个人知识体系梳理

# 后端.NET

学习站一览：

1. <https://docs.microsoft.com/en-us/aspnet/> 微软官方的ASP.NET 和core 学习地址
2. <https://www.runoob.com/csharp/csharp-tutorial.html>菜鸟教程C#基础
3. <https://www.infoq.cn/article/rest-introduction/> 什么是Rest风格的Api
4. <https://www.cnblogs.com/wintersun/p/6219259.html> 微服务

## C#基础

//todo 包含C#,MVC,WebApi,ORM,注入框架,单元测试,Migration,计划任务,异常处理,权限认证,log日志,事务处理,反射,委托,线程,泛型,封装,继承,多态,常见的设计模式,编码规范,CQRS,接口抽象类,SOLID原则，DRY原则，service bus,redis,rabbit mq

C#基础 <https://www.runoob.com/csharp/csharp-tutorial.html>

1. 数据类型

在 C# 中，变量分为以下几种类型：

* 值类型（Value types）比如 **int、char、float**
* 引用类型（Reference types）**内置的** 引用类型有：**object**、**dynamic** 和 **string**
* 指针类型（Pointer types）指针类型变量存储另一种类型的内存地址。C# 中的指针与 C 或 C++ 中的指针有相同的功能。 Int\* intPointer

当一个值类型转换为对象类型时，则被称为 **装箱**；另一方面，当一个对象类型转换为值类型时，则被称为 **拆箱**。

1. 类型转换

显式转换，隐式转换

1. 常量 常量是固定值，程序执行期间不会改变。const Pi = 3.14 //声明时初始化
2. 运算符 算术运算符+-，关系运算符=,!=，逻辑运算符&|，位运算符，赋值运算符
3. 判断，if,if else, 嵌套if, switch
4. 循环，for,foreach, while, do while. break, continue;
5. 封装，**封装** 被定义为"把一个或多个项目封闭在一个物理的或者逻辑的包中"。在面向对象程序设计方法论中，封装是为了防止对实现细节的访问。

C# 封装根据具体的需要，设置使用者的访问权限，并通过 **访问修饰符** 来实现。

* public：所有对象都可以访问；
* private：对象本身在对象内部可以访问；
* protected：只有该类对象及其子类对象可以访问
* internal：同一个程序集的对象可以访问；
* protected internal：访问限于当前程序集或派生自包含类的类型。

1. 方法 注意ref和out的用法，ref 将值类型的数据已引用的方式传递，out可以让一个方法返回多个结果
2. 可空类型，引用都是可空类型，而值类型可以通过?的方式来声明例如bool?，int?

注意??的用法num1 = num2 ?? 5.34;

1. 数组， 数组是一个存储相同类型元素的**固定大小**的顺序集合

double[] balance = new double[10];

double[] balance = { 2340.0, 4523.69, 3421.0};

int [] marks = new int[] { 99, 98, 92, 97, 95};

1. 结构体，结构体是值类型数据结构。它使得一个单一变量可以存储各种数据类型的相关数据。**struct** 关键字用于创建结构体。

C# 中的结构有以下特点：

* 结构可带有方法、字段、索引、属性、运算符方法和事件。
* 结构可定义构造函数，但不能定义析构函数。但是，您不能为结构定义默认的构造函数。默认的构造函数是自动定义的，且不能被改变。
* 与类不同，结构不能继承其他的结构或类。
* 结构不能作为其他结构或类的基础结构。
* 结构可实现一个或多个接口。
* 结构成员不能指定为 abstract、virtual 或 protected。
* 当您使用 **New** 操作符创建一个结构对象时，会调用适当的构造函数来创建结构。与类不同，结构可以不使用 New 操作符即可被实例化。
* 如果不使用 New 操作符，只有在所有的字段都被初始化之后，字段才被赋值，对象才被使用。

## 类 vs 结构

类和结构有以下几个基本的不同点：

* 类是引用类型，结构是值类型。
* 结构不支持继承。
* 结构不能声明默认的构造函数。

1. 枚举，枚举是一组命名整型常量。枚举类型是使用 **enum** 关键字声明的，C# 枚举是值类型，枚举包含自己的值，且不能继承或传递继承。
2. 继承，继承是面向对象程序设计中最重要的概念之一。继承允许我们根据一个类来定义另一个类，这使得创建和维护应用程序变得更容易。同时也有利于重用代码和节省开发时间。当创建一个类时，程序员不需要完全重新编写新的数据成员和成员函数，只需要设计一个新的类，继承了已有的类的成员即可。这个已有的类被称为的**基类**，这个新的类被称为**派生类**。**多重继承**指的是一个类别可以同时从多于一个父类继承行为与特征的功能。与单一继承相对，单一继承指一个类别只可以继承自一个父类。

