学习新技术的方法一：

1. 从理论上下手，了解一下是什么东西。（知道）---概念
2. 从使用上下手，了解一下他的语法。（记住）--使用
3. 找出和他相似的技术点，从性能上 比较两者的实现的原理（理解）--性能、原理
4. 从应用场景上推演，加深对技术的理解（练习--举一反三）--应用场景
5. 生命周期 理解什么时候从哪儿开始，什么时候从哪结束

适应新框架的开始：

数据库连接字符串 增删改查 多表联查 文件上传 条件查询 导入导出 系统日 志记录 用户操作日志记录 当前用户信息 角色、权限、模块配置

列表 添加 修改 搜索框 刷新列表 返填 选项卡 配置样式（文本框等长宽高）

调用控制器的方法js 增删改查 添加按钮 非空验证

1.

问题：

VS2015提示：未安装Style的Visual Studio语言支持，代码编辑Intellisense将不可用。服务器控件的标记Intellisense可能不起作用

解决：

（工具-拓展更新）Microsoft ASP.NET and Web Tools

2.

问题：

Datatables（页码）不刷新

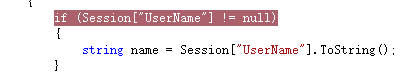
解决：

加上状态: "stateSave":false

1. mvc中的过滤器
2. 授权过滤器：限制进入控制器或控制器的某个行为方法

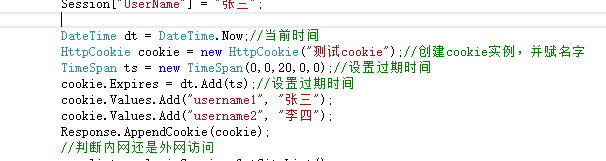
所有过滤器中最先执行IAuthorizationFilter

1. 动作过滤器：动作过滤器IActionFilter
2. 结果过滤器IResultFilter
3. 异常过滤器IExceptionFilter
4. 设置session 在服务端，session[“1423”]=”张三”





1. 设置cookie，创建httpcookie对象并赋名字，创建当前时间，创建timespan实例，赋值过期时间，然后将两者结合的过期时间设置在cookie的expires中，response appendcookie（cookie）添加在浏览器中



# 6.**HashTable, Dictionary, ArrayList、数组**的区别：

1.HashTable和Dictionary 前者在添加数据时，不需要声明类型。而后者需要声明其类型。并且，前者存入的数据都是objct的类型，取数据的时候，需要进行类型转换，对于值类型的数据，需要进行装箱和取消装箱。而后者，因为在声明时候，指定了类型，所以不需要进行类型转换。相比之下，dictionary性能高一些。

前者：数据类型不唯一 后者：体现顺序的情景中（remove之后，顺序被打乱）

1. ArrayList和数组 前者在添加数据的时候，不需要声明其类型。而后者需要声明其类型。并且，前者不需要手动指定数组的长度，会根据数据的变化自动改变数组的长度。后者需要指定数组的长度。

7.viewbag和viewdata的区别：前者是动态类型，viewbag.名称=值 后者是字典类型viewdata【“1320”】=值 的方式，两者只在当前Action中有效，等同于view，两者的值可以互相访问，因为Viewbag的实现中包含了viewdata

8.控制器中actionresult的返回值常用的有 view、content、json 还有file jsvascript css等

9.将实体数据转换成json的几种方式：

1.通过字符串拼接的方式，返回content

2.序列化为json字符串

10.

