# SpatialOS学习笔记

|  |  |  |  |
| --- | --- | --- | --- |
| 版本号 | 修改日期 | 修改人员 | 修改内容 |
| 0.5 | 2019/7/25 | 刘刚 |  |
|  |  |  |  |

## 创建schema

Schema是专门用于创建component的。Component是ECS架构里的C。

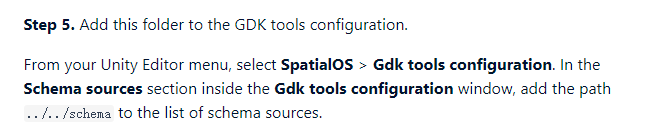
原帖：

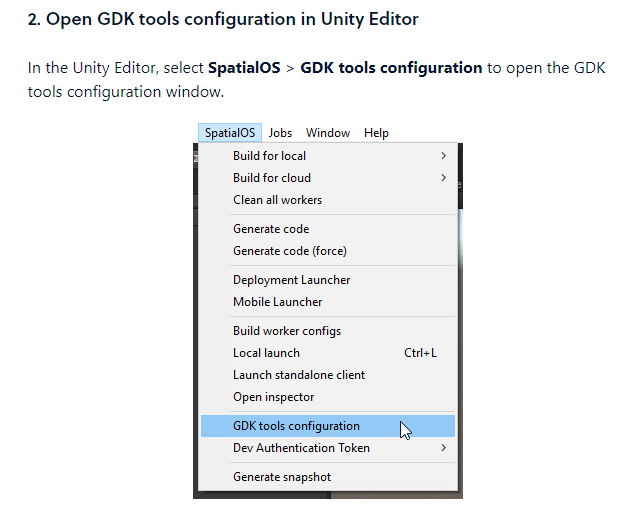
<https://docs.improbable.io/unity/alpha/projects/fps/tutorial>

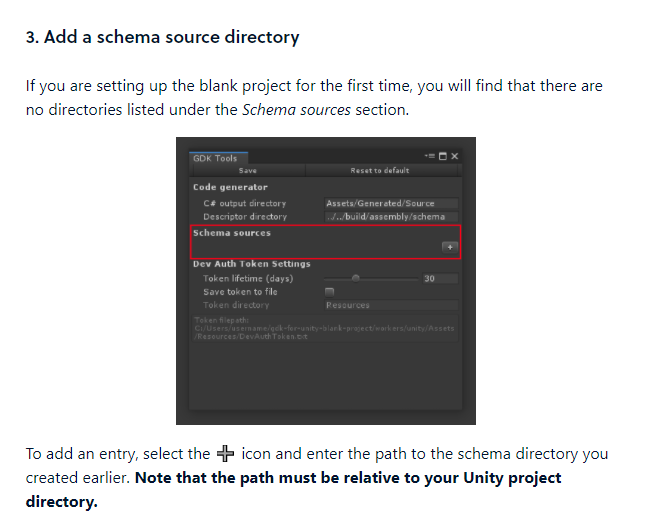
<https://docs.improbable.io/unity/alpha/projects/blank/setup/schema>

在工程根目录创建schema目录，可以再建立子目录，然后把你要创建的schema脚本放进去。

注意别忘了，要在你的GDK Tools Configration里添加schema的路径。

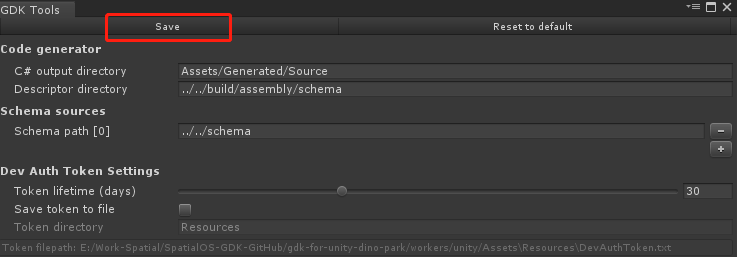








最后别忘了点击“Save”:



## FPS案例-添加Entity

参考案例是gdk-for-unity-fps-starter-project工程。

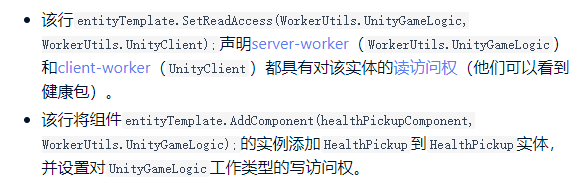
<https://docs.improbable.io/unity/alpha/projects/fps/tutorial>

