ゲームのシーンがスクロールを前提としている場合、ゲーム画面がスクロールするが、
UIなど、スクロールさせえたくないものが１つの画面上に混在することはよくあるはずだ。
また、ゲーム画面は高解像度だが、UIにはそこまで解像度は必要としない場合もある。
カメラがNodeに追従するだけなら、Followを使う手段があるが、
単純なNode同士の追従に使う分には簡単だが、シーンに適応した場合は、
そのシーンにぶら下がるあらゆるNodeが影響を受けるため、
その見た目のつじつまを合わせるのに、カメラの座標を与え続けないとダメになるし、
その小数点の誤差で、UIなど固定表示したいものが、1pixel上下する可能性もあるため、
使いづらい。
では、どうするかだが、cocos2d-xにおいて、カメラは複数セットすることが可能だ。
基本的なゲーム画面は標準のカメラで描画し、それ以外は別途追加したカメラで描画を行う。
例えば、ゲームシーンはデフォルトカメラに、UIは追加カメラ1に、メニューは追加カメラ2に…。
といった感じで、用途などに応じてカメラを切り替える。

ということで、カメラの追加の仕方だが、Spriteなどと同様にCameraをクリエイトして、
それをシーンにAddChilde関数を呼び出して追加する。

```
83      auto screenSize = Director::getInstance()->getWinSize();
84      auto camera = Camera::createOrthographic(screenSize.width, screenSize.height, -768, 768);
85      this->addChild(camera);
```

次にカメラの位置、角度、深度を設定する。

```
89      camera->setPosition3D(Vec3(0.0f, 0.0f, 0.0f));
90      camera->setRotation3D(Vec3(0.0f, 0.0f, 0.0f));
91      camera->setDepth(0.0f);
```

また、カメラのマスク用のフラグをセットする。

```
87      camera->setCameraFlag(CameraFlag::USER1);
```

なお、CameraFlagは標準のDEFAULTをはじめとし、USER1～USER8まで、存在する。

そのフラグを、シーンもしくはlayer、Spriteなど影響を与えたいNodeにもセットする。
そのカメラのレンダリングターゲットとなる。
複数のカメラでレンダリングしたい場合は、ビットをOR演算することで、対応が可能だ。
このフラグは、カメラに設定されているフラグと同一のフラグが立っている場合、
Nodeのツリー構造で下にぶら下がっているNodeには影響がある。
なので、layerなどに立てておくと管理が楽でよい。

```
172     auto uiLayer = Layer::create();
173     this->addChild(uiLayer, 1);
174     uiLayer->setName("UI_LAYER");
175
176     uiLayer->setCameraMask(static_cast<int>(CameraFlag::USER1));
```

この場合は、UI用のlayerをUSER1に設定している形になっている。
複数設定する場合は、CameraFlagをint型にキャストし、ORをとった上で、引数としてセットしよう。

次に、画面のトランジション処理について。
まず、最初にトランジション処理をする際、例えばフェードアウト→フェードインさせる場合であれば、
TransitionFadeを使用する。
GameSceneというシーンを切り替えたいとして、1.5秒でホワイトアウト→ホワイトインを実行の場合、

```
69      auto gameScene = GameScene::createScene();
70      TransitionFade* fade = TransitionFade::create(1.5, gameScene, Color3B::WHITE);
71      Director::getInstance()->replaceScene(fade);
```

このようなコードを実行する。
そのうえで、理解しておいた方がいいポイントがある。
トランジション処理を行う場合、移行前のシーンと移行後のシーンが存在するが、
トランジション処理中はどのシーンが実行中なのだろうか？
ホワイトアウト中は移行元のシーンで、ホワイトイン中は移行先のシーン？
実際はそうではない。
上記コードで、replaceScene関数が実行されている。
この関数は現在のシーンから引数のシーンに移行する。
つまり、ここでは引数であるfadeというシーンに移行していることになる。
TransitionFadeの基底クラスはTransitionSceneとなっており、さらに遡るとSceneクラスが登場する。
フェード機能を持ったNodeぐらいに考えていたかもしれないが、その実態はシーンである。
TransitionSceneの機能としては、内部的に移行元のシーンをout、移行先をinとして管理している。
Transitionでアウト中であれば、outのシーンの描画とトランジションの描画を行い、
Transitionでイン中であれば、inのシーンの描画とトランジションの描画を行う。
そして、Transitionの処理が完了したら、inで設定されているシーンのrunが実行される。
ここで重要なポイントが1点ある。
移行元・先において、カメラが追加され、CameraFlagもそれに合わせて設定されている場合だ。
Transitionが実行中のシーンはあくまで、TransitionSceneだ。
なので、カメラに関しても使用されるのは、out/inに設定されているシーンのカメラではない。
kayerなどに設定されているCameraFlagは有効だが、カメラはTransitionSceneというちぐはぐな
状況が生まれる。
さらに良くないことに、TransitionSceneにはデフォルトのカメラしかない。
なので、CameraFlagでdefault以外が設定されていた場合、そのままでは描画されない。
どうすればいいかというと、シンプルにTransitionSceneにカメラを追加すればよい。
手順は通常のシーンにカメラを追加するときと同じでよい。
outおよびinに設定されるシーンで使っているカメラのフラグ分、同じフラグで作っておこう。
そうすることで、複数カメラを持つシーンであってもTransitionを問題なく動作させることが可能だ。