A Guide to Assertion Macros in WebKit

May 24, 2010
by Darin Adler
@darinadler

Background

WebKit provides a number of macros to assert that conditions in the code are met. They are defined in `Source/JavaScriptCore/wtf/Assertions.h`. This document provides an overview of the various macros, including guidelines and best practices for their use.

Types of ASSERT macros

The `ASSERT()` macro and its variants are compiled out of release builds. They are meant for use during the development process to catch programming mistakes. For those macros that accept an expression as an argument, the expression is also compiled out of release builds and thus incurs no overhead in shipping code.

- `ASSERT(expression)` – for use during development to ensure that unexpected conditions do not occur. If the expression evaluates to false then abort execution and break into the debugger.
- `ASSERT_NOT_REACHED()` – for use when a certain branch of code should never be executed.

```
if (condition) {
  ...
} else {
        // This should never happen.
        ASSERT_NOT_REACHED();
    }
```

- `ASSERT_UNUSED(variable, expression)` – for assertions that check the value of otherwise unused variable. The need for this becomes apparent if you consider the case where you want to assert an expression that uses a variable that wouldn't otherwise be used in the enclosing function. `ASSERT()` can't be used in this case because in a release build the compiler would warn about the unused variable. `ASSERT_UNUSED()` avoids this warning.
  Example from `Source/JavaScriptCore/jit/ExecutableAllocatorPosix.cpp`:

```
void ExecutablePool::systemRelease(const ExecutableP
{
    int result = munmap(alloc.pages, alloc.size);
    // result not used outside the ASSERT().
    ASSERT_UNUSED(result, !result);
}
```

The CRASH() macro

`CRASH()` raises a fatal error resulting in program termination and triggering either the debugger or the crash reporter. It is active in both debug & release mode. `CRASH()` directly affects users in that it will disrupt or end their browsing session. If a browser vendor traces crashes, it can be extremely helpful in diagnosing hard to find issues that may only occur on users' machines.

Considerations when using ASSERT() and CRASH() macros.

Hazards of using the ASSERT() family of macros

The expression inside the `ASSERT` and `ASSERT_UNUSED` macro is compiled out of release builds together with the macro itself. If the expression that's used has side effects, its omission in release build can lead to programming errors that don't manifest in debug builds.

The benefits of using CRASH:

- If a browser vendor traces crashes in their software, `CRASH()` can provide vital information from end users to allow an issue to be resolved.
- Code after `CRASH()` is guaranteed unreachable, which can help prevent some bugs from being security liabilities.

The cost of using CRASH:

- Use of the `CRASH` macro turns a programming error into a crash, blowing away a webpage or an entire web browser in cases that otherwise might be harmless.
- Checking for the error condition in release builds may slow the program down.

Guidelines for using ASSERT() and CRASH() macros.

- Use `ASSERT()` for things that should never happen, but if they do will cause incorrect results rather than a crash or memory corruption.
- Assertions are claims that a programmer knows to be true, and they fire only when that programmer turns out to be wrong because there is some kind of programming mistake. There should be no "wishful thinking" involved. For example, `ASSERT()` should not be used to verify that a file system call has succeeded, because there is no way for a programmer to guarantee that.
- Use `CRASH()` for cases that shouldn't happen, but if they do would be unrecoverable. e.g. out of memory

errors.

Examples of using CRASH() vs ASSERT()

```
// Not having any children indicates a programming e
ASSERT(object->numChildren() > 0);

Allocation bitmapStorage = systemAlloc(allocSize);
if (!bitmapStorage.pages)
    CRASH(); // Not possible to recover from an out
```

If you have any comments on the above or other ideas about improving the clarity, scope, or presentation, please send mail to the WebKit mailing list. ■

Next

Chris Fleizach is now a WebKit reviewer!

Learn more

Previously

Kent Tamura and Ojan Vafai are now WebKit reviewers!

Learn more