在这个fps的游戏中添加健康道具。

#### 创建entity

##### FpsEntityTemplates.cs

public static EntityTemplate HealthPickup(Vector3f position, uint healthValue)  
{  
 // Create a HealthPickup component snapshot which is initially active and grants "heathValue" on pickup.  
 var healthPickupComponent = new Pickups.HealthPickup.Snapshot(true, healthValue);  
  
 var entityTemplate = new EntityTemplate();  
 entityTemplate.AddComponent(new Position.Snapshot(new Coordinates(position.X, position.Y, position.Z)), WorkerUtils.**UnityGameLogic**);  
 entityTemplate.AddComponent(new Metadata.Snapshot("HealthPickup"), WorkerUtils.**UnityGameLogic**);  
 entityTemplate.AddComponent(new Persistence.Snapshot(), WorkerUtils.**UnityGameLogic**);  
 entityTemplate.AddComponent(healthPickupComponent, WorkerUtils.**UnityGameLogic**);  
 entityTemplate.SetReadAccess(WorkerUtils.**UnityGameLogic**, WorkerUtils.**UnityClient**);  
 entityTemplate.SetComponentWriteAccess(EntityAcl.**ComponentId**, WorkerUtils.**UnityGameLogic**);  
  
 return entityTemplate;  
}





#### 创建快照

##### SnapshotMenu.cs

private static Snapshot GenerateDefaultSnapshot()  
{  
 var snapshot = new Snapshot();  
 snapshot.AddEntity(FpsEntityTemplates.Spawner(Coordinates.Zero));  
 AddHealthPacks(snapshot);  
 return snapshot;  
}  
  
private static Snapshot GenerateSessionSnapshot()  
{  
 var snapshot = new Snapshot();  
 snapshot.AddEntity(FpsEntityTemplates.Spawner(Coordinates.Zero));  
 snapshot.AddEntity(FpsEntityTemplates.DeploymentState());  
 AddHealthPacks(snapshot);  
 return snapshot;  
}

private static void AddHealthPacks(Snapshot snapshot)  
{  
 // Invoke our static function to create an entity template of our health pack with 100 heath.  
 var healthPack = FpsEntityTemplates.HealthPickup(new Vector3f(5, 0, 0), 100);  
  
 // Add the entity template to the snapshot.  
 snapshot.AddEntity(healthPack);  
 Debug.Log("Add Health Packs!");  
}

#### 客户端显示实体

##### HealthPickupClientVisibility.cs

using Improbable.Gdk.Subscriptions;

using Pickups;

using UnityEngine;

namespace Fps

{

[WorkerType(WorkerUtils.UnityClient)]

public class HealthPickupClientVisibility : MonoBehaviour

{

[Require] private HealthPickupReader healthPickupReader;

private MeshRenderer cubeMeshRenderer;

private void OnEnable()

{

cubeMeshRenderer = GetComponentInChildren<MeshRenderer>();

healthPickupReader.OnUpdate += OnHealthPickupComponentUpdated;

UpdateVisibility();

}

private void UpdateVisibility()

{

cubeMeshRenderer.enabled = healthPickupReader.Data.IsActive;

}

private void OnHealthPickupComponentUpdated(HealthPickup.Update update)

{

UpdateVisibility();

}

}

}

#### 服务器实体实现

##### HealthPickupServerBehaviour.cs

using System.Collections;

using Improbable.Gdk.Core;

using Improbable.Gdk.Health;

using Improbable.Gdk.Subscriptions;

using Pickups;

using UnityEngine;

namespace Fps

{

[WorkerType(WorkerUtils.UnityGameLogic)]

public class HealthPickupServerBehaviour : MonoBehaviour

{

[Require] private HealthPickupWriter healthPickupWriter;

[Require] private HealthComponentCommandSender healthCommandRequestSender;

private Coroutine respawnCoroutine;

private void OnEnable()

{

// If the pickup is inactive on initial checkout - turn off collisions and start the respawning process.

if (!healthPickupWriter.Data.IsActive)

{

respawnCoroutine = StartCoroutine(RespawnHealthPackRoutine());

}

}

private void OnDisable()

{

if (respawnCoroutine != null)

{

StopCoroutine(respawnCoroutine);

}

}

private void OnTriggerEnter(Collider other)

{

// OnTriggerEnter is fired regardless of whether the MonoBehaviour is enabled/disabled.

if (healthPickupWriter == null)

{

return;

}

if (!other.CompareTag("Player"))

{

return;

}

HandleCollisionWithPlayer(other.gameObject);

}

private void SetIsActive(bool isActive)

{

healthPickupWriter?.SendUpdate(new HealthPickup.Update

{

IsActive = new Option<bool>(isActive)

});

}

private void HandleCollisionWithPlayer(GameObject player)

{

var playerSpatialOsComponent = player.GetComponent<LinkedEntityComponent>();

if (playerSpatialOsComponent == null)

{

return;

}

healthCommandRequestSender.SendModifyHealthCommand(playerSpatialOsComponent.EntityId, new HealthModifier

{

Amount = healthPickupWriter.Data.HealthValue

});

// Toggle health pack to its "consumed" state

SetIsActive(false);

// Begin cool-down period before re-activating health pack

respawnCoroutine = StartCoroutine(RespawnHealthPackRoutine());

}

private IEnumerator RespawnHealthPackRoutine()

