1、关系性数据库事务隔离级别有哪些？MSSQL的默认事务隔离级别是什么？

SQL四类事务隔离级别标准，低级别隔离一般支持更高的并发处理，系统也开销更低。

- **Read Uncommitted [读取未提交]** : 最低隔离级别，所以事务均能看到其它未提交事务结果，即读取未提交的数据。即出现脏读,很少用于实际产线环境。
- **Read Committed [读取已提交]**: 可以读取并发事务已经提交的数据，可阻止脏读，但可能产生不可重复读和幻读
- **Repeatable Read [可重复读]**:解决了脏读的问题，该级别保证了在同一次事务中多次查询相同的语句结果是一致的.
- **Serializable [可串行化]**: 最高的隔离级别，它通常通过强制事务串行，避免了前面说的幻读问题;会在读取的每一行数据上都加锁，所以可能会导致大量的锁等待和超时问题，所以在实际的生产环境中也很少会用到这个隔离级别.

MSSQL默认隔离级别为Read Committed.

2、索引的类型和作用？它的优缺点是什么？

类型:

> **唯一索引**: 主键创建后一定包含一个唯一索引,反之不一定.
> **聚集索引**: 把一个表中的特定列的数据都按照偏旁部首索引起来，更适合查询操作,一个表只能有一个聚集索引
> **非聚集索引**: 与聚合相反，更加适合有插入、删除、更新的操作

优势:

> 提高查询效率,减少数据检索时间
> 保证数据唯一性 减少分组\排序的检索时间

缺点:

> 创建索引和维护索引需要消耗时间 索引需要占一定的物理存储哦那关键 增删改等操作都需要动态维护索引

3、海量数据如何存储？

海量数据主要面临数据量太大查询慢和高并发情况下数据库实例能够支持住

1. 针对查询慢问题

> 建立缓存机制: 针对热门数据,可以缓存在redis中, 将大部分查询操作引流通过redis完成
> 合理分表存储: 如建立年份表,月份表等方式拆分大表,还可以将日志索引分区存储
> 合理使用临时表: 数据在增加同时,建立合理机制汇总整体数据,尽量不要一次查询所有数据.

2. 高并发情况

> 可合理增加数据库实例,一个数据库实例撑不住，就把并发请求分散到多个实例中去.

4、Redis 分布式锁是怎么实现的？

分布式锁用于控制分布式系统或者不同系统之间共同访问共享资源，防止彼此干扰保证一致性。

下面模拟电商行业扣减库存场景下代码实现，利用Redis SETNX命令的实现ts代码块(代码块为最近项目解决不同用户修改同样文件场景中参考)

```
deductStock(product:any)
{
    let requestId = new UUID();
    try {
        let result = redis.setnx(product.id, requestId);    //将商品ID置为key,
value可随机一个UID，用于释放属于自己的锁
        if(!result){    // 意味上锁失败
            return '系统繁忙，请重试';
        }
        redis.expire(product.id, 10); //设置lock失效时间,避免宕机死锁饥饿
        let stock = redis.get(product.id)
        if(stock > 0){
            let realStock = stock - 1;
            redis.set(product.id, realStock);
        }
        else{
            return '库存不足';
        }
    } finally (error) {   // 目的避免上锁后业务异常造成锁无法释放的情况
       if(requestId === redis.get(product.id)){ //释放属于自己的锁，避免误解其它进程
的锁
           redis.delete(product.id);
       }
    }
}
```

