Ruby - Feature #18654

Enhancements to prettyprint

03/22/2022 08:26 PM - kddnewton (Kevin Newton)

Status:	Closed
Priority:	Normal
Assignee:	akr (Akira Tanaka)
Target version:	

Description

This issue mirrors the pull request open at https://github.com/ruby/prettyprint/pull/3.

This pull request adds a bunch of new functionality to the PrettyPrint class. The goal is to enhance the PrettyPrint class enough to support all of the necessary print functionality that a formatter requires in order to print out Ruby source correctly.

First, how PrettyPrint works today. PrettyPrint has 3 primitives:

- Group - an object that represents a set of content and its associated breakpoints. These objects are usually nested. When one of them no longer fits on a single line, it is broken at all of its associated breakpoints. For example, if you have something like [text("abc"), breakable, text("def")] and that group no longer fits on one line, then you will have two lines of output.
- Breakable - an object that represents a location where a line of content can be split. When it is created, if the current group is broken already, then it inserts a newline and carries on. If it is not, then it adds itself to the output buffer.
- Text - this is a set of objects that are appended to a buffer. If the buffer overflows the maximum print width for the line, then the surrounding group is broken and the buffer is flushed to the output.

With those primitives in place, the printer maintains a buffer of content that is being added. If the buffer overflows the line, the content is flushed until another group is hit or there are no more breakpoints. There are a couple of limitations with the current approach:

- PrettyPrint assumes that content in Text will not change its representation if it is contained within a broken group versus contained within a flat group. This isn't a problem for the existing uses of PrettyPrint, but for my purposes of building a formatter, it definitely is. Consider something like trailing commas (where you want a comma if it is broken but nothing if it's not) or block operators (where you would use a do and end for multi-line (broken group) or braces for single line (flat group)).
- The Breakable class assumes that you always want to indent the subsequent line up to the current indentation level. This is true in most cases, and certainly for all the existing use cases. But there are times when you don't want that behavior (for example if you're in the middle of a nested indentation structure but have to force content to start at the beginning of the next line as in a comment with =begin..=end bounds).
- There's no way to force a group to break. You can access the current group in the printer with current_group, but that won't force the parent groups to break. Without hacking around a lot of stuff, it's difficult to get this behavior. This is necessary if you want to ensure a newline is added and respected, like after a keyword where it would be necessary.

This commit adds a couple new nodes to the printing tree, as well as enhancing the Breakable class. First, the new nodes:

- Align - this node effectively wraps the old @indent variable but allows you to align content within any of the other containers. You can also align to a string now instead of just an integer, which will print that string before each line.
- BreakParent - enforces that the surrounding group and all ancestral groups are broken.
- IfBreak - contains two sets of nodes, one for if the surrounding group is broken and one for if the surrounding group is flat.
- Indent - similar to the align node, but you don't have to specify anything and it just indents by one level.
- LineSuffix - this is a big enhancement to the printing algorithm that maintains a separate buffer for content that should be flushed before the next newline. This is convenient for implementing things like heredocs and trailing comments.
- Trim - a rarely used but important node that trims off whitespace that has already been added to the buffer in the case that you need to force something to begin at the start of the next line.

As for the enhancements to Breakable:

- It now accepts a force parameter (default to false), which will insert a BreakParent and slightly change semantics so that a newline is always added.
- It now accepts an indent parameter (default to true), which allows you to specify if you want the subsequent line to indent automatically.

For the most part, the code is completely compatible with the previous version. There are a couple of things that were removed that appeared to be all internally-facing functions. When they were removed all of the tests still passed, so I'm assuming they were only called internally. I can certainly add them back if it's deemed too risky but I very much doubt this is a problem.

- indent attr_reader which is now encapsulated in the printing algorithm

- group_queue attr_reader which had a reference to a queue that is no longer necessary
- break_outmost_groups method, which is now encapsulated in the printing algorithm
- group_sub method, which was only called by the group method anyway and is no longer necessary

There were a bunch of things that were added, including:

- force and indent parameters to the breakable method (they both have defaults so this shouldn't be an issue)
- Align, BreakParent, IfBreak, Indent, LineSuffix, and Trim nodes
- Buffer::DefaultBuffer, Buffer::StringBuffer, and Buffer::ArrayBuffer, which is just there to provide the ability to trim trailing whitespace
- PrettyPrint.visit(doc, visitor), which is useful for debugging and also necessary for propagating break parent nodes up the tree

All in all, none of the tests had to change, which is a good sign. From the user of this class's perspective, nothing is different. Internally however, there's a bunch of additional functionality and a lot more control over the printing algorithm! Also the ability to debug has been greatly enhanced with pretty_print methods on each of the nodes and the ability to walk the print tree nodes before they're printed.

Related issues:

Related to Ruby - Feature #18450: Force break in prettyprint	**Assigned**

History

#1 - 03/23/2022 02:49 AM - mame (Yusuke Endoh)

- Assignee set to akr (Akira Tanaka)

#2 - 04/15/2022 08:54 AM - mame (Yusuke Endoh)

I spent some time to understand your proposal, but gave up because it was too huge. Can you split your proposal to smaller ones?