**C# 不支持多重继承**。但是，您可以使用接口来实现多重继承

1. 多态，多态是同一个行为具有多个不同表现形式或形态的能力。

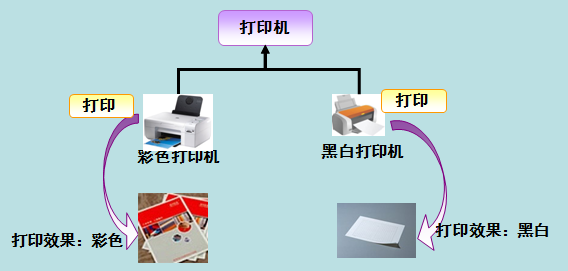
**多态性**意味着有多重形式。在面向对象编程范式中，多态性往往表现为"一个接口，多个功能"。

多态性可以是静态的或动态的。在**静态多态性**中，函数的响应是在编译时发生的。在**动态多态性**中，函数的响应是在运行时发生的。

在 C# 中，每个类型都是多态的，因为包括用户定义类型在内的所有类型都继承自 Object。

C# 允许您使用关键字 **abstract** 创建抽象类，用于提供接口的部分类的实现。当一个派生类继承自该抽象类时，实现即完成。**抽象类**包含抽象方法，抽象方法可被派生类实现。派生类具有更专业的功能。

请注意，下面是有关抽象类的一些规则：

* 您不能创建一个抽象类的实例。
* 您不能在一个抽象类外部声明一个抽象方法。
* 通过在类定义前面放置关键字 **sealed**，可以将类声明为**密封类**。当一个类被声明为 **sealed** 时，它不能被继承。抽象类不能被声明为 sealed。
* 当有一个定义在类中的函数需要在继承类中实现时，可以使用**虚方法**。
* 虚方法是使用关键字 **virtual** 声明的。
* 虚方法可以在不同的继承类中有不同的实现。
* 对虚方法的调用是在运行时发生的。
* 动态多态性是通过 **抽象类** 和 **虚方法** 实现的。

1. 接口，接口定义了所有类继承接口时应遵循的语法合同。接口定义了语法合同 **"是什么"** 部分，派生类定义了语法合同 **"怎么做"** 部分。

接口定义了属性、方法和事件，这些都是接口的成员。接口只包含了成员的声明。成员的定义是派生类的责任。接口提供了派生类应遵循的标准结构。

接口使得实现接口的类或结构在形式上保持一致。

抽象类在某种程度上与接口类似，但是，它们大多只是用在当只有少数方法由基类声明由派生类实现时。

1. 命名空间，**命名空间**的设计目的是提供一种让一组名称与其他名称分隔开的方式。在一个命名空间中声明的类的名称与另一个命名空间中声明的相同的类的名称不冲突。
2. 预处理器指令，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。所有的预处理器指令都是以 # 开始。且在一行上，只有空白字符可以出现在预处理器指令之前。预处理器指令不是语句，所以它们不以分号（;）结束。
3. 正则表达式，**正则表达式** 是一种匹配输入文本的模式。
4. 异常处理，异常是在程序执行期间出现的问题。C# 中的异常是对程序运行时出现的特殊情况的一种响应，比如尝试除以零。异常提供了一种把程序控制权从某个部分转移到另一个部分的方式。C# 异常处理时建立在四个关键词之上的：**try**、**catch**、**finally** 和 **throw。**

## C# 中的异常类

C# 异常是使用类来表示的。C# 中的异常类主要是直接或间接地派生于 **System.Exception** 类。**System.ApplicationException** 和 **System.SystemException** 类是派生于 System.Exception 类的异常类。