{

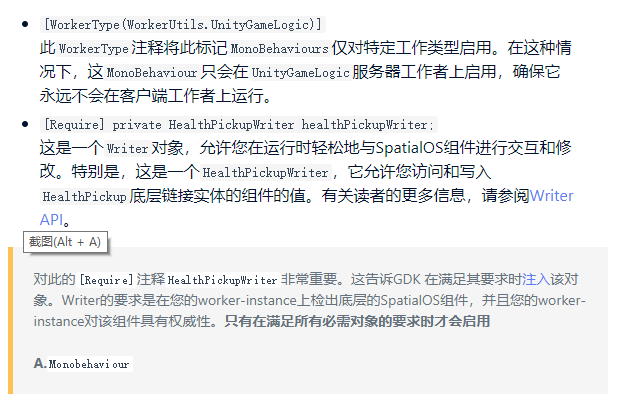
yield return new WaitForSeconds(15f);

SetIsActive(true);

}

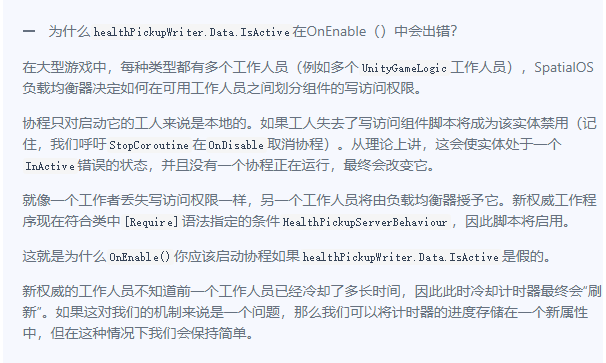
}

}









## 空白案例-添加Player

参考案例是gdk-for-unity-blank-project

<https://forums.improbable.io/t/spatialos-gdk-for-unity-tutorial-series-with-infallible-code/5942>

在一个空白工程内部，添加会移动的玩家角色。

原视频（要翻墙）：

<https://www.youtube.com/playlist?list=PLKERDLXpXl_j5olRrAEls-jdRvgOpNEn4>

#### 创建玩家

##### UnityClientConnector.cs

protected override void HandleWorkerConnectionEstablished()  
{  
 PlayerLifecycleHelper.AddClientSystems(Worker.World);  
   
 var fallbackCreator = new GameObjectCreatorFromMetadata(Worker.WorkerType, Worker.Origin, Worker.LogDispatcher);  
 var customCreator = new PlayerGameObjectCreator(fallbackCreator, Worker.World, Worker.WorkerType);  
   
 GameObjectCreationHelper.EnableStandardGameObjectCreation(Worker.World, customCreator);  
}

##### UnityGameLogicConnector.cs

protected override void HandleWorkerConnectionEstablished()  
{  
 Worker.World.GetOrCreateSystem<MetricSendSystem>();  
 PlayerLifecycleHelper.AddServerSystems(Worker.World);  
 GameObjectCreationHelper.EnableStandardGameObjectCreation(Worker.World);  
}

#### 挂接预制件

##### PlayerGameObjectCreator.cs

public void OnEntityCreated(SpatialOSEntity entity, EntityGameObjectLinker linker)  
{  
 if (!entity.HasComponent<Metadata.Component>()) return;  
  
 var metadata = entity.GetComponent<Metadata.Component>();  
 var isPlayer = metadata.EntityType == "Player";  
 var hasAuthority = PlayerLifecycleHelper.IsOwningWorker(entity.SpatialOSEntityId, \_world);  
 if (isPlayer && hasAuthority)  
 {  
 var pathPrefab = $"Prefabs/{\_WorkerType}/Authoritative/Player";  
 var prefab = Resources.Load(pathPrefab);  
 var playerGameObject = UnityEngine.Object.Instantiate(prefab);  
 linker.LinkGameObjectToSpatialOSEntity(entity.SpatialOSEntityId, (GameObject)playerGameObject);  
 }  
 else  
 {  
 \_fallbackCreator.OnEntityCreated(entity, linker);  
 }  
}

##### UnityGameLogicConnector.cs

private static EntityTemplate CreatePlayerEntityTemplate(string workerId, byte[] serializedArguments)  
{  
 var clientAttribute = EntityTemplate.GetWorkerAccessAttribute(workerId);  
 var serverAttribute = **WorkerType**;  
  
 var template = new EntityTemplate();  
 template.AddComponent(new Position.Snapshot(), clientAttribute);  
 template.AddComponent(new Metadata.Snapshot("Player"), serverAttribute);  
 template.AddComponent(new PlayerTransform.Snapshot(), clientAttribute);  
 TransformSynchronizationHelper.AddTransformSynchronizationComponents(template, clientAttribute);  
 PlayerLifecycleHelper.AddPlayerLifecycleComponents(template, workerId, serverAttribute);  
  
 template.SetReadAccess(UnityClientConnector.**WorkerType**, MobileClientWorkerConnector.**WorkerType**, serverAttribute);  
 template.SetComponentWriteAccess(EntityAcl.**ComponentId**, serverAttribute);  
  
 return template;  
}

到了这时候，一个人移动，另外一个人也会跟着动

#### 确定控制权限

##### WritePlayerTransform.cs

using System.Collections;  
using System.Collections.Generic;  
using Com.Infalliblecode;  
using Improbable;  
using Improbable.Gdk.Subscriptions;  
using UnityEngine;  
  
public class WritePlayerTransform : MonoBehaviour  
{  
 [Require] private PlayerTransformWriter \_writer;  
 // Start is called before the first frame update  
 void Start()  
 {  
   
