Test

Test

- □□ 1
  - □□□ 1
  - PAGE2
  - Nginx
  - □□□□

□□ 1

□□

□□□ 1

page contents

PAGE2

> PAGES2

```
main() {
  }
```

章节 1

Nginx

Nginx常用功能

1、代理

作为web服务器最常用的功能之一，尤其是反向代理。这里我给来2张图，对正向代理与反向代理做个诠释，具体细节，大家可以翻阅下资料。

image-20211021081901761

Nginx在做反向代理时，提供性能稳定，并且能够提供配置灵活的转发功能。Nginx可以根据不同的正则匹配，采取不同的转发策略，比如图片文件结尾的走文件服务器，动态页面走web服务器，只要你正则写的没问题，又有相对应的服务器解决方案，你就可以随心所欲的玩。并且Nginx对返回结果进行错误页跳转，异常判断等。如果被分发的服务器存在异常，他可以将请求重新转发给另外一台服务器，然后自动去除异常服务器。

2、负载均衡

Nginx提供的负载均衡策略有2种：内置策略和扩展策略。内置策略为轮询，加权轮询，Ip hash。扩展策略，就天马行空，只有你想不到的没有他做不到的啦，你可以参照所有的负载均衡算法，给他一一找出来做下实现。上3个图，理解这三种负载均衡算法的实现。

image-20211021082137339

Ip hash算法，对客户端请求的ip进行hash操作，然后根据hash结果将同一个客户端ip的请求分发给同一台服务器进行处理，可以解决session不共享的问题。

image-20211021082157521

3、web缓存

Nginx可以对不同的文件做不同的缓存处理，配置灵活，并且支持FastCGI_Cache，主要用于对FastCGI的动态程序进行缓存。配合着第三方的ngx_cache_purge，对制定的URL缓存内容可以的进行增删管理。

4、常见配置

1.$remote_addr 与 $http_x_forwarded_for 用以记录客户端的 ip地址;

2.$remote_user：用来记录客户端用户名称;

3.$time_local： 用来记录访问时间与时区;

4.$request： 用来记录请求的 url与 http协议;

5.$status： 用来记录请求状态; 成功是 200;

6.$body_bytes_s ent：记录发送给客户端文件主体内容大小;

7.$http_referer：用来记录从那个页面链接访问过来的;

8.$http_user_agent：记录客户端浏览器的相关信息;

章节 1

算法总结

判断闰年

```
static boolean is(int x) {
    return x%400==0 || (x%4==0 && x%100!=0);
}
```

筛素数

```
static boolean is(int n) {
    if(n==1)
        return false;
    for(int i=2;i*i<=Math.sqrt(n);i++)
        if(n%i==0)
            return false;
    return true;
}
```

倍筛法

```
public class 倍筛模板        {


    public static void main(String[] args) {
        boolean[] is = new boolean[1000005];
        is[1] = true;//true是不是素数
        for(int i=2;i<1000005;i++)
            if(!is[2])
                for(int j=2*i;j<1000005;j+=i)
                    is[j] = true;
        System.out.println(is[2004]);
    }
```

```
}
```

01背包

```
public class _01背包    {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int m = in.nextInt();//
        int n = in.nextInt();//物品数
        int[] dp = new int[m+5];//开背包质量的大小
        for(int i=1;i<=n;i++) {
            int v = in.nextInt();//积
            int w = in.nextInt();//物品价值
            for(int j=m;j>=v;j--)
                dp[j] = Math.max(dp[j], dp[j-v]+w);
        }
        System.out.println(dp[m]);
    }

}
```

完全背包

```
import java.util.Scanner;

public class _完全背包        {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int m = in.nextInt();//
        int n = in.nextInt();//物品数
        int[] dp = new int[m+5];//开背包质量的大小
        for(int i=1;i<=n;i++) {
            int v = in.nextInt();//积
            int w = in.nextInt();//物品价值
```

```
for(int j=v;j<=m;j++)
    dp[j] = Math.max(dp[j], dp[j-v]+w);
  }
  System.out.println(dp[m]);
 }


}
```

最小公倍数 gcd

```
 static int gcd(int a,int b) {
  return b==0?a:gcd(b,a%b);
 }
```

最大公约数 lcm

```
 static int lcm(int a,int b) {
  return a*b/gcd(a,b);
 }
```

经典二分法

```
 static void f(int[] a) {
  int n = a.length;
  int l=0,r=n-1,ans=0;//需要自己构造
  while(l<=r) {
   int mid = l + (r-l)/2;
   if(ok(a[mid])) {
    r = mid-1;
    ans = mid;
   }else
    l = mid+1;
  }
  System.out.println(ans);
 }
```

```
static boolean ok(int x) {
    return false;//题意写
}
```

前缀和

```
import java.util.Scanner;

public class 前缀和模板 {

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int n = in.nextInt();
        int[] sum = new int[n+1];
        for(int i=1;i<=n;i++)
            sum[i] = sum[i-1] + in.nextInt();//个数的和

    }

}
```

