

02

OPEN ORIENTED

凹凸实验室

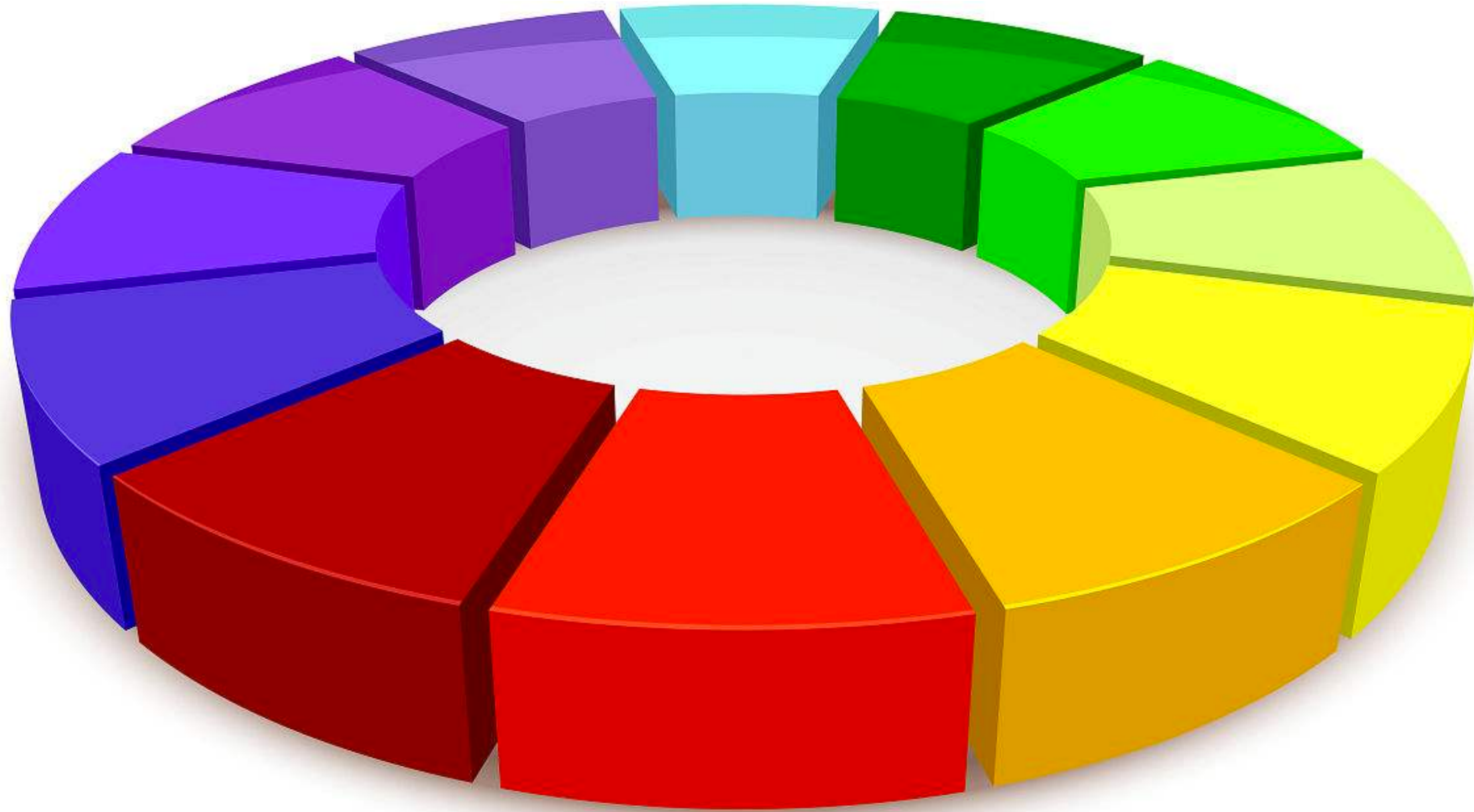
聊聊随机分配

波动均分算法

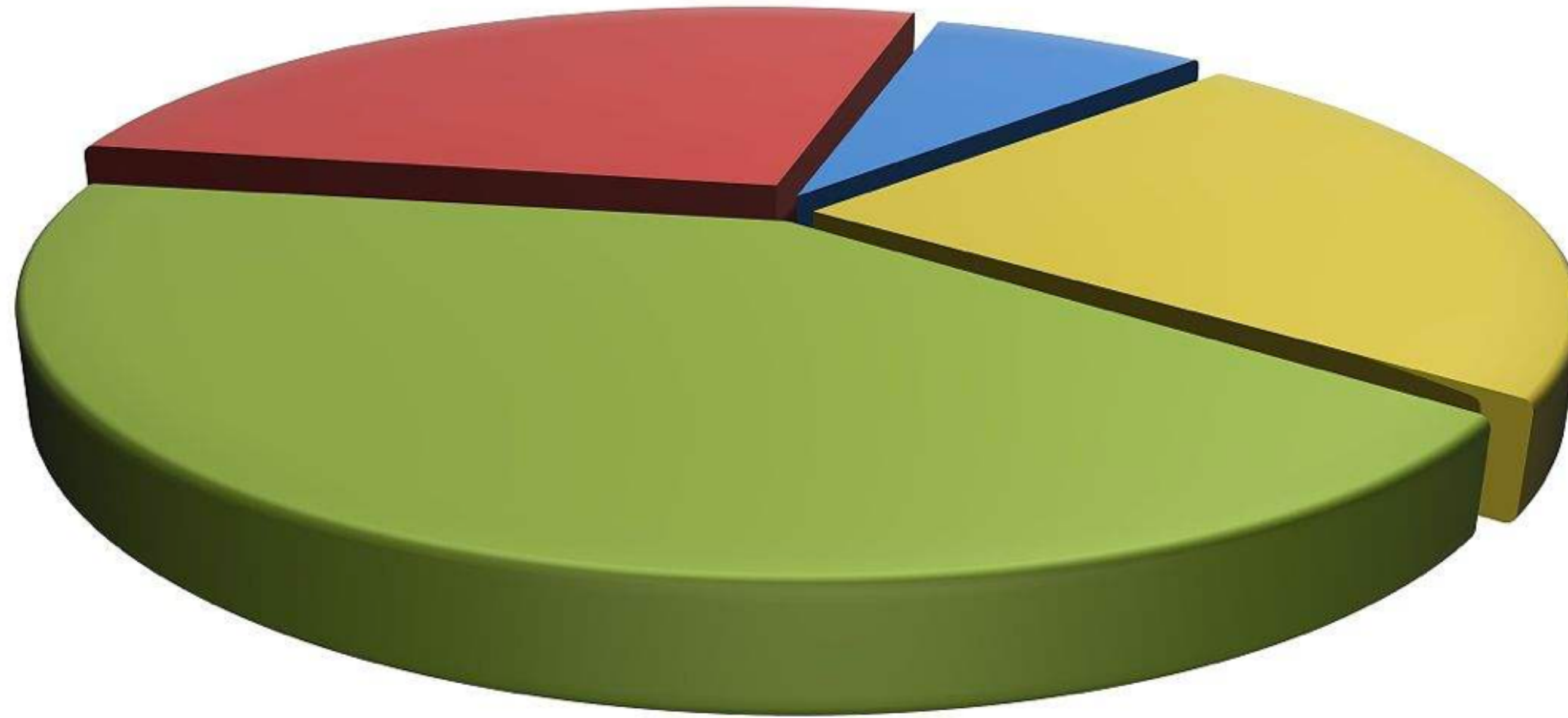
常见的分配算法

常见的分配算法

- 平均分配
- 不平均（随机）分配



$$\text{average} = \text{total} / n$$



没有具体的算法

不公平分配的应用场景

不平均分配的应用场景




```
public static double getRandomMoney(LeftMoneyPackage _leftMoneyPackage) {  
    // remainSize 剩余的红包数量  
    // remainMoney 剩余的钱  
    if (_leftMoneyPackage.remainSize == 1) {  
        _leftMoneyPackage.remainSize--;  
        return (double) Math.round(_leftMoneyPackage.remainMoney * 100) / 100;  
    }  
    Random r = new Random();  
    double min = 0.01; //  
    double max = _leftMoneyPackage.remainMoney / _leftMoneyPackage.remainSize * 2;  
    double money = r.nextDouble() * max;  
    money = money < min ? 0.01 : money;  
    money = Math.floor(money * 100) / 100;  
    _leftMoneyPackage.remainSize--;  
    _leftMoneyPackage.remainMoney -= money;  
    return money;  
}
```



```
public static double getRandomMoney(LeftMoneyPackage _leftMoneyPackage) {  
    // remainSize 剩余的红包数量  
    // remainMoney 剩余的钱  