虚方法可以声明方法体 抽象类和接口不可以声明方法体

/// 抽象类和虚方法 是需要重写的 接口是需要又实现的

/// 抽象类和接口必须重写实现所有的成员 但是虚方法可以先不重写 先不实现

/// 重载是同一个类中 几个方法名相同 参数或参数类型不同的方法

/// 重写是在继承关系中，子类重新定义父类的方法 方法名和参数都相同

1. 静态方法和非静态方法的区别
   1. 使用上：静态方法直接点方法或者属性就好，不需要实例化。而且非静态方法需要实例化之后，才可以调用。
   2. 声明周期不同：
      1. 静态方法又叫类方法，所以静态方法会随着类的定义加载到内存中，一直到线程结束，静态方法才会被销毁。
      2. 非静态方法，在实例化对象之后，才会被创建，直到这个实例化的对象被销毁时，才被销毁。
2. File类和FileInfo类的区别
   1. 两者都可以对文件进行增删改查的操作，他们都是基于Syatem.IO命名空间下的操作类
      1. File 静态类，如果对文件进行单一的操作或者对很多文件进行操作，推荐使用file类，因为它不要实例化对象，可以直接使用。
      2. Fileinfo类，如果对同一文件进行多次操作，推荐使用fileinfo类
3. File类的每个方法都需要对路径进行检查，即使多次对同一个文件进行操作也需要进 行检查。Fileinfo不需要进行多次检查，只需要在创建的时候进行一次检查。所以，在对 同一个文件执行多次操作的时候，用fileinfo。 如果执行一些单一的方法，就用file。

13.

1. 总结思考为荣，浑水摸鱼为耻
2. 定义常量为荣，魔法数字为耻
3. 代码优化为容，复制粘贴为耻
4. 设计模式为荣，代码重复为耻
5. 参数校验为荣，运行异常为耻
6. 代码自测为荣，过度自信为耻
7. 规范日志为荣，乱打日志为耻
8. 接口兼容为荣，接口裸奔为耻

14.Func委托和Action委托的区别

1. Func委托有返回值
2. Action委托没有返回值
3. Out和Ref的区别
   1. 前者只出不进，后者有进有出
   2. 前者在方法内部完成初始化，后者在传入方法参数之前--对其完成初始化
   3. 两者都是按照地址传递的，从而改变参数的值
4. 里氏替换原则
   1. 所有使用基类的地方能够使用子类进行替换，而程序的行为不会发生任何变化（替换为子类之后不会产生错误或异常）
5. 隐式类型转换和显式类型转换
   1. 前者是 从小的类型转换成大的类型，从派生类转换到基类
   2. 后者是 强制类型转换，需要强制转换运算符
   3. 前者不会造成数据丢失，后者会造成数据丢失
6. git和svn的区别
   1. 前者是分布式的，后者不是
   2. 前者把内容按元数据方式存储，后者是按文件：所有的资源控制系统都是把文件的元信息隐藏在一个类似.svn的文件夹里
   3. Git没有一个全局的版本号，svn有
   4. Git的内容完整性要优于svn
7. js闭包
   * 1. Js嵌套函数中 返回子函数，子函数中要有对父函数中私有变量的操作 并把这个父函数赋值给一个变量 在调用这个变量的时候 父函数中的私有变量只声明命一次。
     2. 闭包就是一种保护私有变量的机制，在函数执行时形成私有变量的作用域，保护里面的私有变量不受外界的干扰。
     3. 就是实现了局部变量也能像全局变量一样使用（例子：累加）。
8. 前台线程和后台线程的区别
   1. 应用程序必须在运行完所有的前台线程后退出
   2. 应用程序可以在没有运行完后台线程的情况下直接退出，所有的后台线程在应用程序退出时，自动结束。
   3. 用Thread创建的线程都是前台线程，主线程也是前台线程，线程池的线程默认为后台线程（task创建的默认是线程池中的线程）
   4. IsBackground属性用来设置线程是前台还是后台，需要在线程启动之前运行
   5. 多线程的优点：
      1. 可以让计算机同时坐很多事情，节约时间。
      2. 后台运行程序，提高效率，不会使主界面出现假死或者无响应
9. c#中的? ?? ?. ?””:””
   1. ?可为空（一般在字段中写）
   2. ??如果左边的为空 则 返回右边的
   3. ?.如果左边的不为空 则 执行右边的
   4. ?””:”” 三目运算符
10. 扩展
    1. 给$添加方法:$.extend({ min:function(a,b){return a<b?a:b}, })
       1. 使用$.min(3,5) 返回3
    2. 给jQuery的方法:$.fn.extend({ checkAll:function(){ this.prop(‘checked’,true) }, })
       1. 使用$item.checkAll()
11. 协变与逆变
12. dictionary和hashtable的区别：
    1. Hashtable表示键值对的集合
    2. Dictionary表示键和值的集合
    3. hashtable不支持泛型，而dictionary支持泛型
    4. Hashtable元素属于object类型，所以在存储和检索值类型的时候会装箱和拆箱，需要进行类型转换，耗时间。
    5. 单线程推荐使用dictionary，支持泛型，声明已经定义好了类型，不需要进行装箱和拆箱，速度较快。
13. 事务TransactionScope scope = new TransactionScope（）
    1. 提交的时候 支持循环多次操作 然后统一提交（也就是说，循环执行多次操作的时候，只需要在最后提交一次即可）
14. 添加、修改、删除需要的验证
    1. 添加（验证唯一性，可不可以重复添加，提示已经添加）
    2. 修改（验证有没有删除，两天电脑的情况下，提示数据不存在）
    3. 真删除（验证有没有删除，提示已经删除）
15. 预处理器：预处理器指令指导编译器在实际编译开始之前对信息进行预处理。
    1. 所有的预处理器指令都是以#开始
    2. 且在一行上，只有空白字符可以出现在预处理器指令之前。
    3. 预处理器指令不是语句，所以他们不以分号结束
16. varchar和nvarchar的区别
    1. 两者的最大长度设置不一样，前者是1-8000。后者是1-4000
    2. 前者能存储的字节数就是他的长度 后者能存储的字节数是他的两倍。
    3. 存储方式上看，前者是按照字节存储的。后者是按照字符存储的
    4. 存储量来看，前者比较节省空间，因为存储的大小就是字节的实际长度，而后者是双字节存储
    5. 使用上，如果存储内容是英文字符，建议使用前者。如果是带有汉字的，建议使用后者，因为后者使用的Unicode编码，统一的编码，会减少乱码出现的几率。