亲戚问题 （并查集 ）

```
static int n,m,cnt=0;
static int[] f = new int[10005];//初始化一般是 f[i]=i

static int find(int x) {
    if(f[x]==x)
        return x;
    return f[x] = find(f[x]);
}

static void union(int x,int y) {
    int a = find(x);
    int b = find(y);
    if(a!=b) {
        f[a] = b;
```

```
        cnt++;
    }
}
```

大数类 BigInteger和 BigDecimal

```
import java.math.BigInteger;
public class Main {
    public static void main(String[] args) {
        //加法
        BigInteger add1 = new BigInteger("10");
        System.out.println(add1.add(new BigInteger("20")));

        //减法
        BigInteger sub1 = new BigInteger("10");
        System.out.println(sub1.subtract(new BigInteger("20")));

        //除法
        BigInteger div1 = new BigInteger("10");
        System.out.println(div1.divide(new BigInteger("20")));

        //乘法
        BigInteger mul1 = new BigInteger("10");
        System.out.println(mul1.multiply(new BigInteger("20")));

        //取模
        BigInteger remain1 = new BigInteger("10");
        System.out.println(remain1.remainder(new BigInteger("8")));

        //快速幂取模
        BigInteger mod = new BigInteger ("10");
        BigInteger pow = new BigInteger ("20");
        System.out.println(pow.modPow(pow,mod));

        //比较两个数字的大小          ,小于返回    -1,大于返回    1,等于  0的时候返回      0
        BigInteger comp1 = new BigInteger("10");
        System.out.println(comp1.compareTo(new BigInteger("18")));

        //n次幂的运算
```

```
BigInteger power1 = new BigInteger("2");
System.out.println(power1.pow(10));

//返回较小的数
BigInteger min1 = new BigInteger("2");
System.out.println(min1.min(new BigInteger("-23")));

//返回较大的数
BigInteger max1 = new BigInteger("2");
System.out.println(max1.max(new BigInteger("-23")));

//返回该类型的数字的值
BigInteger val = new BigInteger("123");
System.out.println(val.intValue());

//返回最大公约数
BigInteger gcd1 = new BigInteger("12");
System.out.println(gcd1.gcd(new BigInteger("6")));

//取反
BigInteger neg1 = new BigInteger("12");
System.out.println(neg1.negate());

//按位与
BigInteger and1 = new BigInteger("10");
System.out.println(and1.and(new BigInteger("1")));

//按位或
BigInteger or1 = new BigInteger("10");
System.out.println(or1.or(new BigInteger("10")));

//异或
BigInteger xor1 = new BigInteger("10");
System.out.println(xor1.xor(new BigInteger("10")));

//返回 n进制的字符串 (进制的转换 )
BigInteger decimal1 = new BigInteger("12");
System.out.println(decimal1.toString(2));

//返回数字的绝对值
```

```
        BigInteger abs1 = new BigInteger("-12");
        System.out.println(abs1.abs());

        //检测某位上是否为              1
        BigInteger testBit1 = new BigInteger("4");
        System.out.println(testBit1.testBit(2));

       //左移   1位
        BigInteger moveLeftBit1 = new BigInteger("4");
        System.out.println(moveLeftBit1.shiftLeft(1));

        //右移   1位
        BigInteger moveRightBit1 = new BigInteger("4");
        System.out.println(moveRightBit1.shiftLeft(-1));

        //非
        BigInteger not = new BigInteger ("10");
        System.out.println(not.not());

        //valueOf()初始化

        //返回当前大整数的相反数
        BigInteger negate = new BigInteger ("10");
        System.out.println(negate.negate());

        //判断质数，并返回
        BigInteger prime = new BigInteger ("10");
        System.out.println(prime.probablePrime());
        System.out.println(prime.nextprobablePrime());
    }
}
```

```
package com.vivo.ars.util;
import java.math.BigDecimal;

/**
 * 用于高精确处理常用的数学运算
 */
public class ArithmeticUtils {
    //默认除法运算精度
    private static final int DEF_DIV_SCALE = 10;
```

```
/**
 * 提供精确的加法运算
 *
 * @param v1 被加数
 * @param v2 加数
 * @return 两个参数的和
 */

public static double add(double v1, double v2) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.add(b2).doubleValue();
}

/**
 * 提供精确的加法运算
 *
 * @param v1 被加数
 * @param v2 加数
 * @return 两个参数的和
 */
public static BigDecimal add(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.add(b2);
}

/**
 * 提供精确的加法运算
 *
 * @param v1    被加数
 * @param v2    加数
 * @param scale 保留  scale 位小数
 * @return 两个参数的和
 */
public static String add(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
    }
```

```
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.add(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
}

/**
 * 提供精确的减法运算
 *
 * @param v1 被减数
 * @param v2 减数
 * @return 两个参数的差
 */
public static double sub(double v1, double v2) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.subtract(b2).doubleValue();
}

/**
 * 提供精确的减法运算。
 *
 * @param v1 被减数
 * @param v2 减数
 * @return 两个参数的差
 */
public static BigDecimal sub(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.subtract(b2);