    if (_leftMoneyPackage.remainSize == 1) {  
        _leftMoneyPackage.remainSize--;  
        return (double) Math.round(_leftMoneyPackage.remainMoney * 100) / 100;  
    }  
    Random r = new Random();  
    double min = 0.01; //  
    double max = _leftMoneyPackage.remainMoney / _leftMoneyPackage.remainSize * 2;  
    double money = r.nextDouble() * max;  
    money = money < min ? 0.01 : money;  
    money = Math.floor(money * 100) / 100;  
    _leftMoneyPackage.remainSize--;  
    _leftMoneyPackage.remainMoney -= money;  
    return money;  
}
```

据说微信红包的算法是这样的

```
public static double getRandomMoney(LeftMoneyPackage _leftMoneyPackage) {  
    // remainSize 剩余的红包数量  
    // remainMoney 剩余的钱  
    if (_leftMoneyPackage.remainSize == 1) {  
        _leftMoneyPackage.remainSize--;  
        return (double) Math.round(_leftMoneyPackage.remainMoney * 100) / 100;  
    }  
    Random r = new Random();
```

取0.01到剩余平均值*2之间作为红包的金额

```
    money = Math.floor(money * 100) / 100;  
    _leftMoneyPackage.remainSize--;  
    _leftMoneyPackage.remainMoney -= money;  
    return money;  
}
```

