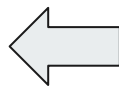
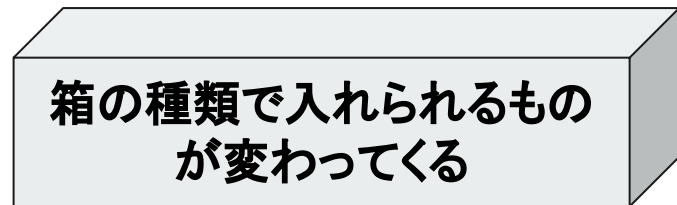


参照と実体

変数

- ・言ってしまうば「箱(関)」



1

aという名前の箱
(aという名前の変数)

```
int a = 1;
```

「整数」だけに入れられる箱を用意して
その名前を「a」にして
その箱に「1」を入れる

どのような種類を
使うか

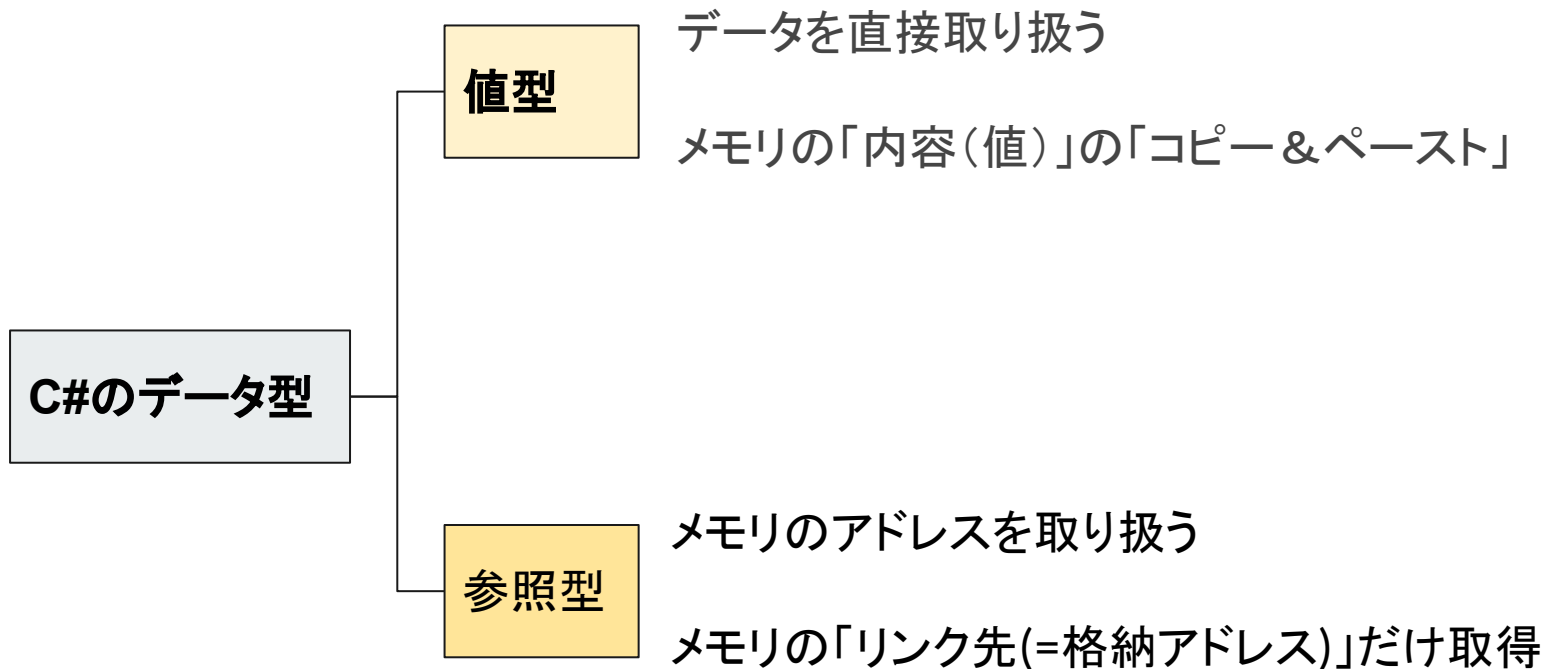
なんという名前を
使うか

中身に何を入れるか

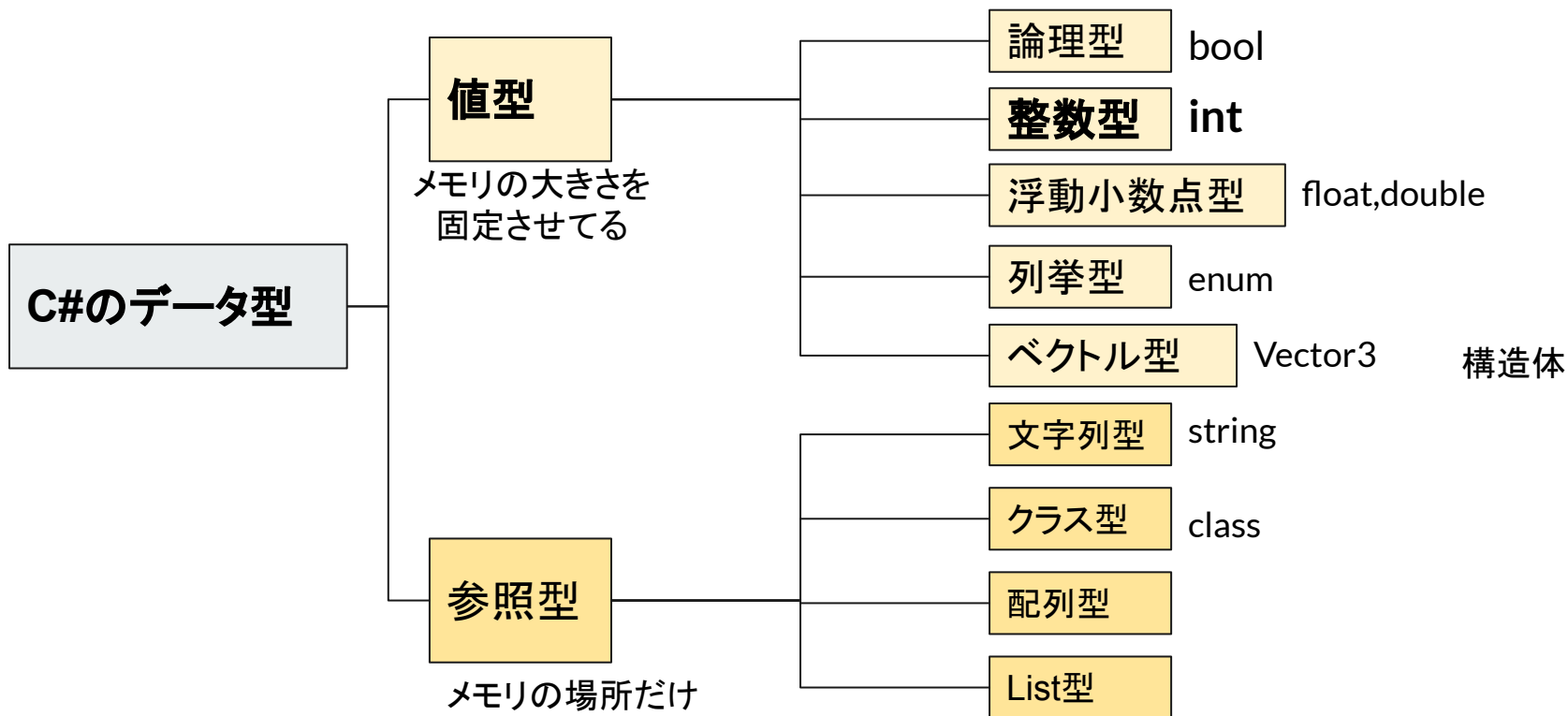


自分で定義して使っていく

C#で利用できるデータ型



C#で利用できるデータ型



ちょっとした違いで変わる参照型・値型の例

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```

```
public class Test : MonoBehaviour
{
    Vector3 t;

    void Start()
    {
        t = transform.position;
    }

    void Update()
    {
        t += Vector3.right;
    }
}
```

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```

ちょっとした違いで変わる参照型・値型の例

1フレーム毎に右に動く (transformは参照型)

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```

1フレーム経っても動かない (transform.positionは値型)

```
public class Test : MonoBehaviour
{
    Vector3 t;

    void Start()
    {
        t = transform.position;
    }

    void Update()
    {
        t += Vector3.right;
    }
}
```

参照型の例

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

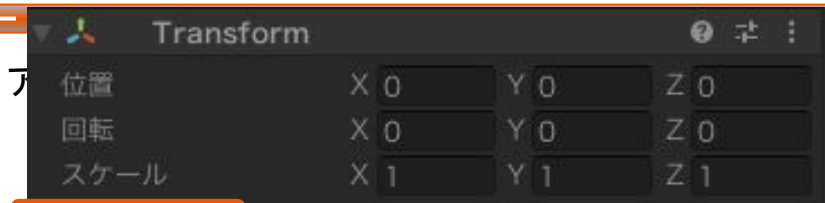
10642番地

参照型①

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        Transformは10578番地
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```