}

/**
 * 提供精确的减法运算
 *
 * @param v1    被减数
 * @param v2    减数
 * @param scale 保留  scale 位小数
 * @return 两个参数的差
 */
public static String sub(String v1, String v2, int scale) {
    if (scale < 0) {
```

```
        throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.subtract(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
}

/**
 * 提供精确的乘法运算
 *
 * @param v1 被乘数
 * @param v2 乘数
 * @return 两个参数的积
 */
public static double mul(double v1, double v2) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.multiply(b2).doubleValue();
}

/**
 * 提供精确的乘法运算
 *
 * @param v1 被乘数
 * @param v2 乘数
 * @return 两个参数的积
 */
public static BigDecimal mul(String v1, String v2) {
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.multiply(b2);
}

/**
 * 提供精确的乘法运算
 *
 * @param v1    被乘数
 * @param v2    乘数
 * @param scale 保留  scale 位小数
 * @return 两个参数的积
```

```
 */
public static double mul(double v1, double v2, int scale) {
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return round(b1.multiply(b2).doubleValue(), scale);
}

/**
 * 提供精确的乘法运算
 *
 * @param v1     被乘数
 * @param v2     乘数
 * @param scale 保留    scale 位小数
 * @return 两个参数的积
 */
public static String mul(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.multiply(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
}

/**
 * 提供（相对）精确的除法运算，当发生除不尽的情况时，精确到
 * 小数点以后          10位，以后的数字四舍五入
 *
 * @param v1 被除数
 * @param v2 除数
 * @return 两个参数的商
 */

public static double div(double v1, double v2) {
    return div(v1, v2, DEF_DIV_SCALE);
}

/**
 * 提供（相对）精确的除法运算。当发生除不尽的情况时，由                                    scale参数指
 * 定精度，以后的数字四舍五入
```

```
 *
 * @param v1    被除数
 * @param v2    除数
 * @param scale 表示表示需要精确到小数点以后几位。
 * @return 两个参数的商
 */
public static double div(double v1, double v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException("The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(Double.toString(v1));
    BigDecimal b2 = new BigDecimal(Double.toString(v2));
    return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).doubleValue();
}

/**
 * 提供（相对）精确的除法运算。当发生除不尽的情况时，由                scale参数指
 * 定精度，以后的数字四舍五入
 *
 * @param v1    被除数
 * @param v2    除数
 * @param scale 表示需要精确到小数点以后几位
 * @return 两个参数的商
 */
public static String div(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException("The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v1);
    return b1.divide(b2, scale, BigDecimal.ROUND_HALF_UP).toString();
}

/**
 * 提供精确的小数位四舍五入处理
 *
 * @param v     需要四舍五入的数字
 * @param scale 小数点后保留几位
 * @return 四舍五入后的结果
 */
public static double round(double v, int scale) {
```

```
    if (scale < 0) {
        throw new IllegalArgumentException("The scale must be a positive integer or zero");
    }
    BigDecimal b = new BigDecimal(Double.toString(v));
    return b.setScale(scale, BigDecimal.ROUND_HALF_UP).doubleValue();
}

/**
 * 提供精确的小数位四舍五入处理
 *
 * @param v      需要四舍五入的数字
 * @param scale 小数点后保留几位
 * @return 四舍五入后的结果
 */
public static String round(String v, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
    }
    BigDecimal b = new BigDecimal(v);
    return b.setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
}

/**
 * 取余数
 *
 * @param v1    被除数
 * @param v2    除数
 * @param scale 小数点后保留几位
 * @return 余数
 */
public static String remainder(String v1, String v2, int scale) {
    if (scale < 0) {
        throw new IllegalArgumentException(
                "The scale must be a positive integer or zero");
    }
    BigDecimal b1 = new BigDecimal(v1);
    BigDecimal b2 = new BigDecimal(v2);
    return b1.remainder(b2).setScale(scale, BigDecimal.ROUND_HALF_UP).toString();
}
```

```
    /**
     * 取余数       BigDecimal
     *
     * @param v1    被除数
     * @param v2    除数
     * @param scale 小数点后保留几位
     * @return 余数
     */
    public static BigDecimal remainder(BigDecimal v1, BigDecimal v2, int scale) {
        if (scale < 0) {
            throw new IllegalArgumentException(
                    "The scale must be a positive integer or zero");
        }
        return v1.remainder(v2).setScale(scale, BigDecimal.ROUND_HALF_UP);
    }

    /**
     * 比较大小
     *
     * @param v1 被比较数
     * @param v2 比较数
     * @return 如果  v1 大于  v2 则 返回  true 否则  false
     */
    public static boolean compare(String v1, String v2) {
        BigDecimal b1 = new BigDecimal(v1);
        BigDecimal b2 = new BigDecimal(v2);
        int bj = b1.compareTo(b2);
        boolean res;
        if (bj > 0)
            res = true;
        else
            res = false;
        return res;
    }
}
```