 }  
  
 // Update is called once per frame  
 void Update()  
 {  
 var update = new PlayerTransform.Update()  
 {  
 Position = Vector3f.FromUnityVector(transform.position),  
 Rotation = Vector3f.FromUnityVector(transform.eulerAngles)  
 };  
 \_writer.SendUpdate(update);  
 }  
}

##### ReadPlayerTransform.cs

using System.Collections;

using System.Collections.Generic;

using Com.Infalliblecode;

using Improbable;

using Improbable.Gdk.Subscriptions;

using UnityEngine;

public class ReadPlayerTransform : MonoBehaviour

{

[Require] private PlayerTransformReader \_reader;

// Start is called before the first frame update

void Start()

{

}

// Update is called once per frame

void Update()

{

transform.position = \_reader.Data.Position.ToUnityVector();

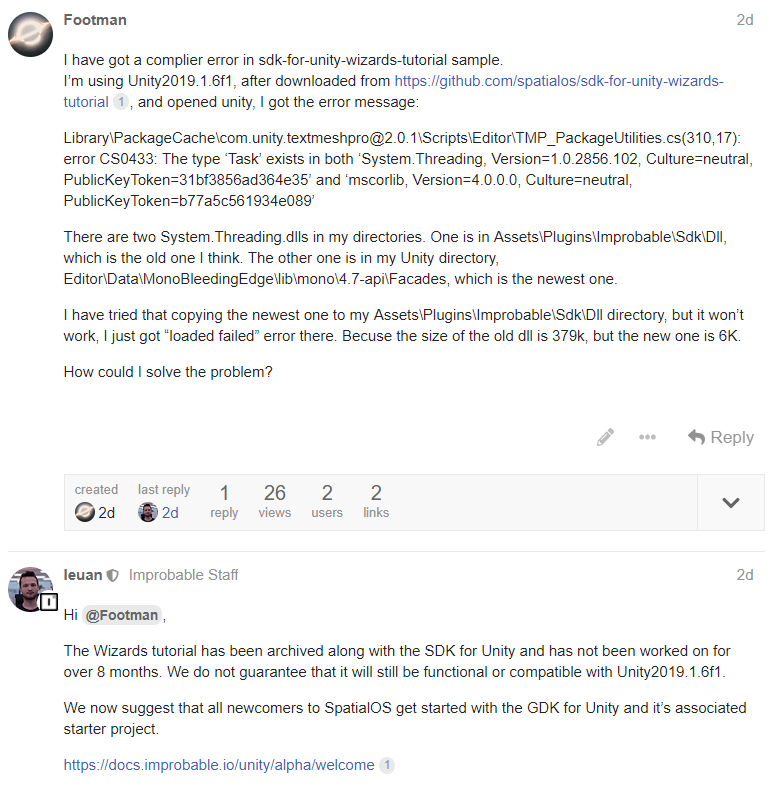
transform.rotation = Quaternion.Euler(\_reader.Data.Rotation.ToUnityVector());

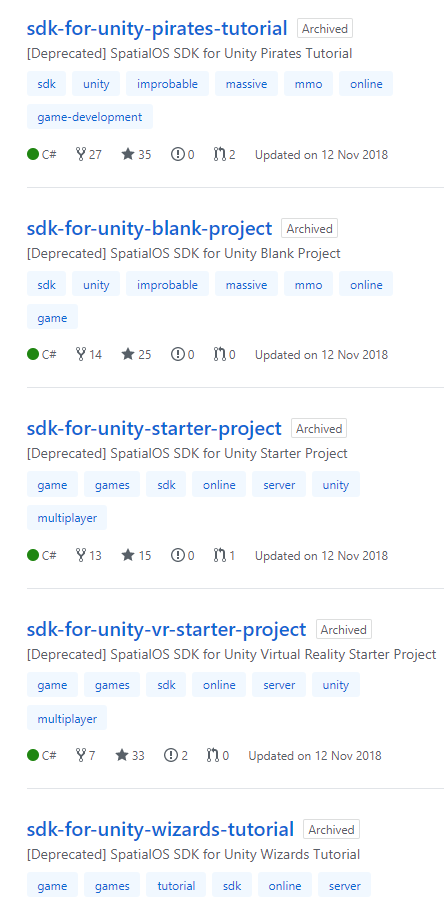
}

}

## Wizards例程的修复（失败）

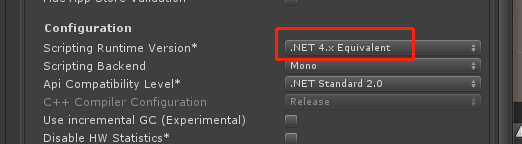
### 向管理员询问





### 尝试自主修改

#### 更新DOTNET到4.0



更新到.net4.x版本。

#### System.Threading.dll冲突

Library\PackageCache\com.unity.textmeshpro@2.0.1\Scripts\Editor\TMP\_PackageUtilities.cs(310,17): error CS0433: The type 'Task' exists in both 'System.Threading, Version=1.0.2856.102, Culture=neutral, PublicKeyToken=31bf3856ad364e35' and 'mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089'