10578番地	Transform
10586番地	変数名 transform Transformは10578番地
10594番地	
10602番地	Transform型 変数t
10610番地	
10618番地	
10626番地	
10634番地	
10642番地	

参照型①

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        Transformは10578番地
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```

アドレス(住所)

10570番地

10578番地

10586番地

10594番地

10602番地

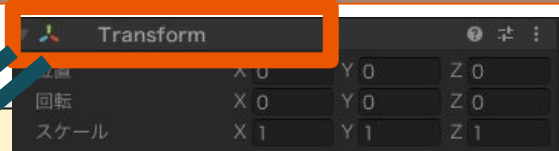
10610番地

10618番地

10626番地

10634番地

10642番地



Transform

変数名 transform

Transformは10578番地

Transform型 変数t

参照型②Start()

```
public class Test : MonoBehaviour
{
    Transform t;

    void Start()
    {
        t = transform;
    }

    void Update()
    {
        t.position += Vector3.right;
    }
}
```

アドレス(住所)

10570番地

10578番地

10586番地

10594番地

10602番地

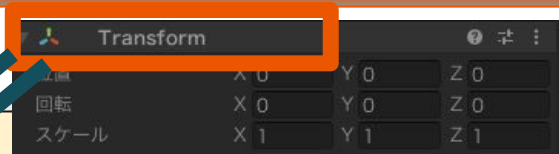
10610番地

10618番地

10626番地

10634番地

10642番地



Transform

変数名 transform

Transformは10578番地

Transform型 変数t

Transformは10578

参照型③Update()

```
public class Test : MonoBehaviour
```

```
{
```

```
    Transform t;
```

```
    void Start()
```

```
    {
```

```
        t = transform;
```

Transformは10578番地

```
    }
```

10578番地の

Transformのposition

← (1.0, 0.0, 0.0)

```
        t.position += Vector3.right;
```

```
    }
```

```
}
```

アドレス(住所)

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

位置	X 0	Y 0	Z 0
回転	X 0	Y 0	Z 0
スケール	X 1	Y 1	Z 1

Transform

変数名 transform

Transformは10578番地

Transform型 変数t

Transformは10578番地

値型の例

```
public class Test : MonoBehaviour
{
    Vector3 t;

    void Start()
    {
        t = transform.position;
    }

    void Update()
    {
        t += Vector3.right;
    }
}
```

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

値型①

```
public class Test : MonoBehaviour
{
    Vector3 t;

    void Start()
    {
        「10578番地のTransform」のpositionという値
        t = transform.position;
    }

    void Update()
    {
        t += Vector3.right;
    }
}
```

アドレス(住所)

10570番地

10578番地

10586番地

4番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

Transform			
位置	X 0	Y 0	Z 0
回転	X 0	Y 0	Z 0
スケール	X 1	Y 1	Z 1

Transform

変数名 transform

Transformは10578番地

Vector3型 変数t

値型②Start()

```
public class Test : MonoBehaviour
{
    Vector3 t;

    void Start()
    {
        「10578番地のTransform」のpositionという値
        t ← transform.position;
        「10578番地のTransform」のpositionという値

        void Update()
        {
            t += Vector3.right;
        }
    }
}
```

アドレス(住所)

10570番地

10578番地

10586番地

4番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

Transform			
位置	X 0	Y 0	Z 0
回転	X 0	Y 0	Z 0
スケール	X 1	Y 1	Z 1

Transform

変数名 transform

Transformは10578番地

Vector3型 変数t

位置の値(0.0 , 0.0 , 0.0)

値型③Update()

```
public class Test : MonoBehaviour
{
    Vector3 t;

    void Start()
    {
        「10578番地のTransform」のpositionという値
        t ← transform.position;
        「10578番地のTransform」のpositionという値

        void Update()
        {
            「10578番地のTransform」のpositionという値
            t += Vector3.right;
            ←(1.0, 0.0, 0.0)
        }
    }
}
```

アドレス(住所)

10570番地

10578番地

10586番地

4番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

Transform			
位置	X 0	Y 0	Z 0
回転	X 0	Y 0	Z 0
スケール	X 1	Y 1	Z 1

Transform

変数名 transform

Transformは10578番地

Vector3型 変数t

位置の値(1.0, 0.0, 0.0)

