

Итерирование

Чтобы реализовать вашу реальную логику Механики, вам потребуется обработать все Сущности, удовлетворяющие [Фильтру](#). В целях эффективности и последовательности, это делается не напрямую, а через [Цепи](#). Итерируясь по цепям, вы итерируетесь по всем Сущностям и объектам типа Сущностный внутри них.

В итерации по цепям помогают Курсоры. Их семантика напоминает итераторы в стандартных контейнерах. Множественность курсоров поддерживается, но с оглядкой на мультипоточность, которая будет полноценно поддерживаться в недалёком будущем. А сейчас вам достаточно использовать только один простой Курсор.

Работа в C++

Итерирование по цепи сделано через специальный тип объекта *Курсор* (Cursor). Можете использовать столько, сколько захотите, но обычно, достаточно одного:

```
FChain::FCursor Cursor = Chain->Iterate();
```

Если цепь [твёрдотельная](#), то код будет выглядеть так:

```
FSolidChain::FCursor SolidCursor = SolidChain->Iterate();
```

Когда вы получили желанный курсор, вы можете построить простой while-цикл:

```
while (Cursor.Provide())
{
    auto Trait = Cursor.GetTrait<FMyTrait>();
    ...
}
```

Метод [Provide\(\)](#) подготавливает нужное состояние и возвращает `false`, когда закончились слоты в цепи (`true` иначе).

Имея твёрдотельный Курсор вы можете получить прямую ссылку (без копирования) на трейт (используя метод [GetTraitRef\(\)](#)):

```
while (SolidCursor.Provide())
{
```

```
auto& Trait = SolidCursor.GetTraitRef<FMyTrait>();  
...  
}
```

Пожалуйста, заметьте, что цепи утилизируются автоматически, когда все итерируемые курсоры закончили итерироваться по слотам. Чтобы предотвратить такое поведение особо, можете использовать вызовы [Retain\(\) / Release\(\)](#), чтобы самостоятельно контролировать время жизни объектов:

```
Chain->Retain(); // Забрать цепь.  
FChain::FCursor Cursor = Chain->Iterate();  
while (Cursor.Provide())  
{  
    ...  
}  
// Здесь выполняем операции над цепью.  
// Гарантируется, что она не будет удалена.  
...  
Chain->Release(); // Очищаем данные цепи.
```

Встроенные курсоры

Аппарат предоставляет способ итерироваться по цепям встроенными (самостоятельно выделенными) Курсорами. В основном, эта технология используется внутри плагина для корректной работы Blueprint-ов, и вам её стоит избегать в своём C++ коде.

Код будет довольно прост. Он состоит из while-цикла с одним условием:

```
while (Chain.BeginOrAdvance())  
{  
    ...  
}
```

Внутри этого цикла вы можете реализовать нужную логику, используя [Сущности](#) напрямую или служебные методы [Цепей](#):

```
while (Chain.BeginOrAdvance())  
{  
    FSubjectHandle Subject = Chain.GetSubject();  
    UMyDetail* MyPosition = Chain.GetDetail<UMyDetail>();  
    FMyTrait MyVelocity;  
    Chain.GetTrait(MyVelocity);  
    MyPosition->X += MyVelocity.VelocityX * DeltaTime;  
    MyPosition->Y += MyVelocity.VelocityY * DeltaTime;  
    ...
```

```
MyVelocity.VelocityX = 0;  
MyVelocity.VelocityY = 0;  
Subject.SetTrait(MyVelocity);  
}
```

Когда указатель-Курсор Цепи пройдёт последний доступный объект Сущности (или типа Сущностный), Цепь будет уничтожена и ранее заблокированные Чанки и Белты вновь разблокируются, все ожидаемые структурные изменения будут незамедлительно выполнены (если они вообще были).

From:
<http://turbanov.ru/wiki/> - **Turbopedia**



Permanent link:
<http://turbanov.ru/wiki/ru/toolworks/docs/apparatus/iterating?rev=1630526555>

Last update: **2021/09/01 20:02**