5、泛型，请用 C#写一个带泛型参数的方法体（要求：泛型参数必须为可以实例化的对象）

```
/// <summary>
/// Clone two data between tow different type.
/// </summary>
/// <typeparam name="TSource"></typeparam>
/// <typeparam name="TTarget"></typeparam>
/// <param name="source"></param>
/// <returns></returns>
public static TTarget Clone<TSource, TTarget>(TSource source)
    where TSource : class, new()
    where TTarget : class, new()
{
    var target = new TTarget();

    var sourceProperties = PropertyCache.GetPropertyInfoList(typeof (TSource));

    foreach (var sourceProperty in sourceProperties)
```

```
    {
        var targetProperty = PropertyCache.GetPropertyInfo(
            typeof (TTarget), sourceProperty.Name, sourceProperty.PropertyType);

        if (null == targetProperty) continue;

        var value = sourceProperty.GetValue(source, BindingFlags.GetProperty,
            null, null, CultureInfo.CurrentCulture);

        targetProperty.SetValue(target, value, null);
    }

    return target;
}
```

6、什么是 IOC、DI，有什么特点？

IOC[控制反转]是一致设计思想，目标是对程序的各组件提供更多的控制，让这些组件更好的完成工作。它把传统上有代码直接操控的对象的调用权交给容器，通过容器实现对对象组件的装配和管理。

DI[依赖注入]是对IOC更准确的描述，对象组件之间的依赖关系有容器再运行期决定，即由容器动态的将某种依赖注入到组件中。

之前基于Net微服务框架中自己实现对象实例化的容器,链接如下.
[https://gitee.com/Shaun_Lai/SDChinaMobile/blob/master/src/Framework/Business/InstanceManager.cs]

7、Winform 线程管理，如何判断多线程是否全部结束，以及在子线程中如何操作 UI？

`System.Environment.Exit(0)` 彻底退出所有线程

子线程操作UI,通过委托实现

```
Thread thread = new Thread(ThreadFuntion);
thread.IsBackground = true;
thread.Start();
```

8、设计一个秒杀活动，简单描述一下设计思路。

1.前端：

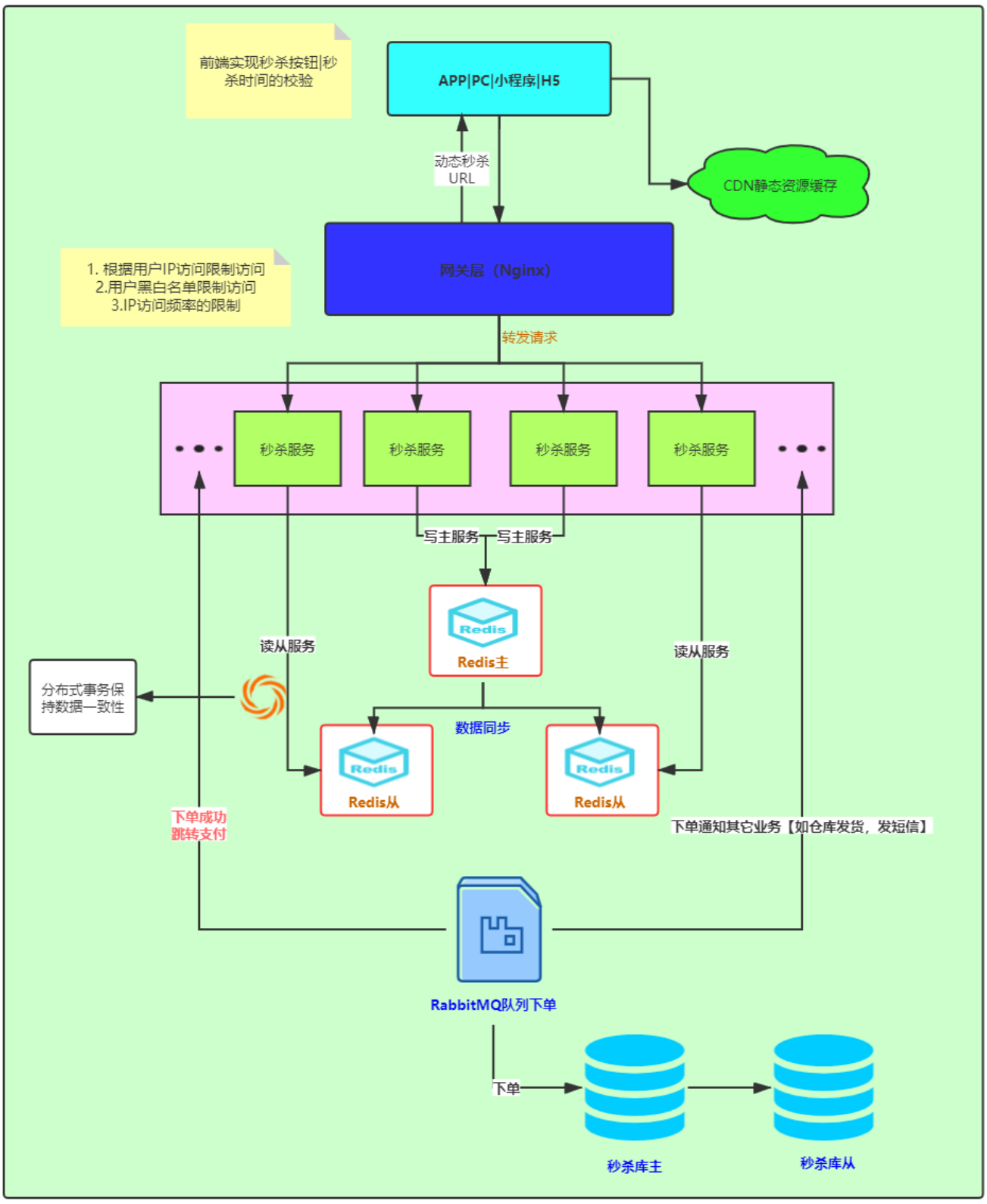
> a.静态资源静态化；将静态资源提前放入CDN服务器，减少秒杀时候服务器压力
> b.秒杀URL动态化，防止秒杀链接提前暴露
> c.前端限流；可以设置秒杀按钮秒杀开始才启用，不可连续点击等行为限制请求