値型・参照型それぞれの特徴

・参照型

実体が1つ・ほぼ1箇所のメモリ消費・アクセスしに行く時間が発生
変更作業をすると元データが変わる

・値型

1つ1つの実データが作成される・軽いデータならアクセス時間が短い
(変更作業をしても元データは変わらない)

実体(インスタンス)取得

—

コンポーネントをスクリプトから操作

コンポーネントをスクリプトから操作する方法

→「自分で操作したいコンポーネント」を「自分で定義した変数名」に入れて操作する

*Transform*コンポーネントの場合

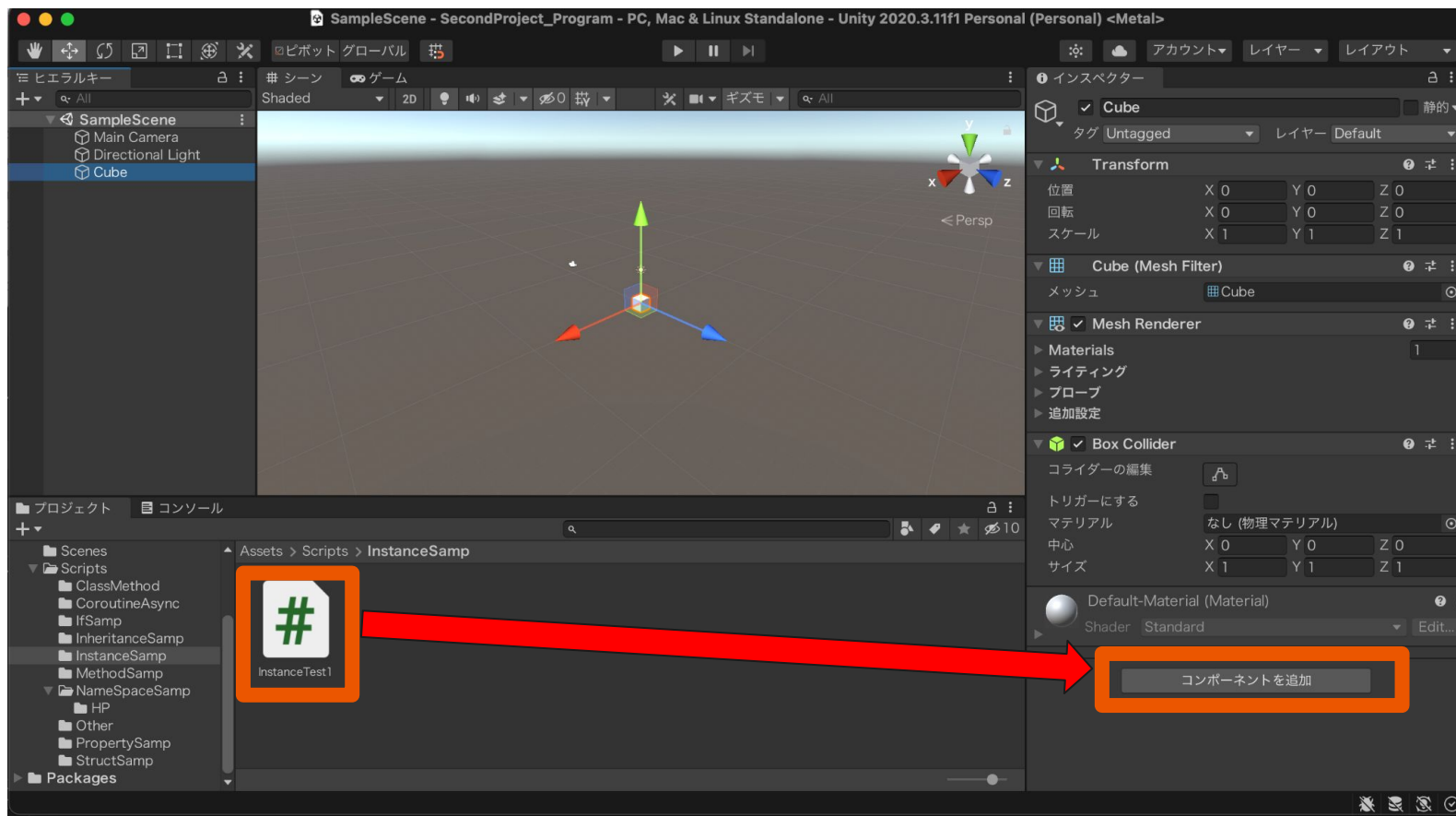
先ほど、Transformというコンポーネントのアドレスが、
あらかじめ用意されている変数名「transform」に入っていました