**System.ApplicationException** 类支持由应用程序生成的异常。所以程序员定义的异常都应派生自该类。

**System.SystemException** 类是所有预定义的系统异常的基类。

|  |  |
| --- | --- |
| **异常类** | **描述** |
| System.IO.IOException | 处理 I/O 错误。 |
| System.IndexOutOfRangeException | 处理当方法指向超出范围的数组索引时生成的错误。 |
| System.ArrayTypeMismatchException | 处理当数组类型不匹配时生成的错误。 |
| System.NullReferenceException | 处理当依从一个空对象时生成的错误。 |
| System.DivideByZeroException | 处理当除以零时生成的错误。 |
| System.InvalidCastException | 处理在类型转换期间生成的错误。 |
| System.OutOfMemoryException | 处理空闲内存不足生成的错误。 |
| System.StackOverflowException | 处理栈溢出生成的错误。 |

1. 文件的输入输出，一个 **文件** 是一个存储在磁盘中带有指定名称和目录路径的数据集合。当打开文件进行读写时，它变成一个 **流。**从根本上说，流是通过通信路径传递的字节序列。有两个主要的流：**输入流** 和 **输出流**。**输入流**用于从文件读取数据（读操作），**输出流**用于向文件写入数据（写操作）。

## C# I/O 类

System.IO 命名空间有各种不同的类，用于执行各种文件操作，如创建和删除文件、读取或写入文件，关闭文件等。

下表列出了一些 System.IO 命名空间中常用的非抽象类：

|  |  |
| --- | --- |
| **I/O 类** | **描述** |
| BinaryReader | 从二进制流读取原始数据。 |
| BinaryWriter | 以二进制格式写入原始数据。 |
| BufferedStream | 字节流的临时存储。 |
| Directory | 有助于操作目录结构。 |
| DirectoryInfo | 用于对目录执行操作。 |
| DriveInfo | 提供驱动器的信息。 |
| File | 有助于处理文件。 |
| FileInfo | 用于对文件执行操作。 |
| FileStream | 用于文件中任何位置的读写。 |
| MemoryStream | 用于随机访问存储在内存中的数据流。 |
| Path | 对路径信息执行操作。 |
| StreamReader | 用于从字节流中读取字符。 |
| StreamWriter | 用于向一个流中写入字符。 |
| StringReader | 用于读取字符串缓冲区。 |
| StringWriter | 用于写入字符串缓冲区。 |

1. 特性，**特性（Attribute）**是用于在运行时传递程序中各种元素（比如类、方法、结构、枚举、组件等）的行为信息的声明性标签。特性（Attribute）用于添加元数据，如编译器指令和注释、描述、方法、类等其他信息

## 预定义特性（Attribute）

.Net 框架提供了三种预定义特性：

* AttributeUsage
* Conditional
* Obsolete

**重点，如何自定义特性**。一个新的自定义特性应派生自 **System.Attribute** 类

1. 反射，反射指程序可以访问、检测和修改它本身状态或行为的一种能力。程序集包含模块，而模块包含类型，类型又包含成员。反射则提供了封装程序集、模块和类型的对象。您可以使用反射动态地创建类型的实例，将类型绑定到现有对象，或从现有对象中获取类型。然后，可以调用类型的方法或访问其字段和属性。

### 优缺点

优点：

* 1、反射提高了程序的灵活性和扩展性。
* 2、降低耦合性，提高自适应能力。
* 3、它允许程序创建和控制任何类的对象，无需提前硬编码目标类。

缺点：

* 1、性能问题：使用反射基本上是一种解释操作，用于字段和方法接入时要远慢于直接代码。因此反射机制主要应用在对灵活性和拓展性要求很高的系统框架上，普通程序不建议使用。
* 2、使用反射会模糊程序内部逻辑；程序员希望在源代码中看到程序的逻辑，反射却绕过了源代码的技术，因而会带来维护的问题，反射代码比相应的直接代码更复杂。

## 反射（Reflection）的用途

反射（Reflection）有下列用途：

* 它允许在运行时查看特性（attribute）信息。
* 它允许审查集合中的各种类型，以及实例化这些类型。
* 它允许延迟绑定的方法和属性（property）。
* 它允许在运行时创建新类型，然后使用这些类型执行一些任务。