删除Assets\Plugins\Improbable\Sdk\Dll目录下的System.Threading.dll文件。

#### 其他修改

##### 删除Assets/Plugins/Improbable目录

这里是旧的引擎库。

拷贝新的manifest.json到目录：workers\unity\Packages

##### 旧库缺少

using Improbable.Unity;  
using Improbable.Unity.Visualizer;

##### Schema ID重复了

修改：

10001 UnityWorkerAuthorityCheck

10002 Transform

10003 Heartbeat

10004 Inventory

10005 TeamAssignment

### 事后分析

* Unity工程的重要性在降低，5个与Unity有关的例程都已经不维护了。
* 库文件被大量重构过，Wizards工程与Blank工程和Fps工程均有很大的不同。Blank和Fps相似度更搞一些。(Coordinates类看来已经被废弃)
* 对SpatialOS还不够熟悉。还无法进行更加细致的改造。
* **正在研究中。。。**

## ECS

<https://improbable.io/blog/unity-ecs-1>

### Schema语法

<https://docs.improbable.io/reference/13.8/shared/schema/reference>

| **Syntax** | **Type** | **Notes** |
| --- | --- | --- |
| bool | Boolean | True or false. |
| uint32, uint64 | Unsigned integer | Variable-length encoding; smaller values use fewer bits. |
| int32, int64 | Signed integer | Variable-length encoding; smaller values use fewer bits. Negative values are represented in the usual two’s-complement manner, and so use the maximum number of bits. |
| sint32, sint64 | Zig-zag signed integer | Variable-length zig-zag encoding; smaller *absolute* values use fewer bits. More space-efficient than int32 or int64 when values are likely to be negative. |
| fixed32, fixed64, sfixed32, sfixed64 | Fixed-width integer | Fixed-width encoding (always 4 or 8 bytes depending on type); more space-efficient when values are likely to be very large. |
| float, double | Floating-point |  |
| string, bytes | String of characters or bytes | Strings should always be either ASCII or UTF-8. |
| EntityId | ID of an entity | A special version of int64 used to store the ID of a SpatialOS entity. IDs are > 0. |
| Entity | A component set | A data structure that represents an arbitrary collection of components. |
| improbable.Coordinates, improbable.Vector3d, improbable.Vector3f |  | Coordinates represents positions in space, while the other two represent vectors such as velocity. Both Coordinates and Vector3d are 3D vectors of doubles, representing absolute positions and differences between positions in 3D space, respectively. Vector3f is a 3D vector of floats.   To use these types, include import "improbable/vector3.schema"; for Vector3f and Vector3d, and import "improbable/standard\_library.schema"for Coordinates, in your schema file. |

## 案例制作

### 玩家的创建

玩家的创建，感觉上比较复杂，貌似有两个地方：

#### 在Snapshot里的创建

为什么需要在Snapshot里创建？感觉是因为这个数据，需要传递给服务器，在服务器那边创建同类物体的时候会用到。

SnapshotUtil.cs

public static void AddPlayerSpawner(Snapshot snapshot)  
{  
 var entity = EntityTemplateFactory.CreatePlayerTemplate(snapshot);  
 snapshot.AddEntity(entity);  
}

EntityTemplateFactory.cs

public static EntityTemplate CreatePlayerTemplate(Snapshot snapshot)  
{  
 var serverAttribute = UnityGameLogicConnector.**WorkerType**;  
  
 var template = new EntityTemplate();  
 template.AddComponent(new Position.Snapshot(), serverAttribute①);  
 template.AddComponent(new Metadata.Snapshot { EntityType = "PlayerCreator②" }, serverAttribute);  
 template.AddComponent(new Persistence.Snapshot(), serverAttribute);  
 template.AddComponent(new PlayerCreator.Snapshot(), serverAttribute);  
  
 template.SetReadAccess(UnityClientConnector.**WorkerType**, UnityGameLogicConnector.**WorkerType**, MobileClientWorkerConnector.**WorkerType**);  
 template.SetComponentWriteAccess(EntityAcl.**ComponentId**, serverAttribute);  
  
 return template;  
}

#### 客户端连接到服务器以后的创建

服务器端在使用的时候，就是这里：UnityGameLogicConnector.cs

private static EntityTemplate CreatePlayerEntityTemplate(string workerId, byte[] serializedArguments)  
{  
 var clientAttribute = EntityTemplate.GetWorkerAccessAttribute(workerId);  
 var serverAttribute = WorkerType;  
  