↑

「変数名transformを使えばTransformの値を変更できる」

といったものをUnityが用意してくれていた

「コンポーネントの取得」をやってみる



「コンポーネントの取得」のやり方

```
GetComponent<型名>();
```

コンポーネントの名前

ゲームオブジェクトについているコンポーネント(のアドレス)

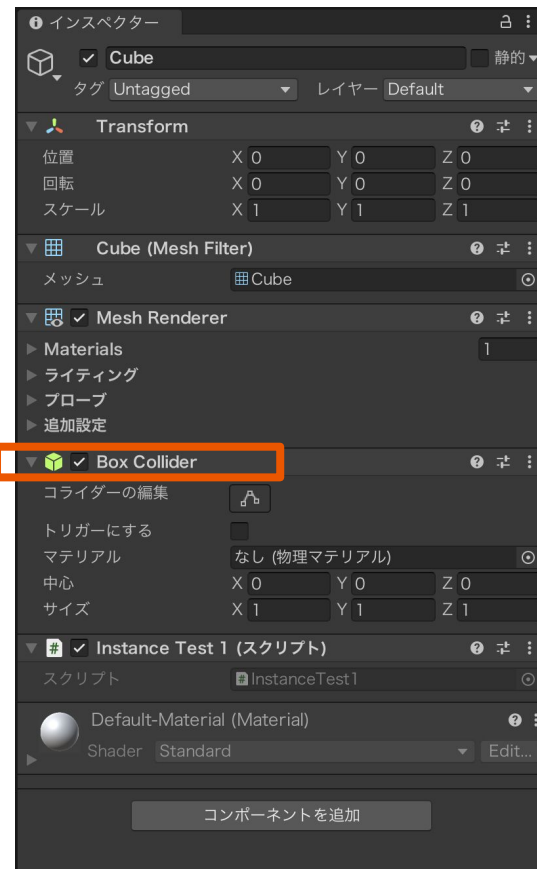
を取得できる

(MonoBehaviourを継承しているから使用できる)

コンポーネントの取得

Start(){}の中に

```
BoxCollider box = null;  
Debug.Log(box);  
box = GetComponent<BoxCollider>();  
Debug.Log(box);
```



「コンポーネント」の取得

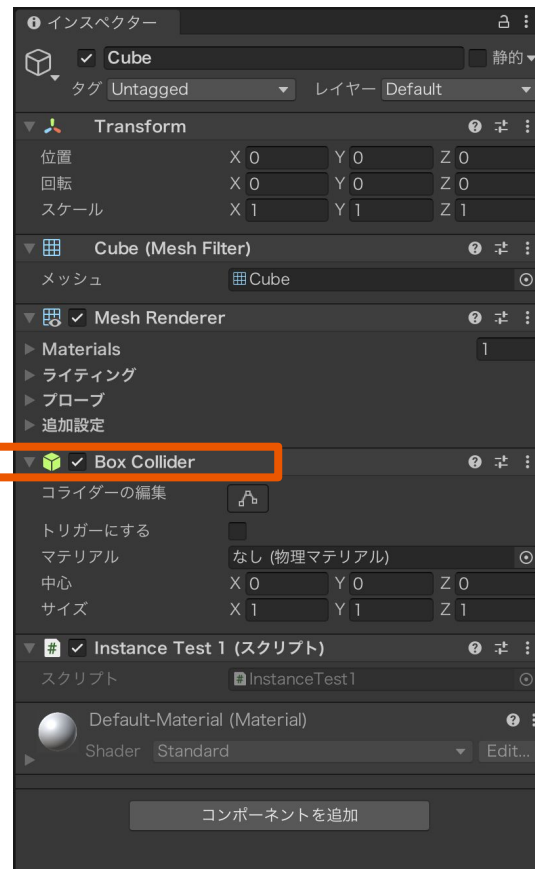
Start()の中に

```
BoxCollider box = null;  
Debug.Log(box);  
box = GetComponent<BoxCollider>();  
Debug.Log(box);
```

アドレス

[16:17:47] Null
UnityEngine.Debug.Log (object)

[16:17:47] Cube (UnityEngine.BoxCollider)
UnityEngine.Debug.Log (object)



「コンポーネント」の取得

「コンポーネント」の取得の利点: ゲーム途中でも自由に値を変えられる

```
BoxCollider box = null;  
box = GetComponent<BoxCollider>();
```

```
Vector3 v = new Vector3();
```

```
v.x = 10;
```

```
v.y = 5;
```