1. 属性，**属性（Property）** 是类（class）、结构（structure）和接口（interface）的命名（named）成员。类或结构中的成员变量或方法称为 **域（Field）**。属性（Property）是域（Field）的扩展，且可使用相同的语法来访问。它们使用 **访问器（accessors）** 让私有域的值可被读写或操作。属性（Property）不会确定存储位置。相反，它们具有可读写或计算它们值的 **访问器（accessors）**。
2. 索引器，**索引器（Indexer）** 允许一个对象可以像数组一样被索引。当您为类定义一个索引器时，该类的行为就会像一个 **虚拟数组（virtual array）** 一样。您可以使用数组访问运算符（[ ]）来访问该类的实例。看过实例后感觉索引器像是数组，但索引本身可以不是int,也可以是string或其他类型。
3. 委托，C# 中的委托（Delegate）类似于 C 或 C++ 中函数的指针。**委托（Delegate）** 是存有对某个方法的引用的一种引用类型变量。引用可在运行时被改变。委托（Delegate）特别用于实现事件和回调方法。所有的委托（Delegate）都派生自 **System.Delegate** 类。委托可以组播。
4. 事件，**Event** 基本上说是一个用户操作，如按键、点击、鼠标移动等等，或者是一些出现，如系统生成的通知。应用程序需要在事件发生时响应事件。例如，中断。事件是用于进程间通信。

## 通过事件使用委托

事件在类中声明且生成，且通过使用同一个类或其他类中的委托与事件处理程序关联。包含事件的类用于发布事件。这被称为 **发布器（publisher）** 类。其他接受该事件的类被称为 **订阅器（subscriber）** 类。事件使用 **发布-订阅（publisher-subscriber）** 模型。

**发布器（publisher）** 是一个包含事件和委托定义的对象。事件和委托之间的联系也定义在这个对象中。发布器（publisher）类的对象调用这个事件，并通知其他的对象。

**订阅器（subscriber）** 是一个接受事件并提供事件处理程序的对象。在发布器（publisher）类中的委托调用订阅器（subscriber）类中的方法（事件处理程序）。

1. 集合，Collection类是专门用于数据存储和检索的类。这些类提供了对栈（stack）、队列（queue）、列表（list）和哈希表（hash table）的支持。大多数集合类实现了相同的接口。集合（Collection）类服务于不同的目的，如为元素动态分配内存，基于索引访问列表项等等。这些类创建 Object 类的对象的集合。在 C# 中，Object 类是所有数据类型的基类。

## 各种集合类和它们的用法

下面是各种常用的 **System.Collection** 命名空间的类。点击下面的链接查看细节。

|  |  |
| --- | --- |
| **类** | **描述和用法** |
| [动态数组（ArrayList）](https://www.runoob.com/csharp/csharp-arraylist.html) | 它代表了可被单独**索引**的对象的有序集合。  它基本上可以替代一个数组。但是，与数组不同的是，您可以使用**索引**在指定的位置添加和移除项目，动态数组会自动重新调整它的大小。它也允许在列表中进行动态内存分配、增加、搜索、排序各项。 |
| [哈希表（Hashtable）](https://www.runoob.com/csharp/csharp-hashtable.html) | 它使用**键**来访问集合中的元素。  当您使用键访问元素时，则使用哈希表，而且您可以识别一个有用的键值。哈希表中的每一项都有一个**键/值**对。键用于访问集合中的项目。 |
| [排序列表（SortedList）](https://www.runoob.com/csharp/csharp-sortedlist.html) | 它可以使用**键**和**索引**来访问列表中的项。  排序列表是数组和哈希表的组合。它包含一个可使用键或索引访问各项的列表。如果您使用索引访问各项，则它是一个动态数组（ArrayList），如果您使用键访问各项，则它是一个哈希表（Hashtable）。集合中的各项总是按键值排序。 |
| [堆栈（Stack）](https://www.runoob.com/csharp/csharp-stack.html) | 它代表了一个**后进先出**的对象集合。  当您需要对各项进行后进先出的访问时，则使用堆栈。当您在列表中添加一项，称为**推入**元素，当您从列表中移除一项时，称为**弹出**元素。 |
| [队列（Queue）](https://www.runoob.com/csharp/csharp-queue.html) | 它代表了一个**先进先出**的对象集合。  当您需要对各项进行先进先出的访问时，则使用队列。当您在列表中添加一项，称为**入队**，当您从列表中移除一项时，称为**出队**。 |
| [点阵列（BitArray）](https://www.runoob.com/csharp/csharp-bitarray.html) | 它代表了一个使用值 1 和 0 来表示的**二进制**数组。  当您需要存储位，但是事先不知道位数时，则使用点阵列。您可以使用**整型索引**从点阵列集合中访问各项，索引从零开始。 |

