# 序列化类的部分成员

[JsonObject(MemberSerialization.OptOut)]修饰类，只序列化public属性和public字段，不需要被序列化的属性或字段可以用[JsonIgnoreAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonIgnoreAttribute.htm)修饰。

[JsonObject(MemberSerialization.OptIn)]修饰类，只序列化被 [JsonPropertyAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonPropertyAttribute.htm)修饰的属性和字段(无论属性和字段的访问修饰符是public，protected还是private，只要被[JsonPropertyAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonPropertyAttribute.htm)修饰都会被序列化)。

[JsonObject(MemberSerialization.Fields)]修饰类，会序列化所有的字段 (无论字段的访问修饰符是public，protected还是private)，但不会序列化任何属性。不需要被序列化的字段或会生成back-field的属性可以用[JsonIgnoreAttribute](https://www.newtonsoft.com/json/help/html/T_Newtonsoft_Json_JsonIgnoreAttribute.htm)修饰。

默认是1，不需要opt,只用JsonProperty.

    [JsonObject(MemberSerialization.OptOut)]

    public class Student

    {

        public Student()

        {

            Name = "Jack";

            Age = 20;

            Address = "广东珠海";

            Email = "123@qq.com";

            Height = 177;

            HairColor = Color.Aqua;

            weight = 63;

            className = "大二";

            teacher = "玛丽亚";

            girlFriend = "琳娜";

            schoolId = 97251;

            identificationId = 123456789;

        }

        #region Properties

        public string Name { get; set; }

        public int Age { get; set; }

        protected string Address { get; set; }

        protected string Email { get; set; }

        private int Height { get; set; }

        private Color HairColor { get; set; }

        #endregion Properties

        #region Fields

        public double weight;

        public string className;

        protected string teacher;

        protected string girlFriend;

        private int schoolId;

        private int identificationId;

        #endregion Fields

    }

            string json = JsonConvert.SerializeObject(new Student(),Formatting.Indented);

            Console.WriteLine(json);