 var template = new EntityTemplate();  
 template.AddComponent(new Position.Snapshot(), clientAttribute①);  
 template.AddComponent(new Metadata.Snapshot("Player②"), serverAttribute);  
 template.AddComponent(new PlayerTransform.Snapshot(), clientAttribute①);  
 TransformSynchronizationHelper.AddTransformSynchronizationComponents(template, clientAttribute);  
 PlayerLifecycleHelper.AddPlayerLifecycleComponents(template, workerId, serverAttribute);  
  
 template.SetReadAccess(UnityClientConnector.WorkerType, MobileClientWorkerConnector.WorkerType, serverAttribute);  
 template.SetComponentWriteAccess(EntityAcl.ComponentId, serverAttribute);  
  
 return template;  
}

***注①：这里的【写入权限】必须是客户端。玩家的坐标属性，由玩家自己创建的那个客户端来控制。***

***注②：entity的名字是不一样的，为什么要这样不是很了解。***

#### 预制件的加载

Unity预制件是在客户端（自己的和别人的）显示自己用的。

加载代码在这里：

UnityClientConnector.cs

protected override void HandleWorkerConnectionEstablished()  
{  
 PlayerLifecycleHelper.AddClientSystems(Worker.World);  
   
 // 创建实体的预制件  
 var fallbackCreator = new GameObjectCreatorFromMetadata(Worker.WorkerType, Worker.Origin, Worker.LogDispatcher);  
 var customCreator = new EntityGameObjectCreator(fallbackCreator, Worker.World, Worker.WorkerType);  
 Debug.Log("HandleWorkerConnectionEstablished!");  
   
 GameObjectCreationHelper.EnableStandardGameObjectCreation(Worker.World, customCreator);  
}

EntityGameObjectCreator.cs

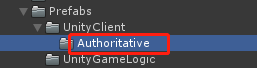
public void OnEntityCreated(SpatialOSEntity entity, EntityGameObjectLinker linker)  
{  
 if (!entity.HasComponent<Metadata.Component>()) return;  
  
 var metadata = entity.GetComponent<Metadata.Component>();  
 var isPlayer = metadata.EntityType == "Player";// 玩家  
 var isTree = metadata.EntityType == SimulationSettings.TreePrefabName;// 树  
 var isLumberJack = metadata.EntityType == SimulationSettings.NPCPrefabName; // 伐木工  
 var isDino = metadata.EntityType == SimulationSettings.Dino\_Brachio\_PrefabName;// Dino Brachiosaurus  
 var hasAuthority = PlayerLifecycleHelper.IsOwningWorker(entity.SpatialOSEntityId, \_world);  
 if (isPlayer && hasAuthority①)  
 {  
 var pathPrefab = $"Prefabs/{\_WorkerType}/Authoritative/Player";  
 var prefab = Resources.Load(pathPrefab);  
 var playerGameObject = UnityEngine.Object.Instantiate(prefab);  
 linker.LinkGameObjectToSpatialOSEntity(entity.SpatialOSEntityId, (GameObject)playerGameObject);  
 Debug.Log("EntityGameObjectCreator OnEntityCreated - A Player GameObject created!");  
 }

else  
 {  
 \_fallbackCreator②.OnEntityCreated(entity, linker);  
 }  
}

1. ：这里【hasAuthority】可以识别当前的客户端是不是玩家自己的客户端，还是其他玩家的客户端。加载的位置和其他玩家的是不一样的。其他玩家加载的位置在Prefabs/UnityClient/根目录下。
2. ：其他情况会运行【\_fallbackCreator】，但是这里是如何加载的，因为包含在引擎内部，尚不了解。基本上就是按照[下面的规定](#_预制件的挂接)来加载。

#### 预制件的挂接

##### 玩家自己客户端

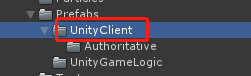
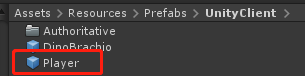
 



这里要挂接写操作脚本。

*注：真正的游戏里，一般不这样写。一般仅把客户端的移动指令发送到服务器端，由服务器端负责寻路，并把路线传回来。除了FPS游戏，一般游戏很少实时更新坐标的。*

##### 其他玩家客户端



这里挂接读取操作脚本。

##### 服务器

实际上，因为是一个玩家发送，其他玩家接收，所以不需要服务器客户端的存在。除非你在服务器端需要对这个客户端还有其他的操作。以后再添加。

*注：这里我曾经尝试过添加，但是有报错误，没有继续尝试。*

#### 控制权的定义

##### 首先要在EntityTemplate中定义好控制权。

UnityGameLogicConnector.cs

template.AddComponent(new Position.Snapshot(), clientAttribute);

##### 然后在不同的地方挂接写入和读取的脚本。





#### 接收和发送脚本

##### 发送

WritePlayerTransform.cs

public class WritePlayerTransform : MonoBehaviour  
{  
 [Require] private PlayerTransformWriter \_writer;  
 [Require] private PositionWriter spatialPosition;  
  
 // Update is called once per frame  
 void Update()  
 {  
 var update = new Position.Update()  
 {  
 Coords = transform.position.ToCoordinates()  
 };  
 spatialPosition.SendUpdate(update);  
 var update2 = new PlayerTransform.Update()  
 {  
 //Position = Vector3f.FromUnityVector(transform.position),  
 Rotation = Vector3f.FromUnityVector(transform.eulerAngles)  
 };  
 \_writer.SendUpdate(update2);  
 }  
}