You said "There are a couple of limitations with the current approach". I can understand each of the problems more or less. But I don't know which new feature solves which problem. How about focusing each of them? For example, can you show the minimum set of new features to achieve only the first one (training comma, or do..end/braces)? In addition, it would be very helpful if you could write not only English but also a small demo program that uses features you are proposing.

BTW, the maintainer of prettyprint, @akr (Akira Tanaka), created the library based on Wadler's paper. AFAIK, he was not very keen on implementing anything beyond the paper. This proposal may be facing high hurdle to moving forward.

#3 - 04/15/2022 08:55 AM - mame (Yusuke Endoh)

- Related to Feature #18450: Force break in prettyprint added

#4 - 04/15/2022 02:50 PM - kddnewton (Kevin Newton)

Thank you @mame (Yusuke Endoh). I can definitely split it up. I will include examples as well. If it's alright, I'll leave this ticket open and reference it.

For the most part, all of the nodes I've added are aligned with the Wadler paper. The algorithm difference of printing after the whole doc tree is built versus printing while it's being built is an implementation detail; it's not necessarily spelled out in the paper. All of this is to say, I feel that these changes are still in the spirit of the paper and don't deviate too much. However, if it is deemed too hard to maintain, I will just keep it in syntax tree if I need to.

#5 - 04/15/2022 03:47 PM - kddnewton (Kevin Newton)

@mame (Yusuke Endoh) I have included only a small amount of changes in a smaller PR just to get this started: https://github.com/ruby/prettyprint/pull/6. For now, it only addresses the problem of allowing the user to specify whether or not a Breakable should indent when it is broken. To support that change, the overall algorithm change has to happen as well, so I've added that. There are accompanying tests. If this PR gets merged, I can bring the rest of the functionality in much smaller increments.

#6 - 04/18/2022 08:38 AM - mame (Yusuke Endoh)

Thank you. So, is the essence of your proposal to delay output until flush is called? Sounds like a significant change for the design of prettyprint. I will ask @akr (Akira Tanaka) on this at the next dev meeting.

I'm unsure if breakable(indent: true) is a good first issue. The use case is not clear enough to me. Can you show a concrete example demonstrating the feature is actually useful? IMO, a "trailing comma" issue is better because the use case if clear (at least, to me). I think it is a good-to-have not only for syntax tree but also the current Kernel#pp.

I took a quick look at your new pull request, but it still looks huge to me. Is it really a minimal change to implement breakable(indent: true)? For example, it removes some public attr_reader fields, and adds some. Because they are incompatible changes, we basically need to investigate the actual usages and estimate the impact of the incompatibilities for each change. Also, I want you to change the variable names as little as possible. You may think they are trivial, but the accumulation of such changes makes it harder to review the pull request, I think.

#7 - 04/19/2022 09:11 PM - kddnewton (Kevin Newton)

Yes, the majority of the code changes in the initial PR are about delaying output until flush is called.

To be clear, this is what already happens until the maximum width is hit. The user has always had to call flush in case the final line contained content that hadn't been flushed yet. This just delays the whole thing so that we can do more dynamic content based on groups breaking.

I understand what you mean about indent: false not being a great first issue. It was just one of the smallest, so I was hoping that would be something I could put in. In syntax tree I use it in heredocs. When you're printing each line, you don't want it to indent up to the current level, you just want to print the string literal.

I can attempt a 3rd PR with trailing commas. That one is a little *more* involved because it requires having break contents and flat contents and then choosing between. It's not a big change off the 2nd PR, but it would be a big change off the existing gem.

I will go back to the 2nd PR and try to change as few variable names as possible. Maybe I can make it small enough that it would be okay to review.

#8 - 05/07/2022 01:38 AM - akr (Akira Tanaka)

kddeisz (Kevin Newton) wrote in #note-4:

> The algorithm difference of printing after the whole doc tree is built versus printing while it's being built is an implementation detail

I designed PrettyPrint that can handle very big or infinite input.
So it is degradation and I don't want to accept it.

In general, I couldn't understand why new nodes are required.

I think that it is possible to implement a method to break all outer groups.
So I can't understand why BreakParent node is required.

I feel IfBreak is too complex: it contains two sets of nodes.
It seems that trailing comma and {}/do-end require only two texts (not sets of nodes).
So, it would be enough to extend text method to have two texts for the surrounding group is broken and not.

I couldn't understand why Indent node is required.
There is PrettyPrint.nest method.

I couldn't understand Trim node.

I couldn't understand Align node well but I guess it is much different from current algorithm.

I think LineSuffix node to support here-document in PrettyPrint makes the algorithm too complex.
How about post-processing to support here-document?
Since PrettyPrint can pass any object to output,
we can pass an object to represent heredoc marker and heredoc content,
such as [:heredoc, "<<End", "heredoc content"].
The output can be post-processed to move the heredoc content after the next newline.

This mechanism is similar to (reverse of) Ruby parser (parse.y):
The lexer moves a heredoc content to the position of the heredoc marker.
The parser doesn't need to care specially with here-document.

#9 - 05/12/2022 01:44 PM - kddnewton (Kevin Newton)

@akr (Akira Tanaka) okay, thank you for the consideration. I'm going to maintain it as a separate project in this case as I don't see a way around some of your concerns.

#10 - 08/09/2023 01:19 PM - kddnewton (Kevin Newton)

- Status changed from Open to Closed