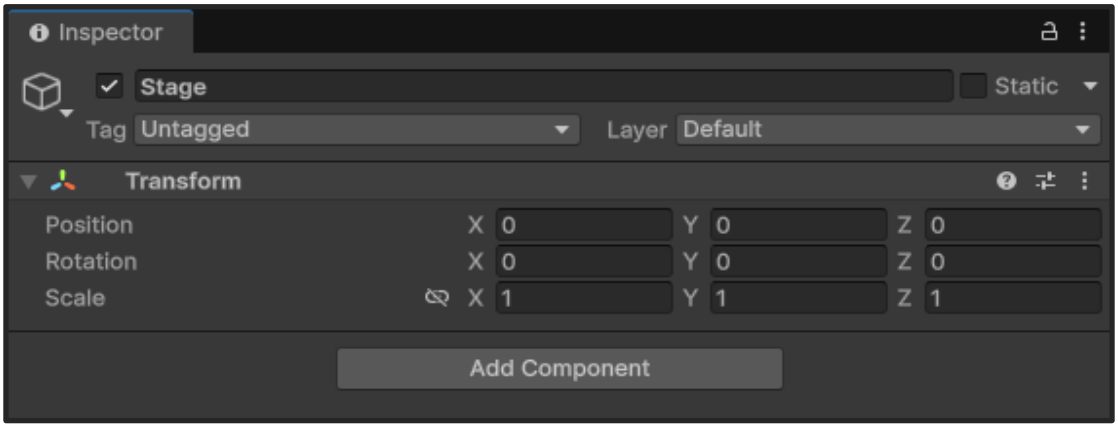
# Transformクラス

モデルを制御するにあたり、
大きさ、回転、位置、モデルのハンドルIDのセット情報が必要になります。
また、モデルにそれらの情報を反映させる、MVISetMatrix関数、
MVISetMatrix関数の引数に必要な合成行列の作成など、
定番的にやることが出てきますので、これを1つのクラスにまとめたのが
Transformクラスです。
設計の由来は、UnityのTransformです。




通常は、メンバ変数をprivateに隠してカプセル化を行いますが、
構造体的な使い方が強いため、特別にpublicに宣言しています。

## Transformの設計

- モデル制御に必要な情報、単純な更新処理をまとめて、
  使用する際に、手間が掛からないように

- 回転は、クォータニオンをベースに処理を行い、
  メンバ変数にある回転行列(matRot)やオイラー角(rot)は、
  クォータニオンから変換したものが格納されている。
  よって、回転行列やオイラー角を更新しても、モデルは回転しない。

## Transformの使い方

Playerクラスなどに実体として宣言する。
※特にロードなどもなく、機能ボリュームも大きくなることはない。
  ほとんどの場合、Playerなどの親を持つことになるので、実体で十分

例：TitleSceneで簡易的にモデルを描画する

手順①　TitleScene.hでTransformクラスをincludeし、
メンバ変数宣言を行う

```
#include "../Object/Common/Transform.h"

・・・・

// 惑星
Transform bigPlanet_;
```

手順②　ResourceManagerで対象モデルのリソースを準備する

リソース名　　：　PIT_FALL_PLANET
リソース種別　：　MODEL
読込先　　：　PATH_MDL +
"Stage/PitfallPlanet/PitfallPlanet.mvl"

手順③　TitleSceneクラスのInit関数で初期化を行う

```
// メイン惑星
bigPlanet_.SetModel(resMng_.LoadModelDuplicate(
    ResourceManager::SRC::PIT_FALL_PLANET));
bigPlanet_.scl = AsoUtility::VECTOR_ONE;
bigPlanet_.quaRot = Quaternion::Identity();
bigPlanet_.quaRotLocal = Quaternion::Identity();
bigPlanet_.pos = AsoUtility::VECTOR_ZERO;
bigPlanet_.Update();
```

手順④　TitleSceneクラスのDraw関数でモデルを描画する

```
// モデル描画
MV1DrawModel(bigPlanet_.modelId);
```

完成図　　正しくモデルが描画されていればＯＫ。


【要件①】

TitleSceneに、Z回転しつづける球体惑星を描画すること。

| | | |
|---|---|---|
| リソース名 | ： | SPHERE_PLANET |
| リソース種別 | ： | MODEL |
| 読込先 | ： | PATH_MDL + "Stage/SpherePlanet/SpherePlanet.mvl" |
| スケール | ： | 0.7 |
| 回転 | ： | X軸90度 |
| 座標 | ： | { -250.0f, -100.0f, -100.0f } |

【目標①】


回転を忘れずに。

【要件②】

TitleSceneの球体惑星上で、歩くキャラクターを描画すること。
但し、Playerクラスなどは作らず、
TransformクラスとAnimationコントローラークラスのみで実装すること。

| | | |
|---|---|---|
| リソース名 | ： | PLAYER |
| リソース種別 | ： | MODEL |
| 読込先 | ： | PATH_MDL + "Player/Player.mvl" |
| | | |
| スケール | ： | 0.4 |
| 回転 | ： | 画面左を向くこと |
| ローカル回転 | ： | Y軸180度 |
| 座標 | ： | { -250.0f, -32.0f, -105.0f } |
| | | |
| アニメーション | ： | Run.mvl |

【目標②】


キャラクターがアニメーションを行っていること。