{

  "weight": 63.0,

  "className": "大二",

  "Name": "Jack",

  "Age": 20

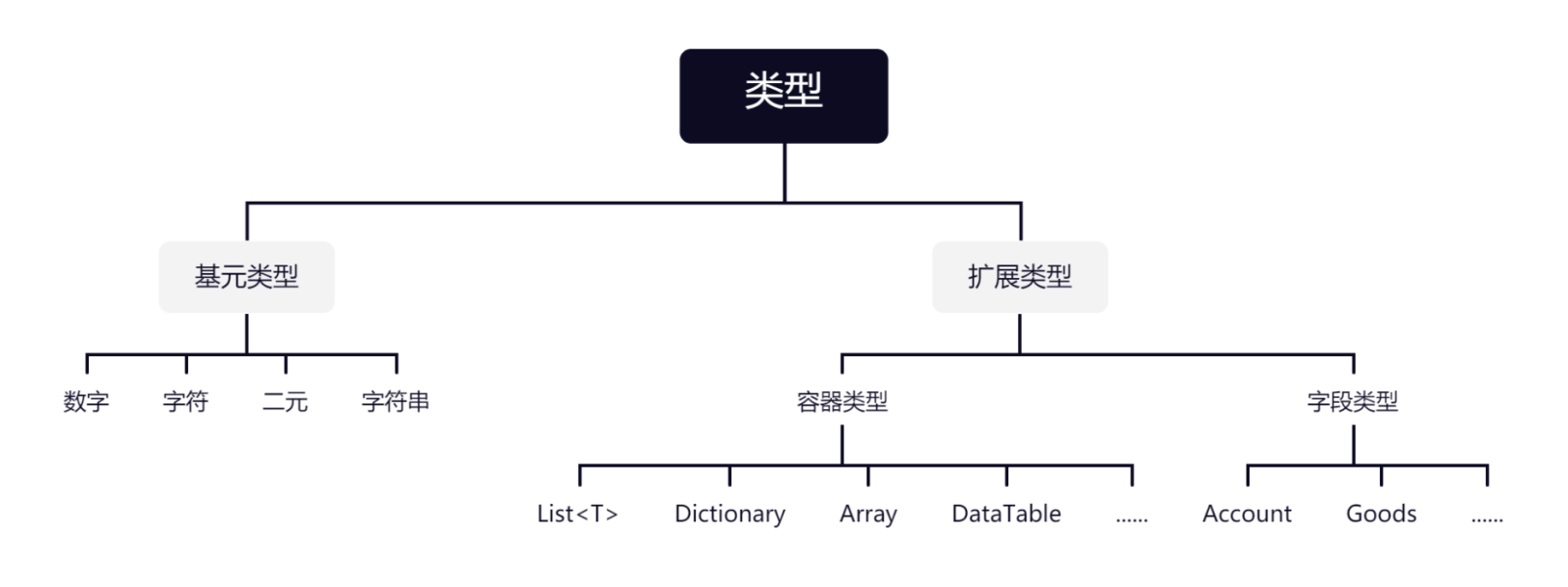
}

## JosnObject JsonProperty ContractResolver

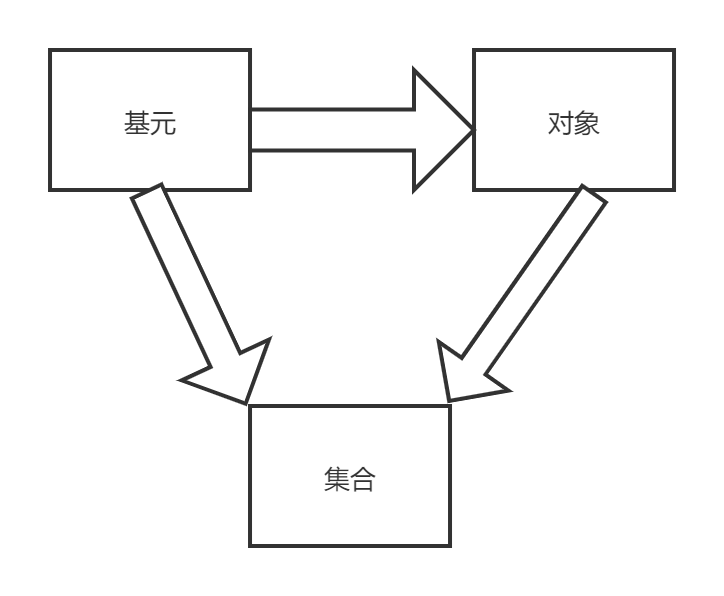
C#的基础类型的序列化规则如下：

【数字】byte，sbyte，short，ushort，int，uint，long，ulong，float，double，decimal；【字符】char；【布尔】true，false；【字符串】string。

类型分类：



类型间关系：



# 面向对象和面向过程

C语言是面向过程的，宏观来看C语言代码，是由比较零散的变量和函数构成。面向对象把变量封装成字段，类就成了变量集合。类的实例方法和静态方法本质上没有任何区别，就相当于把C语言中零散的函数加上命名空间包装起来。即无论面向对象还是面向过程，方法永远是拿来即用，只是面向对象的方法的参数是从字段中获取，面向过程的方法是从零散分布的变量获取。类是不同类型的变量构成的变量集合，通过字段名索引。数组是相同类型的变量的集合，通过下表索引。

