虚拟机。请确保您有权运行该程序、访问该程序使用的所有目录以及访问所有临时文件目录。

1. 进入服务 services.msc
2. 找到VMware Authorization Service这项服务
3. 右键-属性 将启动类型选择"自动(延迟启动)"，然后应用，确定
4. 可以重启电脑或者关机重新开机

注:每次虚拟机不用的时候要 挂起！！！ 不要关机！！！

centos 系统时间不一致

如果时间不正确

    cp /usr/share/zoneinfo/Asia/Shanghai /etc/localtime   修改时区

时区正确，时间不正确

 使用yum 安装ntpdate

yum install -y ntpdate

ntpdate us.pool.ntp.org

Linux 设置静态ip

```
cd /etc/sysconfig/network-scripts
vim ifcfg-enp0s3
将其改成static
systemctl restart network
```

linux添加dns

```
vim /etc/resolv.conf

nameserver 192.168.1.1
```

Linux查看物理cpu个数

```
cat /proc/cpuinfo| grep "physical id"| sort| uniq| wc -l
```

查看每个物理cpu的核数

```
cat /proc/cpuinfo| grep "cpu cores"| uniq
```

VIM 批量替换

abc 批量替换成 def

```
:%s#abc#def#g
```

grep遍历文件夹查找文本内容

```
grep -r "要查找的内容" ./
```

linux 查看所有的环境变量

```
env
```

linux 和 macos 命令行光标移动首尾

首

```
Control + a
```

尾

```
Control + e
```

sed 替换文件内容

-i 本来的意思就原地替换的意思, 为啥不能替换呢? 一般情况 原地替换是比较危险的行为, 为了规避这样的行为, 需要你提供一个 备份的扩展名, 才能进行替换. 如果你给了空的扩展,就不会备份源文件. 但这不是推荐的方式. 因为 你面临这损坏或者部分损坏的危险 恰巧当磁盘空间用完的时候.

```
sed -i "" "s/.*/${currTime}/g" /Users/fangcong/source/study-log/time.log
```

java interface类型不能进行序列化和反序列化

当使用interface作为一个类属性时，将这个类存储到数据库时会报错 因为springboot使用的jackson 序列化不了接口，因为无法确定使用的是哪个实现类。

解决办法: 使用范型来解决

```
public class FieldSchema implements Serializable {

    private String property_id;
    private String property_name;
    private Option option;

}
```

```
public class FieldSchema<T> implements Serializable {

    private String property_id;
    private String property_name;
    private T option;

}
```

这样是可以序列化和反序列化的，此时jackson会使用 LinkedHashMap来序列化和反序列化。

直接使用 HikariCP 数据源的连接不会释放?

> HikariPool-1 - Connection is not available, request timed out after 30003ms.

默认连接池的连接数为10,当我直接使用datasource.getConnection()去执行sql，10次之后，就会报这个错。

解决方案：更换获取连接的方法，使用spring提供的DataSourceUtils.getConnection()后不再出现。 X

后续解决方案：使用上面的方案后还是有问题，维护了一个ThreadLocal 一个线程内共用一个链接，并且加大了maxPoolSize 暂时没有复现了。

备选方案：stackoverflow上有人说需要用完关闭connection,未验证。

Fastjson 重复引用对象的问题

出现相同的对象时，fastjson默认开启引用检测将相同的对象写成引用的形式

引用是通过"$ref"来表示的

引用	描述
"$ref":".."	上一级
"$ref":"@"	当前对象，也就是自引用
"$ref":"$"	根对象
"$ref":"$.children.0"	基于路径的引用，相当于 root.getChildren().get(0)

```
    public static void main(String[] args) {
        final JSONObject src = new JSONObject();
        src.put("a", "b");
        src.put("c", "d");
        src.put("e", "f");
        final JSONObject object = new JSONObject();
        object.put("src", src);
        object.put("target", src);

        System.out.println(object.toJSONString());
    }
```

输出:

```
{"src":{"a":"b","c":"d","e":"f"},"target":{"$ref":"$.src"}}

Process finished with exit code 0
```

解决办法:

使用 `DisableCircularReferenceDetect` 来禁止循环引用检测

```
//在可以添加SerializerFeature参数的地方添加此配置项即可
System.out.println(object.toString(SerializerFeature.DisableCircularReferenceDetect));
```

输出:

```
{"src":{"a":"b","c":"d","e":"f"},"target":{"a":"b","c":"d","e":"f"}}

Process finished with exit code 0
```