2.后端

> a. 服务单一，秒杀服务即便崩了，也不能影响其它服务（订单），保证服务的高可用
> b. 库存预热， 秒杀本质是对库存的抢夺，可将秒杀前通过任务将商品库存加载到Redis中，让整个服务

> 流程再redis中完成，秒杀结束后再异步修改库存即可
> c. 超卖问题redis本身支持事务，可以利用LUA脚本完成库存扣减

[架构设计图] https://www.processon.com/view/link/5f9551d37d9c0806f28d7c23

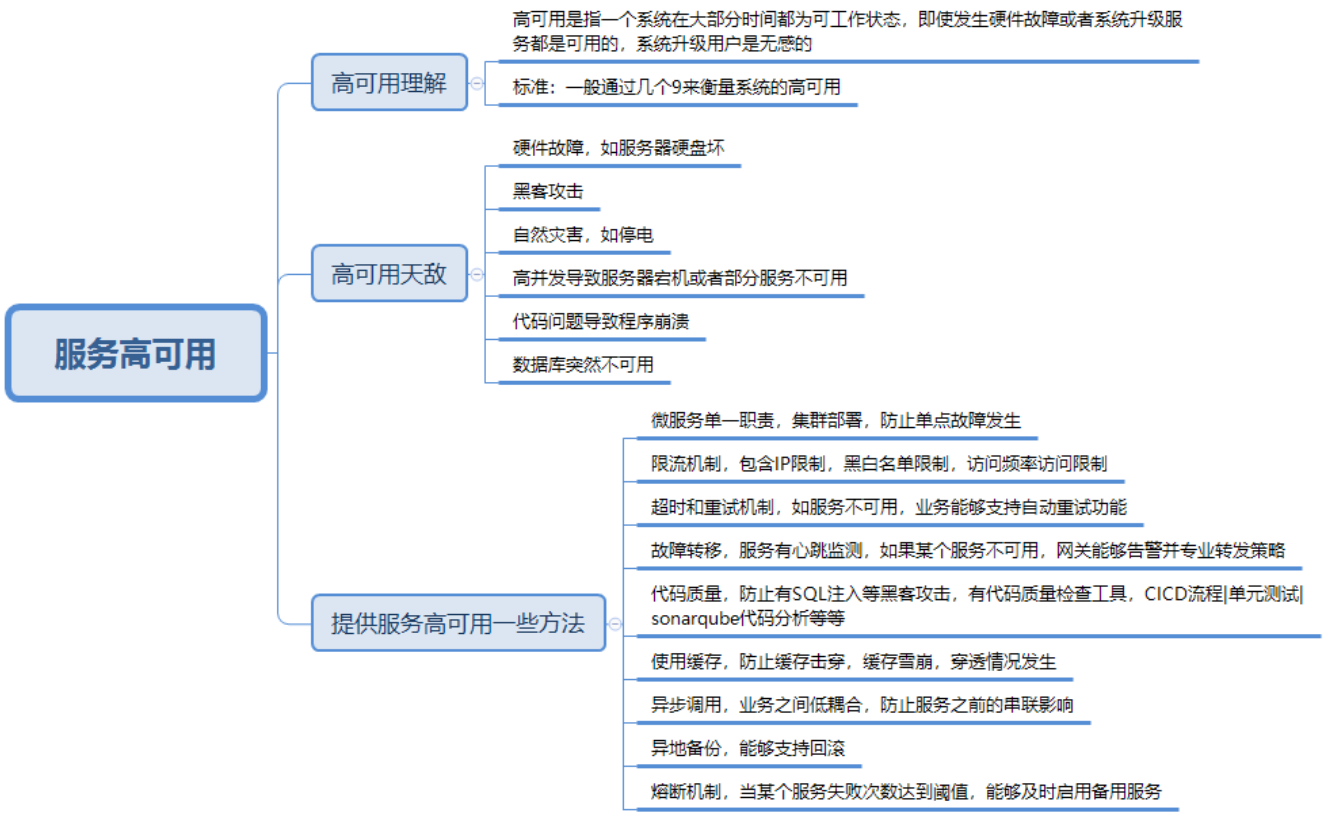


9、什么前后端分离，为什么要采取前后端分离？

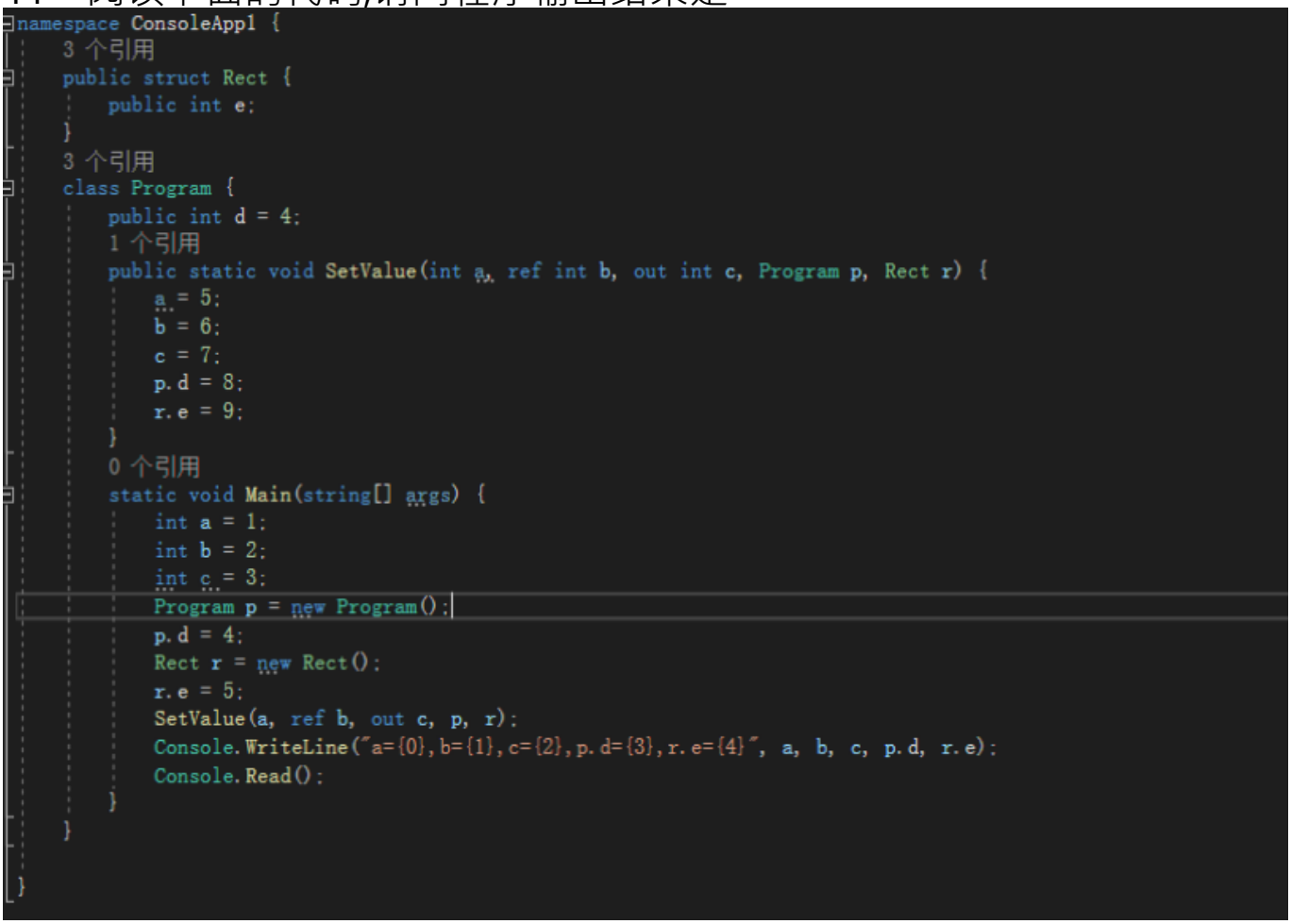
前后端分离是属于一种开发模式,开发过程中需要约定交互接口,实现并行开发,前端专注渲染和交互,后端专注业务开发,开发结束后需要集成测试,独立部署.

> 1.真正实现前后业务解耦,减少后台压力
> 2.系统出现问题,快速锁定问题属于谁,逻辑清晰
> 3.后端服务暂时不可用页面也能正常渲染,用户体验更佳
> 4.开发效率,双方可并行开发,代码,技术栈清晰
> 5.页面可以做到动态加载数据

10、如何确保服务的高可用性，有哪些解决方案？



11、阅读下面的代码,请问程序输出结果是



```
namespace ConsoleApp1 {
    3 个引用
    public struct Rect {
        public int e;