对象和数组区别：

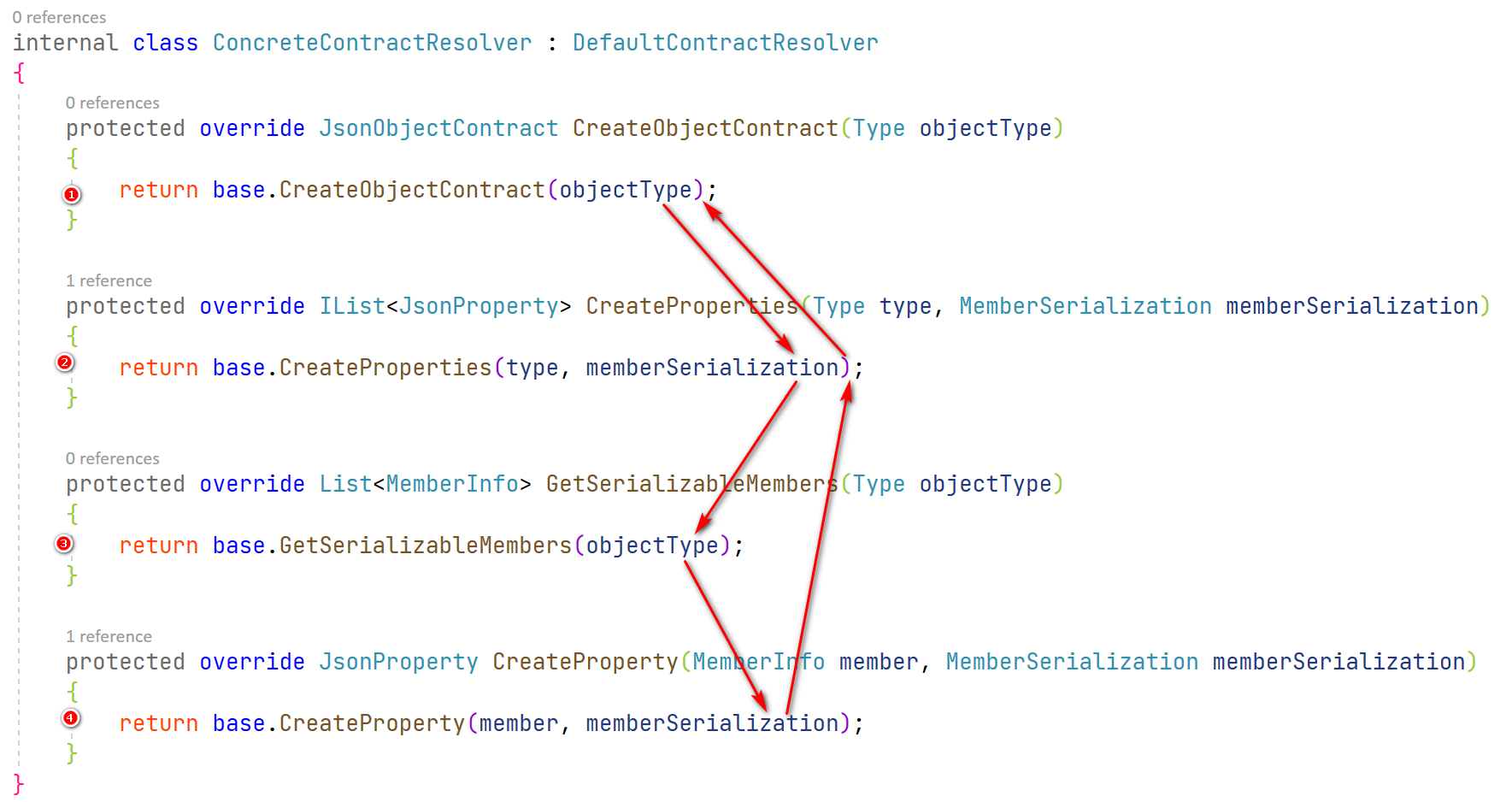
* 对象是名称-值对构成的列表或集合，数组是值构成的列表或集合。
* 数组中所有的值应具有相同的数据类型。

# 序列化

序列化相关设置的三个节点、

1. 根类的所有类：JsonSerializerSettings
2. 具体类的所有属性ContractResolver
3. 具体类的单个属性CreateProperty

优先级: 3 > 2 > 1



## 设置属性在Json中出现的次序

Order的值越大，在Json中出现的次序越靠前。Order的默认值是-1.

internal class ConcreteContractResolver : DefaultContractResolver

{

protected override JsonProperty CreateProperty(MemberInfo member, MemberSerialization memberSerialization)

{

JsonProperty jp = base.CreateProperty(member, memberSerialization);

switch (jp.PropertyName)