1. **泛型（Generic）** 允许您延迟编写类或方法中的编程元素的数据类型的规范，直到实际在程序中使用它的时候。换句话说，泛型允许您编写一个可以与任何数据类型一起工作的类或方法。您可以通过数据类型的替代参数编写类或方法的规范。当编译器遇到类的构造函数或方法的函数调用时，它会生成代码来处理指定的数据类型。

## 泛型（Generic）的特性

使用泛型是一种增强程序功能的技术，具体表现在以下几个方面：

* 它有助于您最大限度地重用代码、保护类型的安全以及提高性能。
* 您可以创建泛型集合类。.NET 框架类库在 *System.Collections.Generic* 命名空间中包含了一些新的泛型集合类。您可以使用这些泛型集合类来替代 *System.Collections* 中的集合类。
* 您可以创建自己的泛型接口、泛型类、泛型方法、泛型事件和泛型委托。
* 您可以对泛型类进行约束以访问特定数据类型的方法。
* 关于泛型数据类型中使用的类型的信息可在运行时通过使用反射获取。

1. **匿名方法（Anonymous methods）** 提供了一种传递代码块作为委托参数的技术。匿名方法是没有名称只有主体的方法，您可以使用委托对象调用可由委托引用的方法，匿名方法中您不需要指定返回类型，它是从方法主体内的 return 语句推断的。匿名方法是通过使用 **delegate** 关键字创建委托实例来声明的。
2. **线程** 被定义为程序的执行路径。每个线程都定义了一个独特的控制流。如果您的应用程序涉及到复杂的和耗时的操作，那么设置不同的线程执行路径往往是有益的，每个线程执行特定的工作。线程是**轻量级进程**。一个使用线程的常见实例是现代操作系统中并行编程的实现。使用线程节省了 CPU 周期的浪费，同时提高了应用程序的效率。到目前为止我们编写的程序是一个单线程作为应用程序的运行实例的单一的过程运行的。但是，这样子应用程序同时只能执行一个任务。为了同时执行多个任务，它可以被划分为更小的线程

## 线程生命周期

线程生命周期开始于 System.Threading.Thread 类的对象被创建时，结束于线程被终止或完成执行时。

下面列出了线程生命周期中的各种状态：

* **未启动状态**：当线程实例被创建但 Start 方法未被调用时的状况。
* **就绪状态**：当线程准备好运行并等待 CPU 周期时的状况。
* **不可运行状态**：下面的几种情况下线程是不可运行的：
  + 已经调用 Sleep 方法
  + 已经调用 Wait 方法
  + 通过 I/O 操作阻塞
* **死亡状态**：当线程已完成执行或已中止时的状况。

## 主线程

在 C# 中，**System.Threading.Thread** 类用于线程的工作。它允许创建并访问多线程应用程序中的单个线程。进程中第一个被执行的线程称为**主线程**。

当 C# 程序开始执行时，主线程自动创建。使用 **Thread** 类创建的线程被主线程的子线程调用。您可以使用 Thread 类的 **CurrentThread**属性访问线程。

# Rest

五条关键原则列举如下：

* 为所有“事物”定义 ID 例如xxx.com/customer/6, xxx.com/order/15
* 将所有事物链接在一起 例如xxx.com/customer/6/orders
* 使用标准方法 使用标准的http动词，
* 资源多重表述 返回的数据可以是xml或者json或者其他格式
* 无状态通信 http本身是无状态的通信协议

# Sql

## 游标：

我们在处理数据的时候，经常会出现需要循环处理数据的需求，如果我们能用CTE或者其他语句处理的话，没有问题，但有时候往往处理起来比较困难，这时候我们可以选择使用游标处理，选择使用哪种形式，要考虑效率问题，一般游标效率不高，但也有适合使用的场景。

       游标分为静态游标和动态游标，静态游标的数据是固定的，不会因为数据表的改变而改变；动态游标的数据是随着数据表变化而变化的，游标默认是动态游标，通过关键字STATIC设置，OK，上测试数据

DECLARE @id NVARCHAR(50) , @name NVARCHAR(50) --声明变量，需要读取的数据

DECLARE cur CURSOR --去掉STATIC关键字即可

FOR

SELECT Id,jobName FROM [FieldSalesV2].[sale].[S\_Jobs]