    }
    3 个引用
    class Program {
        public int d = 4;
        1 个引用
        public static void SetValue(int a, ref int b, out int c, Program p, Rect r) {
            a = 5;
            b = 6;
            c = 7;
            p.d = 8;
            r.e = 9;
        }
        0 个引用
        static void Main(string[] args) {
            int a = 1;
            int b = 2;
            int c = 3;
            Program p = new Program();
            p.d = 4;
            Rect r = new Rect();
            r.e = 5;
            SetValue(a, ref b, out c, p, r);
            Console.WriteLine("a={0},b={1},c={2},p.d={3},r.e={4}", a, b, c, p.d, r.e);
            Console.Read();
        }
    }

}
```

```
a=1,b=6,c=7,p.d=8,r.e=5
```

12、用 C#实现一个冒泡排序算法

```
public static void BubbleSort<T>(T[] array, Comparison<T>comparison)
{
    bool hasExchagend = false;
    for (int i = 0; i < array.Length - 1; i++) {
        for (int j = array.Length - 1; j > i; j--) {
            hasExchagend = false;
            if (comparison(array[j - 1], array[j]) > 0) {
                Exchange<T> (ref array[j], ref array[j - 1]);
                hasExchagend = true;
            }
        }

        if (!hasExchagend) {
            return;
        }

    }
```

```
}

private static void Exchange<T>(ref T x, ref T y)
{
    T temp = x;
    x = y;
    y = temp;
}
```