(10, 5, 0)

```
box.size = v;
```

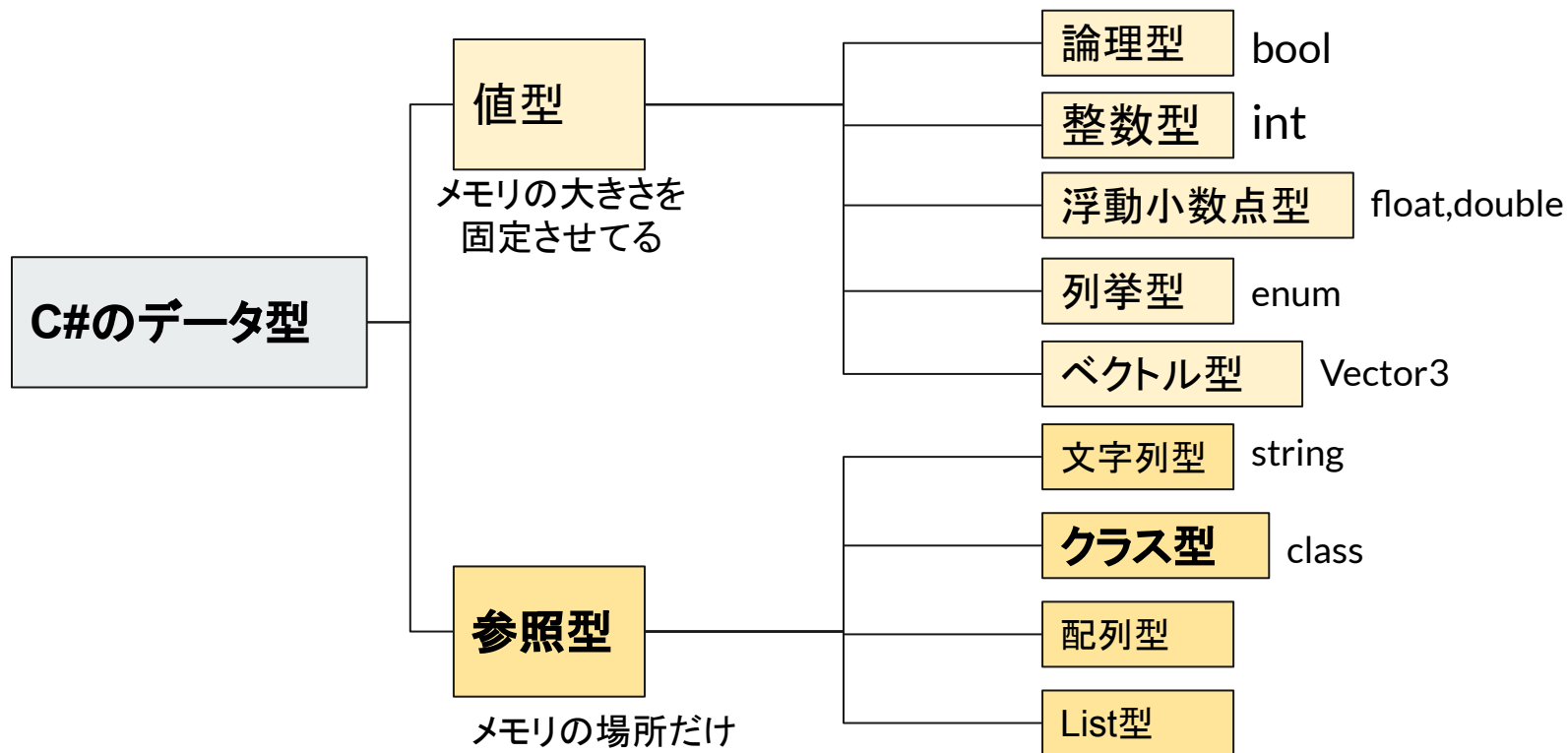


公式リファレンス見るしかない

→ <https://docs.unity3d.com/ja/current/ScriptReference/index.html>

クラス・構造体

C#で利用できるデータ型



クラスのイメージ

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Sample : MonoBehaviour
{
    class Test
    {
        public int HP = 10;
    }

    void Start()
    {
        Test hp;
        hp = new Test();
        Debug.Log(hp.HP);

        // Test hp_display = new Test();
        // Debug.Log(hp_display.HP.ToString());
    }

    void Update()
    {
    }
}
```

ここだけ拡大してみる

クラスのイメージ①

```
class Test
{
    public int HP = 10;
}

void Start()
{
    Test hp;
    hp = new Test();
    Debug.Log(hp.HP);

    // Test hp_display = new Test();
    // Debug.Log(hp_display.HP.ToString());
}
```

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

class型 クラス名Test 設計図

①Test型 変数(箱)

クラスのイメージ