* 这里用到两个组件，一个是Position，一个是PlayerTransform。
* 这两个组件都是SpatialOS系统自己创建的。可以直接使用（不需要自己创建schema）。
* 其中，Position里只包含了坐标，没有旋转。但是PlayerTransform里都有。
* ***但是，只有Position的组件才会影响到【网页】里，我们可以从网页里看到坐标位置的变化。***

##### 接收

ReadPlayerTransform.cs

public class ReadPlayerTransform : MonoBehaviour  
{  
 [Require] private PlayerTransformReader \_reader;  
 [Require] private PositionReader spatialPosition;  
  
 // Update is called once per frame  
 void Update()  
 {  
 transform.position = spatialPosition.Data.Coords.ToUnityVector();  
 //transform.position = \_reader.Data.Position.ToUnityVector();  
 transform.rotation = Quaternion.Euler(\_reader.Data.Rotation.ToUnityVector());  
 }  
}

### NPC的创建

#### 在Snapshot里的创建

NPC的创建就只有一个地方了，与Player不同。

SnapshotUtil.cs

public static void AddNPCsAroundHQs(Snapshot snapshot, Coordinates[] locations)  
{  
 for (uint **teamId** = 0; **teamId** < locations.Length; **teamId**++)  
 {  
 SpawnNpcsAroundPosition(snapshot, locations[**teamId**], **teamId**);  
 }  
}

public static void SpawnNpcsAroundPosition(Snapshot snapshot, Coordinates position, uint team)  
{  
 float totalNpcs = SimulationSettings.HQStartingWizardsCount + SimulationSettings.HQStartingLumberjacksCount;  
 float radiusFromHQ = SimulationSettings.NPCSpawnDistanceToHQ;  
  
 for (int **i** = 0; **i** < totalNpcs; **i**++)  
 {  
 float radians = (**i** / totalNpcs) \* 2 \* Mathf.**PI**;  
 Vector3 **offset** = new Vector3(Mathf.Cos(radians), 0, Mathf.Sin(radians));  
 **offset** \*= radiusFromHQ;  
 Coordinates coordinates = (position.ToVector3() + **offset**).ToCoordinates();  
  
 EntityTemplate **entity** = null;  
 if (**i** < SimulationSettings.HQStartingLumberjacksCount)  
 {  
 //entity = EntityTemplateFactory.CreateNPCLumberjackTemplate(coordinates, team);  
 **entity** = EntityTemplateFactory.CreateDinoBrachioTemplate(coordinates, team);  
 }  
 else  
 {  
 //entity = EntityTemplateFactory.CreateNPCWizardTemplate(coordinates, team);  
 }  
  
 if (**entity** != null)  
 {  
 snapshot.AddEntity(**entity**);  
 }  
 }  
 Debug.Log("Snapshot Dinosaurs generated ! count<"+totalNpcs+">");  
}

EntityTemplateFactory.cs

public static EntityTemplate CreateDinoBrachioTemplate(Coordinates initialPosition, uint teamId)  
{  
 var serverAttribute = UnityGameLogicConnector.**WorkerType**;  
 var template = new EntityTemplate();  
 template.AddComponent(new Position①.Snapshot(initialPosition), serverAttribute);  
 template.AddComponent(new Metadata.Snapshot(SimulationSettings.Dino\_Brachio\_PrefabName), serverAttribute);  
 template.AddComponent(new Persistence.Snapshot(), serverAttribute);  
 template.AddComponent(new PlayerTransform.Snapshot(), serverAttribute);  
 template.AddComponent(new Health.Snapshot(SimulationSettings.LumberjackMaxHealth, SimulationSettings.LumberjackMaxHealth, true), serverAttribute);  
 template.AddComponent(new Flammable.Snapshot(false, true, FireEffectType.**SMALL**), serverAttribute);  
 //template.AddComponent(new TargetNavigation.Snapshot(NavigationState.INACTIVE, Vector3f.Zero, new EntityId(), 0f), serverAttribute);  
 template.AddComponent(new Inventory.Snapshot(0), serverAttribute);  
 template.AddComponent(new DinoBrachio②.Snapshot(0), serverAttribute);  
   
 template.SetReadAccess(UnityClientConnector.**WorkerType**, UnityGameLogicConnector.**WorkerType**, MobileClientWorkerConnector.**WorkerType**);  
 template.SetComponentWriteAccess(EntityAcl.**ComponentId**, serverAttribute);  
   
 return template;  
}

1. *：同Player一样，需要注册****坐标****和****旋转****的组件，进行传递。*
2. *：同时，新增【DinoBrachio】这个组件，来实现它的****状态****传递。注意，这里它们的权限都是服务器端的。这里的逻辑是，服务器负责逻辑，然后发送给所有的客户端。*