13、用 C#实现一个单例

Lazy来实现延迟加载

```
public sealed class Singleton
{
    private static readonly Lazy<Singleton> lazy = new Lazy<Singleton>(() => new
Singleton());

    public static Singleton Instance
    {
        get
        {
            return lazy.Value;
        }
    }

    private Singleton()
    {
    }
}
```

14

14、用户表（User）如下

ID	Name
1	小明
2	张三
3	李四
4	张三
5	王二
6	小刘

请写出删除 Name 重复记录的 SQL 语句（保留 ID 最小的）

> 避免使用NOT IN, 可通过LEFT JOIN优化
> 子查询不能使用ORDER BY

```
DELETE FROM dbo.[user] WHERE id IN (
SELECT
 U.id
```

```
FROM
 dbo.[user] AS U
LEFT JOIN(
 SELECT
  TOP 100 PERCENT MIN(id) AS minid
 FROM
  dbo.[user]
 GROUP BY
  "name"
 ORDER BY
  minid) AS D ON
 U.id = d.minid
WHERE
 D.minid IS NULL)
```

15、学员表（Student）如下

ID	Name
1	小明
2	张三
3	李四

课程表（Course）如下

ID	Title
1	语文
2	数学
3	英语

成绩（Scores）如下

ID	StudentID	CourseID	Score
1	1	1	80
2	1	2	76
3	1	3	99
4	2	1	105
5	2	2	88

分页查询出所有学生的各自成绩总分，并按成绩由高到低排列，要求显示学员 ID、姓名、成绩总分

15
分页查询出所有学生的各自成绩总分，并按成绩由高到低排列，要求显示学员 ID、姓名、成绩总分

```
SET STATISTICS TIME ON;

SELECT TOP 2    --pageSize
    tmp.id
    ,tmp.name
    ,tmp.total
FROM (
    SELECT TOP 100
        ROW_NUMBER() OVER(ORDER BY s.id asc) AS rownumber
        ,s.id
        ,MIN(s."name") as name
```

```
        ,(CASE WHEN SUM(sc.score) IS NULL THEN 0 ELSE SUM(sc.score) END) AS total
    FROM
        dbo.student AS s
    LEFT JOIN
        dbo.scores AS sc
    ON
        sc.studentid = s.id
    GROUP BY s.id
    ORDER BY total DESC
) AS tmp
WHERE tmp.rownumber > (1-1)*2 -- (pageIndex-1)*pageSize
```

```
SET STATISTICS TIME ON;

SELECT TOP 2    --pageSize
    tmp.id
    ,tmp.name
    ,tmp.total
FROM (
    SELECT TOP 100
        ROW_NUMBER() OVER(ORDER BY s.id asc) AS rownumber
        ,s.id
        ,MIN(s."name") as name
        ,(CASE WHEN SUM(sc.score) IS NULL THEN 0 ELSE SUM(sc.score) END) AS total
    FROM
        dbo.student AS s
    LEFT JOIN
        dbo.scores AS sc
    ON
        sc.studentid = s.id
    GROUP BY s.id
    ORDER BY total DESC
) AS tmp
WHERE tmp.rownumber > (1-1)*2 --(pageIndex-1)*pageSize
```

100 %

Results Messages

	id	name	total
1	1	小明	255
2	2	张三	193

16、有一列数 1，1，2，3，5，8，……，求第 50 个数。请用 C#实现过程

```
public class Fibona
{
    public ulong getFibonacci(long n)
    {
        if (n == 1 || n == 0)
        {
            return 1;
        }
```

```
            return getFibonacci(n - 1) + getFibonacci(n - 2);
        }
    }
```