据说微信红包的算法是这样的

小明发了一个100元的随机红包给十个人抢，
理论上手气最佳者能得几元？

小明发了一个100元的随机红包给十个人抢，
理论上手气最佳者能得几元？

答案： 99.91元

如何把手气最佳者与最差者
的金额差限制在10元内？

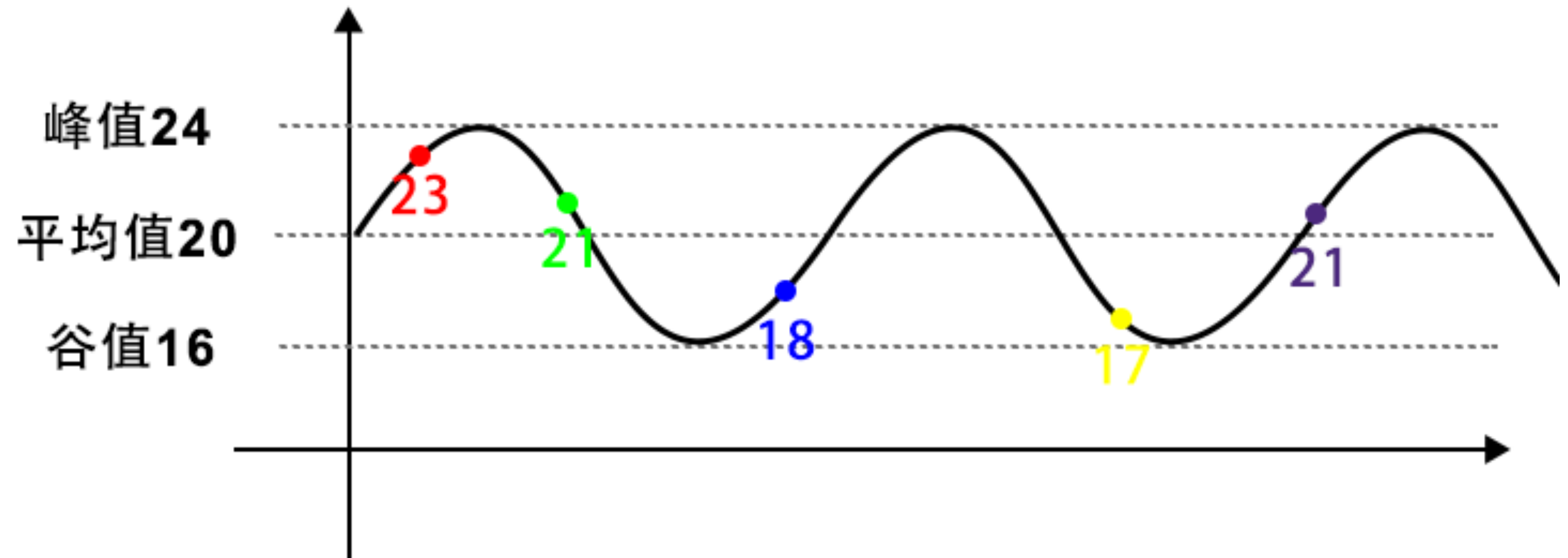
如何把手气最佳者与最差者
的金额差限制在10元内？

可以使用「波动均分」实现

1. 分配数量
2. 波峰高度
3. 波谷深度
4. 随机分配
5. 组合全面

波动均分介绍

1. 分配数量
2. 波峰高度
3. 波谷深度
4. 随机分配
5. 组合全面




- 穷举法
- 快速分配

- 穷举法
- 快速分配



用一棵 N 叉树表示所有可能的组合

- 穷举法
- 快速分配

- 
- 获取可分配波动范围;
 - 在波动范围内随机取值;

- 
- 获取可分配波动范围;
 - 在波动范围内随机取值;

每一次迭代都需要执行上述判断

```
1 function quickWave(n = 5, crest = 4, trough = 4, isInteger = true) {  
2     let list = [];  
3     // 无法进行波动均分, 直接返回完全平分  
4     if(crest > (n - 1) * trough || trough > (n - 1) * crest) {  
5         return new Array(n).fill(0);  
6     }  
7     let base = 0; // 最少需要消除的高度  
8     let wave = 0; // 波动量  
9     let high = crest; // 高位  
10    let low = -trough; // 低位  
11    let sum = 0; // 累计量  
12    let count = n; // 剩余数量  
13    while(--count >= 0) {  
14        // 动态当前的波动量  
15        if(crest > count * trough - sum) {  
16            high = count * trough - sum;  
17        }  
18        if(trough > count * crest + sum) {  
19            low = -sum - count * crest;  
20        }  
21        base = low;  
22        wave = high - low;  
23        let rnd; // 随机波动量  
24        if(count > 0) {  
25            rnd = base + Math.random() * (wave + 1); // 随机波动  
26        } else {  
27            rnd = -sum;  
28        }  
29        if(isInteger === true) {  
30            rnd = Math.floor(rnd);  
31        }  
32        sum += rnd;  
33        list.push(rnd);  
34    }  
35    return list;  
36 }
```

`"https://aotu.io/notes/2018/01/11/waveaverage/"`

–Auto Blog

THANKS
FOR YOUR WATCHING