OPEN cur --打开游标

FETCH NEXT FROM cur INTO @id, @name --取数据

WHILE ( @@fetch\_status = 0 ) --判断是否还有数据

BEGIN

SELECT '数据: ' + RTRIM(@id) + @name

FETCH NEXT FROM cur INTO @id, @name --这里一定要写取下一条数据

END

CLOSE cur --关闭游标

DEALLOCATE cur

## 函数

Sql 中的函数包含三类函数，1.标量值函数。2.表值函数。3.多语句表值函数

1 **创建标量值函数：**

2 create function fun\_name (

3 变量声明区

4 )

5 returns return\_type

6 as

7 begin

8 操作主体区

9 end

10 **创建表值函数：**

11 内联表值函数：

12 create function fun\_name (

13 变量声明区

14 )

15 returns table

16 as

17 return (满足条件的查询语句)

18 **多语句表值函数**：

19 create function fun\_name (

20 变量声明区

21 )

22 returns @var\_name table

23 (

24 表变量定义区

25 )

26 as

27 begin

28 函数主体区

29 return

30 end

31 **调用函数：**

32 标量函数：

33 set @var\_name = fun\_name(参数列表)

34 表值函数：

35 select \* from fun\_name(参数列表)

36 **删除函数：**

37 drop function fun\_name

## 存储过程：

<https://blog.csdn.net/u014088408/article/details/93620799>

1.什么是存储过程

存储过程是SQL语句和控制语句的预编译集合，保存在数据库里，可由应用程序调用执行，而且允许用户声明变量、逻辑控制语句及其他强大的编程功能。保存在SQLServer中，通过名称和参数执行，也可以返回结果。

2.为什么要使用存储过程

应用程序通过T-SQL语句到服务器的过程是不安全的。

      1） 数据不安全

      2）每次提交SQL代码都要经过语法编译后在执行，影响应用程序的运行性能

      3） 网络流量大

3.存错过程的优缺点

  (1) 优点

     1） 模块化呈现设计

     2） 执行速度快，效率高

     3） 减少网络流量

     4） 具有良好的安全性

 (2) 缺点

    1）不可移植性，每种数据库的内部编程语法都不太相同，当你的系统需要兼容多种数据库时最好不要用存储过程。

    2）学习成本高，DBA一般都擅长写存储过程，但并不是每个程序员都能写好存储过程，除非你的团队有较多的开发人员熟悉            写存储过程，否则后期系统维护会产生问题。

3）业务逻辑多处存在，采用存储过程后也就意味着你的系统有一些业务逻辑不是在应用程序里处理，这种架构会增加一些系           统维护和调试成本。

# 函数与存储过程的差异

# Get和Post

<https://www.zhihu.com/question/28586791/answer/767316172>

1. 幂等“（Idempotent),表示请求没有副作用，不会更改数据本身
2. 所以其实并不是GET只能用url，而是浏览器直接发出的GET只能由一个url触发。所以没办法，GET上要在url之外带一些参数就只能依靠url上附带querystring。但是HTTP协议本身并没有这个限制
3. 浏览器在POST一个表单时，url上也可以带参数，只要<form action="url" >里的url带querystring就行。只不过表单里面的那些用<input> 等标签经过用户操作产生的数据都在会在body里。
4. 因此我们一般会**泛泛的说**“GET请求没有body，只有url，请求数据放在url的querystring中；POST请求的数据在body中“。但这种情况仅限于浏览器发请求的场景。

# TCP/IP知识

<https://zhuanlan.zhihu.com/p/72587882>

# 微服务

<https://baijiahao.baidu.com/s?id=1624246157241005608&wfr=spider&for=pc>

# MVC基础

学习地址：<https://www.cnblogs.com/powertoolsteam/p/MVC_one.html>

重点强调四五六节。

# WebApi基础

# 数据库知识

//todo 基本的增删改查，常见聚合查询，试图，存储过程，触发器，游标，简单的分页查询，常见的性能优化，备份，还原，导入，导出

# 前端

//todo javascript基础,html基础，css基础，prototype，闭包，模块化，jquery Dom基本操作，bootstrap，lodash，moment，angular,react,vue

# APP开发

//todo ionic,react native.

# 英文

## 自我介绍

## 工作team/group job duty介绍

# 实施与运维

//todo docker,k8s,nginx，linux，X shell,XFtp，jenkins