{

case "ExecuteResult":

jp.Order = -2;

break;

case "Id":

jp.Order = 0;

break;

}

return jp;

}

}

string json = JsonConvert.SerializeObject(new Message() { Params = new Params() }, new JsonSerializerSettings()

{

ContractResolver = new ConcreteContractResolver() { },

});

Console.WriteLine(json);

输出：

{"ExecuteResult":false,"Key":null,"Params":{"Param1":"\u0000","Param2":0.0},"Id":0}

## LINQ To JSON

JToken – 抽象基类

Jcontainer -能够包含其它JToken的JToken抽象基类

JArray – 表示一个JSON数组（包含一个有序的List）

JObject – 表示一个JSON对象（包含一个IEnumerable）

JProperty – 表示一个JSON属性（在Jobject中是一个name/Jtoken键值对）

Jvalue – 表示一个原生JSON值（number，boolean，string，null）

## .NET Type 🡸🡺 JToken

JValue v = new JValue(1d);

Console.WriteLine(v.Type);

Console.WriteLine((double)v);

v = new JValue('A');

Console.WriteLine(v.Type);

Console.WriteLine((char)v);

v = new JValue(true);

Console.WriteLine(v.Type);

Console.WriteLine((bool)v);

v = new JValue("ABC");

Console.WriteLine(v.Type);

Console.WriteLine((string)v);

v = JValue.CreateNull();

Console.WriteLine(v.Type);

v = new JValue(System.Guid.NewGuid());

Console.WriteLine(v.Type);

Console.WriteLine((Guid)v);

v = new JValue(DateTime.Now);

Console.WriteLine(v.Type);

Console.WriteLine(((DateTime)v));

v = new JValue(new TimeSpan(9, 3, 1));

Console.WriteLine(v.Type);

Console.WriteLine((TimeSpan)v);

v = new JValue(new Uri(@"C:\\d.txt"));

Console.WriteLine(v.Type);

Console.WriteLine((Uri)v);

JObject obj = JObject.FromObject(new Person());

Console.WriteLine(obj.Type);

Console.WriteLine(obj.ToObject<Person>());

JArray arr = JArray.FromObject(new int[] { 1, 2, 3 });

Console.WriteLine(arr.Type);

Console.WriteLine(arr.ToObject<int[]>());

JToken jt = JToken.FromObject(new object());

Console.WriteLine(jt.Type);

if (jt is JValue)

{

}

else if (jt is JArray)

{

}

else if (jt is JObject)

{

}

else if (jt is JProperty)

{

}

if (jt.Type == JTokenType.Null || jt.Type == JTokenType.Object || jt.Type == JTokenType.Array)

{

}

else

{

if (jt.Type == JTokenType.Integer || jt.Type == JTokenType.Date || jt.Type == JTokenType.Guid || jt.Type == JTokenType.Boolean || jt.Type == JTokenType.String)

{

}

}

## JToken 🡸🡺 JSON

// JSON ==> JToken

JObject obj;

JArray arr;

JToken token;

JProperty property;

obj = JObject.Parse("{}");

arr = JArray.Parse("[]");

token = JToken.Parse("1");

if (token.Type == JTokenType.Object)

{

obj = token as JObject;

}

else if(token.Type == JTokenType.Array)

{

arr = token as JArray;

}

else if(token.Type == JTokenType.Property)

{

property = token as JProperty;

// property.Name; 属性名称

// property.Value; 属性值，为JToken

}

// JToken ==> JSON

string json = obj.ToString(Formatting.Indented);

json = arr.ToString(Formatting.Indented);

json = token.ToString(Formatting.Indented);

## 遍历JObject

foreach (JProperty jp in obj.Properties())

{

Console.WriteLine(jp.Name);

Console.WriteLine(jp.Value.Type); //jp.Value的类型是JToken

}

## 遍历JArray

foreach (JToken item in arr)

{

}

# 深克隆和深判等

序列化对象得到Json，比较Json字符串，进行深判等。

对待深克隆的对象的Json进行反序列化，得到副本，完成深克隆。