Semantic scope naming

Status

Proposed

Summary

When deciding which scopes to apply, built-in grammars should be guided only by what would be useful to annotate. A suggested scope addition should be assessed by the semantic value it adds. It not be rejected if its *only* drawback is that it would result in undesirable syntax highlighting for one of Atom's built-in syntax themes.

Motivation

Tree-sitter grammars are a unique opportunity to deliver more accurate scoping. They can identify constructs that would've been too ambiguous for a TM-style grammar.

Scopes themselves are immensely powerful; they're hooks that allow weirdos like myself to customize Atom to my exact specifications. Not only for syntax themes, either; I've got lots of commands that behave in different ways based on the surrounding scope. The richer the scope descriptor, the better. (Within reasonable bounds, of course.)

Explanation

I think this is best illustrated by example. Here's a ticket from `language-javascript` about syntax highlighting of imports. For example:

```
import { foo } from "thing";
```

For reference, the `language-babel` grammar scopes `foo` as `variable.other.readwrite.js`. I'd probably opt for something like `variable.import`; others may want to put it into the `support` namespace. There's actually little cross-language consensus here.

But right now, that `foo` doesn't have *any* scope name applied in the tree-sitter JavaScript grammar, and this is by design. The explanation, as stated in the ticket, is that Atom wants to avoid marking a variable with a certain color if it doesn't have the capability of making it the same color *throughout* the document, across its various usages.

This is a fine design goal; I don't think @maxbrunsfeld is wrong to argue for it. But I'd suggest that it isn't a design goal for `language-javascript` or any other grammar; it's a design goal for a *syntax theme*, and a grammar should not refrain from applying scopes in order to satisfy the design goals of a specific syntax theme.

This isn't just a beard-stroking nitpick on my part. I can think of a handful of reasons why someone might want to be able to spot import names at a glance. It's reasonable for someone to want `foo` in the example above to remain the same color throughout the file. It's also reasonable, I think, to want `foo` to have a special color on the line where it's introduced. Or to communicate that this token has special behavior — as it would if you have the js-hyperclick package installed and are in the habit of cmd-clicking package names to jump to the files where they're defined.

I don't mind that the One Dark syntax theme doesn't want to give `foo` a special color; I mind that its decision is also binding on *all possible* syntax themes. If that scope name is present, and it's undesirable to a syntax theme, that theme can apply the overrides necessary to ignore it. But if that scope name is missing altogether, that constrains *all* syntax themes, and I'm unable to write a syntax theme that behaves differently.

Thus, here's what I propose:

If an issue or PR proposes adding a scope and can justify its presence somehow — including the goal of parity with its non-tree-sitter predecessor — an answer of "no, because the built-in syntax themes don't want to highlight it that way" should become "OK, but only if someone does the associated work to ensure no visual regressions in the built-in syntax themes." That someone could be the PR's author or anyone else who has an interest in getting it landed.

This is tricky, of course — not only the coordination of PRs across packages, but also the need to apply overrides to all six (is it six? I think it's six) of the built-in syntax themes. If all built-in themes are going to share an austere philosophy (and it seems like that's the plan), then perhaps it makes sense for them to start sharing a core set of contextual rules. The only difference between them would be the specific color choices that they make.

Drawbacks

The drawback is that what I'm suggesting is a lot of work. I don't propose it lightly; I propose it because it strikes me as the least bad of all available choices.

Rationale and alternatives

And what are those other choices?

1. The status quo, in which built-in grammars (like `language-javascript`'s tree-sitter grammar) are developed with goals that are tightly coupled to the goals of syntax themes. I think this would be a tragic lost opportunity. The fact that the `tree-sitter-javascript` parser groks ES6 and JSX means that lots of people no longer have to rely on a third-party grammar like `language-babel`. If I can't get my syntax highlighting the way I want it because the built-in grammar applies scopes too sparsely, then my only recourse is to write my own tree-sitter grammar that adds in the mappings I want. That's easier than writing a TM-style grammar, but it still involves some portion of the community dedicating their efforts to an effective fork of `language-javascript`, and for far more mundane reasons than the fork that produced `language-babel` in the first place.

2. A suggestion made by @Ben3eeE in the issue that inspired this RFC: intentionally picking ornery scope names that don't have implicit highlighting in the built-in syntax themes. That's at least a way forward, but I think it abandons a hard-won lesson. TextMate's attempt to devise a system of semantic scope names has borne quite a bit of fruit. It's the reason why three major successor editors have signed on to the same conventions. Semantic naming acts as a kind of "middleware" that allows syntax themes and grammars to be unaware of each others' implementation details. I *could* write a PR that scopes our `foo` import from above with something like `import-specifier.identifier`, and I still might, but in choosing an arbitrary name I'm once again obligating syntax themes to care about a grammar's implementation details.

3. Some sort of grand compromise that I don't have the breadth of experience to envision on my own. I'm hoping for this one, actually. For instance, it occurs to me that the "variables shouldn't ever be highlighted with different colors across different usages" problem is only a problem in languages where there's no sigil to mark variables. PHP, Perl, and Less don't have this problem because all variables begin with a symbol. Maybe the solution is to include some token like `without-sigil` in the scope name, and then the built-in themes can write a rule like `.syntax--without-sigil { color: @mono-1 !important }`.

Unresolved questions

- Ideally, I'd love to have some sort of canonical scope document like TextMate and Sublime Text have. But the future of TM-style scope naming seems to be up in the air. I think that they're no less relevant in the era of tree-sitter grammars, but I bet others disagree.

- To what extent should tree-sitter grammars be expected to scope documents identically to their TM-style predecessors? Obviously not 100%, or else there'd be no gains. What's the right balancing test?
- Atom has made some infrastructural choices that can complicate how scopes get applied and consumed. Are these permanent? For instance, if I wanted to implement Alternative 2 (as described above), I could choose a scope name like `meta.variable.import`, on the assumption that `meta.`-prefixed scope names won't have syntax highlighting. But `meta.variable` gets caught by a `.syntax--variable` CSS selector just as much as it would if the scope name began with `variable`. The order and hierarchy implied in the scope name is not actually present. Syntax themes could write selectors more creatively to get around this — e.g., `*[class^="syntax--variable "]` instead of `.syntax--variable` — but I don't think many do, and I can hardly blame them. Is this a limitation that Atom can evolve its way out of without breaking anything? Or are we stuck with it?