#### 预制件的加载

这里的方法和【[玩家预制件的加载](#_预制件的加载)】是一样的。

EntityGameObjectCreator.cs

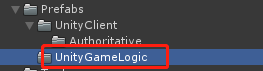
public void OnEntityCreated(SpatialOSEntity entity, EntityGameObjectLinker linker)  
{  
 if (!entity.HasComponent<Metadata.Component>()) return;  
  
 var metadata = entity.GetComponent<Metadata.Component>();  
 var isDino = metadata.EntityType == SimulationSettings.Dino\_Brachio\_PrefabName;// Dino Brachiosaurus  
 if (isDino)  
 {  
 var pathPrefab = $"Prefabs/{\_WorkerType}/" + SimulationSettings.Dino\_Brachio\_PrefabName;  
 Debug.Log("Begin Create a Dinosauer GameObject! - " + pathPrefab);  
 var prefab = Resources.Load(pathPrefab);  
 var entityGameObject = UnityEngine.Object.Instantiate(prefab);  
 entityGameObject.name = SimulationSettings.Dino\_Brachio\_PrefabName + "(EntityID:" + entity.SpatialOSEntityId + ", Worker: " + \_WorkerType + ")";  
 linker.LinkGameObjectToSpatialOSEntity(entity.SpatialOSEntityId, (GameObject)entityGameObject);  
 Debug.Log("EntityGameObjectCreator OnEntityCreated - A Dinosauer Brachiosaurus GameObject created");  
 }  
 else  
 {  
 \_fallbackCreator.OnEntityCreated(entity, linker);  
 }

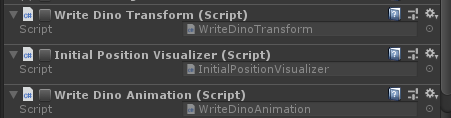
}

***注：这时候不区分自己玩家还是其他玩家了。***

#### 预制件的挂接

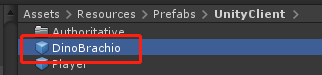
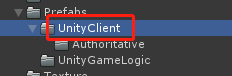
##### 服务器端

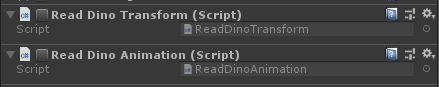




服务器是写入方。控制权在服务器端。

##### 所有玩家客户端





这时候不区分自己玩家还是其他玩家了。所有玩家都是接收方，读取信息。

#### 状态脚本

坐标移动的脚本是WriteDinoTransform.cs/ReadDinoTransform.cs，和Player类似，

##### 写入动画的脚本

WriteDinoAnimation.cs

public class WriteDinoAnimation : MonoBehaviour  
{  
 [Require] private DinoBrachioWriter dinoWriter; // 恐龙的状态  
  
 private Animator animator; // 恐龙的动画，直接修改动画的播放  
 private DinoFSMState.StateEnum \_lastStatus;  
   
 void Awake()  
 {  
 animator = GetComponent<Animator>();  
 animator.applyRootMotion = false;  
 }  
  
 // Update is called once per frame  
 void Update()  
 {  
 var status = GetStatus();  
 if (status == DinoFSMState.StateEnum.**NONE**)  
 return;  
 var update = new DinoBrachio.Update()  
 {  
 CurrentState = status  
 };  
 dinoWriter.SendUpdate(update);  
 }  
  
 DinoFSMState.StateEnum GetStatus()  
 {  
 string[] animationBool = { "isEating", "isWalking", "isRunning", "isAttacking", "isDead"};  
 int **index** = 0;  
 foreach (var ani in animationBool)  
 {  
 var isPlaying = animator.GetBool(ani);  
 if (isPlaying && **index** != (int) \_lastStatus)  
 {  
 return (DinoFSMState.StateEnum)**index**;  
 }  
  
 **index**++;  
 }  
  
 return DinoFSMState.StateEnum.**NONE**;  
 }  
}

##### 读取状态的脚本

ReadDinoAnimation.cs

public class ReadDinoAnimation : MonoBehaviour  
{  
 private Animator animator; // 恐龙的动画，直接修改动画的播放  
 private DinoFSMState.StateEnum \_lastStatus = DinoFSMState.StateEnum.**IDLE**;  
   
 [Require] private DinoBrachioReader dinoReader; // 恐龙的状态  
  
 void Awake()  
 {  
 animator = GetComponent<Animator>();  
 animator.applyRootMotion = false;  
 }  
  
 // Update is called once per frame  
 void Update()  
 {  
 SetStatus(dinoReader.Data.CurrentState);  
 }  
  
 void SetStatus(DinoFSMState.StateEnum inStatus)  
 {  
 if (animator == null)  
 return;  
 if (\_lastStatus == inStatus)  
 return;  
 string[] animationBool = { "isEating", "isWalking", "isRunning", "isAttacking", "isDead"};  
 if (inStatus < 0 || inStatus > DinoFSMState.StateEnum.**ON\_FIRE**)  
 return;  
 animator.SetBool(animationBool[(int)\_lastStatus], false);  
 animator.SetBool(animationBool[(int)inStatus], true);  
 \_lastStatus = inStatus;  
 }  
}