Оперирование

Более новый и функциональный способ оперировать над Цепями через процесс, называемый *оперирование* (operating).

Организация в C++

Использование лямбда-выражений

Можно легко оперировать над Цепями через C++ лямбды и вот как следует это делать:

```
Chain->Operate([](FMyTrait Trait)
{
    ...
});
```

Обратите внимание, что вам не позволяется получать ссылку на трейт, если вы итерируетесь по не-твердотельной цепи, - разрешено только копирование. Итак, оперировать над твердотельной Цепью вам следует следующим образом:

```
SolidChain->Operate([](FMyTrait& Trait)
{
    ...
});
```

Теперь можете менять свойства (поля) трейта напрямую, без привлечения копирования.

Параллельность

Твердотельные цепи также поддерживают специальный тип оперирования - мультипоточный. К названию функции, которую надо вызвать, по такому случаю подписали `Concurrently`, и она принимает ещё два аргумента:

- максимальное количество потоков, выделенных на выполнение задачи, и
- минимальное число слотов на каждый поток.

Например:

```
SolidChain->OperateConcurrently([](FMyTrait& Trait)
{
    ...
```

```
}, 4, 32);
```

Второй параметр помогает ограничить количество потоков. Если слишком мало доступных слотов, излишние потоки не понадобятся и они не будут помещены в очередь.

Доставка аргументов

Хорошая особенность оперирования - это то, что аргументы функции решаются и доставляются автоматически в вашу логику. Например, если вы также модифицировали текущую сущность итерирования, то просто укажите хэндлер в самом начале кода:

```
Chain->Operate([](FSubjectHandle Subject, FMyTrait Trait)
{
    ...
});
```

Это, конечно, должно соответствовать твердотельности цепи. Для твердотельной цепи код выглядит так:

```
SolidChain->Operate([](FSolidSubjectHandle Subject, FMyTrait& Trait)
{
    ...
});
```

Вы можете запрашивать разную информацию контекста внутри цикла. Например:

```
Chain->Operate([](const FChain* Chain, const FChainCursor& Cursor,
ISubjective* Subjective, FMyTrait Trait, UMyDetail* Detail)
{
    ...
});
```

Индекс текущей итерации

В случае, если надо получить номер текущего Слота в итерации (т.е. место сущности в цепи), используйте выделенный GetChainSlotIndex() метод соответствующего типа курсора. Курсор можно получить так же, как было представлено в предыдущей секции:

```
SolidChain->Operate([](FPlacementTrait& Placement, const FSolidChainCursor&
Cursor)
{
    Placement.Number = Cursor.GetChainSlotIndex();
});
```

Остановка

Хотя и непонятно, зачем вдруг понадобится останавливать процесс итерирования над цепью вручную, но этого легко достичь при помощи выделенного метода.

Например:

```
int32 Counter = 0;
Chain->Operate([&Counter](const FChain* Chain, FMyTrait Trait)
{
    if (Counter > 100)
    {
        Chain->StopIterating();
    // Досрочно возвращаем управление, чтобы не инкрементировать счётчик:
        return;
    }
    Counter += Trait.Value;
});
```