Systems

What：

将组件数据从当前状态转换为下一个状态的逻辑。

System types：

SystemBase

EntityCommandBufferSystem

ComponentSystemGroup

GameObjectConversionSystem

The ComponentSystem and JobComponentSystem classes, along with IJobForEach, are being phased out of the DOTS API。

How:

ECS通过World和Group组织System。默认情况下，ECS使用预定义的group创建默认Wrold。它查找所有可用的系统，实例化它们，并将它们添加到默认的Wrold中的simulation group中。

可以指定同一组中System的更新顺序。Group是一种System，因此您可以将Group添加到另一个Group，并像其他系统一样指定其顺序。Group中的所有System在下一个Group或System之前更新。

如果不指定顺序，ECS会以不依赖于创建顺序的确定方式将系统插入更新顺序。换句话说，即使没有显式指定顺序，同一组系统在其组内也始终以相同的顺序更新。

Unity ECS会自动发现项目中的系统类，并在运行时实例化它们。

可以使用系统属性指定系统的父组以及该组中该系统的顺序。

系统的更新循环由其父组件SystemGroup驱动。Group是可以嵌套的。

Create：

实习抽象类SystemBase

实现每帧调用的OnUpdate方法

OnCreate() OnStartRunning() OnStopRunning() OnDestroy()都是可选的。

Entities.ForEach

How：

要执行作业lambda函数，可以使用schedule（）和ScheduleParallel（）调度作业，也可以使用Run（）立即（在主线程上）执行作业。

Entities.ForEach((ref Translation translation,in Velocity velocity) =>{

translation.Value += velocity.Value;}).Schedule();

您还可以使用WithAll、WithAny和WithNone子句进一步优化选择的实体。

Entities.WithAll<LocalToWorld>().WithAny<Rotation, Translation, Scale>().WithNone<LocalToParent>()

.ForEach((ref Destination outputData, in Source inputData) =>{}).Schedule();

要访问Entities.ForEach创建的EntityQuery对象，请使用带有ref参数修饰符的[WithStoreEntityQueryInField（ref query）]。

Entities.WithStoreEntityQueryInField(ref query).ForEach((int entityInQueryIndex, in Data data) =>

使用Change filtering

Entities.WithChangeFilter<Source>().ForEach((ref Destination outputData,in Source inputData) =>

使用Shared component filtering

Entities.WithSharedComponentFilter(cohort).ForEach((ref DisplayColor color) => { color = newColor; }).ScheduleParallel();

对于动态缓冲区，使用dynamic buffer<T>而不是存储在缓冲区中的组件类型（待续，有例子）

Special, named parameters：

Entity entity — 当前实体的实体实例。

int entityInQueryIndex — 查询选择的所有实体列表中实体的索引。

int nativeThreadIndex — 执行lambda函数当前迭代的线程的唯一索引。

捕捉的变量的限制：

只能捕获native containers和blittable类型。

作业只能写入作为native containers的捕获变量。

Job.WithCode

What：

Job.WithCode将函数作为单个后台作业运行。可以在主线程上运行Job.WithCode，并且仍然可以利用Burst编译来加快执行速度。

Job.WithCode(() =>{for (int i = 0; i < randomNumbers.Length; i++){

randomNumbers[i] = randomGen.NextFloat();}}).Schedule();

可以使用WithReadOnly指定不更新容器，使用WithDeallocateOnJobCompletion在作业完成后自动释放容器。（Entities.ForEach提供相同的功能。）

有两个选项可以执行lambda函数:Schedule()和Run()

IJobChunk jobs

How：

要实现IJobChunk作业，请使用以下步骤：

创建EntityQuery以标识要处理的实体。

定义Job结构，并包含ArchetypeChunkComponentType对象的字段，这些字段标识Job必须直接访问的组件类型。指定Job是否读取或写入这些组件。

实例化Job并在system OnUpdate（）函数中调度Job。

在Execute（）函数中，获取Job读取或写入的组件的NativeArray实例，然后遍历当前块以执行所需的工作。

[BurstCompile]

struct RotationSpeedJob : IJobChunk

{

public float DeltaTime;

public ArchetypeChunkComponentType<Rotation> RotationType;

[ReadOnly] public ArchetypeChunkComponentType<RotationSpeed> RotationSpeedType;

public void Execute(ArchetypeChunk chunk, int chunkIndex, int firstEntityIndex)

{

var chunkRotations = chunk.GetNativeArray(RotationType);

var chunkRotationSpeeds = chunk.GetNativeArray(RotationSpeedType);

for (var i = 0; i < chunk.Count; i++)

{

var rotation = chunkRotations[i];

var rotationSpeed = chunkRotationSpeeds[i];

if (chunk.Has<OptionalComp>(OptionalCompType))

{//...}

}

}

}

跳过未更改实体的块：

1，m\_Query.SetChangedVersionFilter（new ComponentType[]{

ComponentType.ReadWrite<InputA>(),ComponentType.ReadWrite<InputB>()});

2， public ArchetypeChunkComponentType<InputA> InputAType;

var inputAChanged = chunk.DidChange(InputAType, LastSystemVersion);

Instantiate and schedule the job：

protected override void OnUpdate(){

var job = new RotationSpeedJob()

{

RotationType = GetArchetypeChunkComponentType<Rotation>(false),

RotationSpeedType = GetArchetypeChunkComponentType<RotationSpeed>(true),

DeltaTime = Time.DeltaTime

};

this.Dependency = job.ScheduleParallel(m\_Query, this.Dependency);

}

Manual iteration（待续）

System Update Order

How：

可以使用System的类声明上的[UpdateInGroup]属性将系统放置在组中。

然后可以使用[UpdateBefore]和[UpdateAfter]属性指定组内的更新顺序。

[DisableAutoCreation]-防止在default world初始化期间创建系统。

Default World包含ComponentSystemGroup实例的层次结构。只有三个根级别的系统组被添加到Unity player循环中:

InitializationSystemGroup

SimulationSystemGroup

PresentationSystemGroup

你可以创建多个Worlds