简单的来说，一般有汉字的使用后者。没有汉字的使用前者。

1. String和stringbuilder的区别
   1. 1
2. 内存泄漏：（内存浪费）未能释放已经不在使用的内存（因为疏忽或者错误导致应用程序产生的情况）
3. 缓冲区溢出：缓冲的数据超出了缓冲区的容量，造成多余的部分溢出。
4. 复杂度
   1. 算法效率：
      1. 时间效率
      2. 空间效率
   2. 时间复杂度：一个算法所花费的时间与其中语句的执行次数成正比例，算法中的基本操作的执行次数，为算法的时间复杂度。（执行的次数，并且根据n的 大小也记录了增长的趋势--比如O（1）表示算法消耗的时间不随规模的增长而增长）
      1. 常用的时间复杂度大小比较：O(1)<O(logn)<O(n)<O(n的2次方)<O(2的n次方)<O（n的阶乘）
      2. 计算时间复杂度的方式，就是算出程序运行的基本操作的执行次数的最坏情况。
   3. 空间复杂度：表示算法需要消耗的最大数据空间。
      1. 是时间复杂度一样 也是用o（n）这样的式子来表示，大小比较也如上所示。
   4. 编码复杂度:是一个定性的概念，并没有什么量化的标准。对于一个问题来说，如果用了冗长的算法思想，那么代码量就会非常巨大，其编码复杂度就会非常大。
5. Unix时间戳：是从1970年1月1日 开始所经过的秒数，不考虑闰秒。
6. 乐观锁和悲观锁
   1. 乐观锁：乐观的认为别人不会同时操作数据，乐观锁默认是不会上锁的。
   2. 悲观锁：悲观的认为别人一定会同时修改数据，任何情况都会上锁。
   3. 乐观锁适合在读取频繁的时候，悲观锁适合在写入频繁的时候。
7. 互斥锁：对共享数据进行锁定，保证同一时刻只能有一个线程去操作。
8. 死锁：指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，他们都将无法进行下去。
9. 主从复制：例：一台redis服务器的数据，复制到其他的redis服务器。前者称为主节点，后者称为从节点，数据的复制是单向的，只能由主节点到从节点。主节点以写为主，从节点以读为主。
   1. 读写分离：就比如主从复制的模式，主节点负责写，从节点负责读。
   2. 数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式
   3. 故障恢复：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复，实际上是一种服务的冗余
   4. 负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写redis数据时，应用连接主节点，读redis数据时，应用连接从节点），分担服务器负载，尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高redis服务的并发量。
   5. 高可用（集群）基石：除了上述作用外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是redis高可用的基础。