The Try/Catch That Cried Wolf

The Try/Catch That Cried Wolf

Posted by Nathaniel Francis
14 Aug 2019 10:23 AM

Had a PRversation (conversation around a pull request) with a colleague about the nature of try/catch and the simplicity of features.

Try/catch is designed to simplify and streamline our error handling process. Code that works, flows through the try block as written, code that fails gets an error thrown into the catch block where we can choose what we want to do with it. This is commonly known.

Less commonly known is that we can use this dynamic as a hack. For example, in a seldom used feature, one of our devs had the task of detecting a specific value within a five level deep data structure and assign that deeply nested value to a variable. Any of the five levels could've existed or not. Instead of writing a mile long conditional statement or complex detection function, he chose to assign the expected five-layer deep value to his variable inside a small try/catch block. If the assignment triggered an error, which would happen if any of the five levels were not in the expected state for the assignment to work, then the code would trigger the catch block. If the value was where it needed to be, then the try block completed as planned. This is a hack - bypassing thorough, responsible nested object value detection for a simple try/catch block. The thrown error from the try block to the catch block was an assessed and approved tradeoff for the more complex detection methods that would have been correct in this coding case.

I'm being specific for a reason. I cannot stress enough that this is a hack. It is a hack in the sense that it's using a code feature that is not intended, but possible. That's not a bad thing - it obviously helped to keep the complex nested data example simple. At the same time, it's not meant to become a coding norm. It should remain an exception. I would also suggest that it should be clearly commented as an exceptional code choice.

This is the discussion that the PRversation revolved around. The requester had provided multi-lined code that performed a moderate level detection logic on a value. The reviewer suggested that the detection logic could be replaced by the try/catch hack. We decided to not use the try/catch hack. This is why.

Remember Aesop's fable The Boy Who Cried wolf? The job of the boy in the fable was to care for the town flock and get help if a wolf came for the sheep (no King David here, apparently). The boy in the story discovered that he would get attention if he cried out that there was a wolf, regardless of whether a wolf was present or not - the villagers would come running to help him because they had vested interest in the flock. After a number of times of the boy faking a wolf attack, the villagers ignored his warnings. Then, when a real wolf came for the sheep, the boy and the sheep both perished. All this because of the fake alarming of the boy for whatever reason he had.

Using this fable as our analogy, let's make the village our program, the flock is the try block, the boy is the catch block, and the wolf is an error. The boy's (catch block) job is to warn the village (program) that a wolf (error) has attacked the flock (try block). This system works if the catch block (the boy) alerts when a real wolf is present. The catch block (boy) should not be triggered when there is no error nor should it trigger when a situation occurs in the try block (flock) that should be able to be handled. That's enough analogy for now.

Back to the PRversation. We decided to keep the moderate level detection logic in the code instead of implementing the try/catch hack for this reason. The hack should be used sparingly and only in well reasoned situations. Replacing it for moderate level logic does a few things:

1. It forces the code engine to generate an unnecessary error and capture it
2. It uses a dependable error detection pattern in a non-error way inconsistently

We don't want a try/catch block crying wolf. We want it handling wolves.

Yes, the hack works and is valuable in what I would hope are rare circumstances. Generally speaking, we want responsible try/catch blocks, not try/catch blocks that cry wolf.

Tags: development, JavaScript, tips and tricks
0 Comments