```
class Test
{
    public int HP = 10;
}
```

```
void Start()
{
```

```
    Test hp;
```

```
    hp = new Test();
```

```
    Debug.Log(hp.HP);
```

```
    //Test hp = new Test();
```

```
}
```

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

class型 クラス名 Test 設計図

①Test型 変数名hp ②実体化

②設計図を使いやすい形にする
インスタンス作成・実体作成

クラスのイメージ

```
class Test
{
    public int HP;
    public string name;
}

void Start()
{
    Test hp = new Test();
    Debug.Log(hp.HP); //0
    Debug.Log(hp.name); //null
    hp.name = "上善";
    Debug.Log(hp.name); //上善
}
```

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

class型 クラス名 Test 設計図

①Test型変数名 hp ②実体化

クラス(コンストラクタ)

```
class Test
{
    public int HP;
    public string name;
    public Test()
    {
        Debug.Log("Testクラスのコンストラクタが呼ばれました");
        HP = 100;
        name = "モブ";
    }
}

void Start()
{
    Test hp = new Test(); //Testクラスのコンストラクタが呼ばれました
}
```

メンバー変数・インスタンス変数

インスタンス生成時と同時に実行される (インスタンスの初期化)
特別なメソッド(関数)

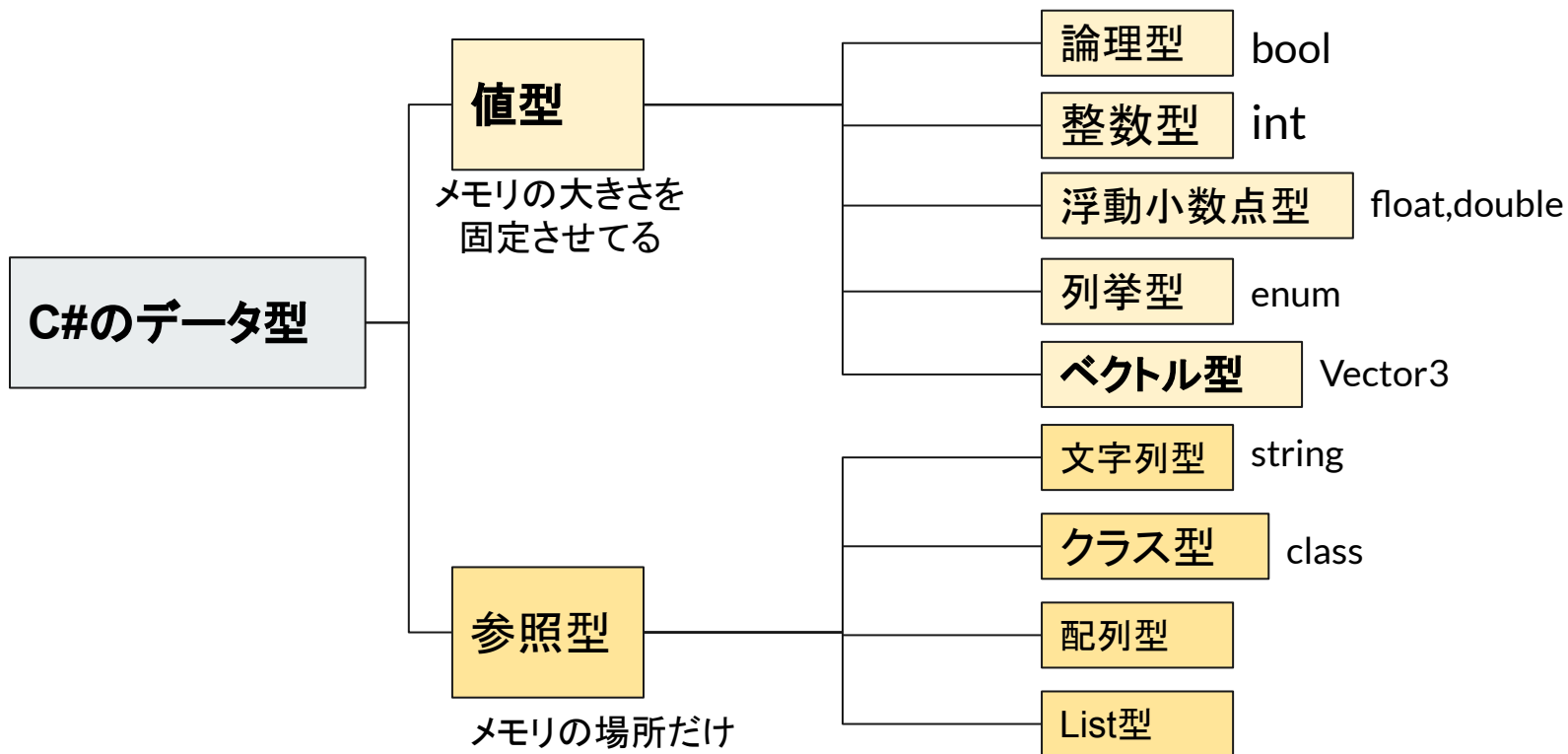
- ・型名と同じ名前のメソッド
- ・他のメソッドと異なり、戻り値の値は書かない

クラス(引数つきコンストラクタ)

