Turning Runtime Performance Errors into Compiler Errors

KEITH STOCKDALE

RARE LTD

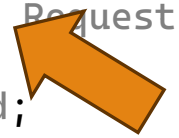
A bug that tests can’t (reasonably) catch

```
struct AsyncLoadRequest
{
    int Id = 0;
    std::vector<std::string> Assets;
};

std::vector<AsyncLoadRequest> LoadRequests;

void OnAssetLoadCompleted(int AssetId)
{
    const auto RangeToBeDeleted = std::ranges::remove_if(LoadRequests,
        [AssetId](const AsyncLoadRequest Request)
        {
            return Request.Id == AssetId;
        });

    LoadRequests.erase(RangeToBeDeleted.begin(), RangeToBeDeleted.end());
}
```



Enter a rather verbose type trait

```
TFuncArgsAreAllRefOrPointerOrSmallTrivial<PredicateType, ElementType>


(
    std::is_rvalue_reference_v<InArgs> ||
    std::is_lvalue_reference_v<InArgs> ||
    std::is_pointer_v<InArgs>
) ||
(
    std::is_trivially_copyable_v<InArgs> &&
    (sizeof(InArgs) <= 16)
)
```

Testing the type trait

```
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(NoArgs)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneTrivialVal)>::Value, "");
static_assert(!TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneNonTrivialVal)>::Value, "");
static_assert(!TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneLargeTrivialVal)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneTrivialRef)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneNonTrivialRef)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneLargeTrivialRef)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneTrivialRef), FTrivialSmallPOD>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneNonTrivialRef), FNotTrivial>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneLargeTrivialRef), FLargePOD>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneTrivialPtr)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneNonTrivialPtr)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneLargeTrivialPtr)>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneAutoParam), FTrivialSmallPOD>::Value, "");
static_assert(!TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneAutoParam), FLargePOD>::Value, "");
static_assert(!TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneAutoParam), FNotTrivial>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneAutoParam), const FTrivialSmallPOD&>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneAutoParam), const FLargePOD&>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneAutoParam), const FNotTrivial&>::Value, "");
static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<decltype(OneConstRefAutoParam), FTrivialSmallPOD>::Value, "");
```

Asserting

```
static_assert(
TFuncArgsAreAllRefOrPointerOrSmallTrivial<PredicateType, ElementType>::Value,

"Trying to use a predicate which takes its arguments by value which may
involve an expensive copy."
"If passing by value is intentional, please pass a default arg of type
FOverrideFuncTestTag as well,"
"(An example and rational for this can be found just above the definition of
FOverrideFuncTestTag in CallableQueries.h)."
"otherwise change your argument to a reference or const reference“
);
```

Applying the static assert everywhere

```
template <typename Predicate>
int32 FindLastByPredicate(Predicate&& Pred, int32 StartIndex) const
{
    static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<Predicate, ElementType>::Value,
template <typename Predicate>
int32 RemoveAll(const Predicate& Pred)
{
    static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<Predicate, ElementType>::Value,
template <class PREDICATE_CLASS>
void Sort(const PREDICATE_CLASS& Predicate)
{
    static_assert(TFuncArgsAreAllRefOrPointerOrSmallTrivial<PREDICATE_CLASS, ElementType,
    ElementType>::Value,
    "Trying to use a predicate which takes its arguments by value which may involve an expensive copy."
    "If passing by value is intentional, please pass a default arg of type FOverrideFuncTestTag as well,"
    "(An example and rational for this can be found just above the definition of FOverrideFuncTestTag in
    CallableQueries.h)."
    "otherwise change your argument to a reference or const reference");
}
```

Take Aways

- Missing ‘&’ can be costly
- C++ type system is powerful
- Write good error messages
- Try it yourself!