```
class Test
{
    public int HP;        //変数宣言
    public string name;   //変数宣言
    public Test(int HP, string name) //引数指定
    {
        Debug.Log("Testクラスのコンストラクタが呼ばれました");
        this.HP = HP;    //「hp.HP」はインスタンス自身。「HP」は引数「hp.HP」に引数を代入
        this.name = name; //インスタンス自身 hp.name = name
    }
}

void Start()
{
    Test hp = new Test(10, "テスト");
    Debug.Log(hp.HP);    //10
    Debug.Log(hp.name);  //テスト
}
```


C#で利用できるデータ型



構造体

基本的にはclassの値型ver

(クラスでできることが構造体ではできないなど、違いがあるので全く同じとは思えない
てください)

```
struct Test
{
    public int HP;
    public string name;
}
void Start()
{
    Test hp = new Test();
    Debug.Log(hp.HP); //0
    Debug.Log(hp.name); //null
    hp.name = "上善";
    Debug.Log(hp.name); //上善
}
```

```
class Test
{
    public int HP;
    public string name;
}
void Start()
{
    Test hp = new Test();
    Debug.Log(hp.HP); //0
    Debug.Log(hp.name); //null
    hp.name = "上善";
    Debug.Log(hp.name); //上善
}
```

構造体のイメージ

```
struct Test
{
    public int HP;
    public string name;
}

void Start()
{
    Test hp = new Test();
    Debug.Log(hp.HP); //0
    Debug.Log(hp.name); //null
    hp.name = "上善";
    Debug.Log(hp.name); //上善
}
```

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

class型 クラス名 Test 設計図

①Test型変数名 hp ②実体化

コンソール画面からみる構造体とクラスの違い

```
class Test
{
    public int HP;
    public Test(int HP) { this.HP = HP; }
}

void Start()
{
    Debug.Log("参照型の場合");

    Test a = new Test(100);
    Test b = a;
    Test c = a;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");

    b.HP = 0;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");
}
```

```
struct Test
{
    public int HP;
    public Test(int HP) { this.HP = HP; }
}

void Start()
{
    Debug.Log("値型の場合");

    Test a = new Test(100);
    Test b = a;
    Test c = a;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");

    b.HP = 0;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");
}
```

クラス(参照型)

```
class Test
{
    public int HP;
    public Test(int HP) { this.HP = HP; }
}

void Start()
{
    Debug.Log("参照型の場合");

    Test a = new Test(100);
    Test b = a;
    Test c = a;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");

    b.HP = 0;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");
}
```



[14:21:51] 参照型の場合
UnityEngine.Debug:Log (object)



[14:21:51] aの値 : 100 , bの値 : 100 , cの値 : 100
UnityEngine.Debug:Log (object)



[14:21:51] aの値 : 0 , bの値 : 0 , cの値 : 0
UnityEngine.Debug:Log (object)

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

class型 Test 設計図

Test型 a 実体 10602番地

Test型 b 10602番地

Test型 c 10602番地

構造体(値型)

```
struct Test
{
    public int HP;
    public Test(int HP) { this.HP = HP; }
}

void Start()
{
    Debug.Log("値型の場合");

    Test a = new Test(100);
    Test b = a;
    Test c = a;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");

    b.HP = 0;
    Debug.Log($"aの値:{a.HP} , bの値:{b.HP} , cの値:{c.HP}");
}
```



[14:05:52] 値型の場合
UnityEngine.Debug:Log (object)



[14:05:52] aの値 : 100 , bの値 : 100 , cの値 : 100
UnityEngine.Debug:Log (object)



[14:05:52] aの値 : 100 , bの値 : 0 , cの値 : 100
UnityEngine.Debug:Log (object)

アドレス(住所)

メモリ

10570番地

10578番地

10586番地

10594番地

10602番地

10610番地

10618番地

10626番地

10634番地

10642番地

struct型 Test 設計図

Test型 a 実体

Test型 b 実体

Test型 c 実体