Ruby - Feature #20443

Allow Major GC's to be disabled

04/22/2024 04:02 PM - eightbitraptor (Matt V-H)

Status:	Closed	
Priority:	Normal	
Assignee:		
Target version:		

Description

[Github PR #10598]

Background

Ruby's GC running during Rails requests can have negative impacts on currently running requests, causing applications to have high tail-latency.

A technique to mitigate this high tail-latency is Out-of-band GC (OOBGC). This is basically where the application is run with GC disabled, and then GC is explicitly started after each request, or when no requests are in progress.

This can reduce the tail latency, but also introduces problems of its own. Long GC pauses after each request reduce throughput. This is more pronounced on threading servers like Puma because all the threads have to finish processing user requests and be "paused" before OOBGC can be triggered.

This throughput decrease happens for a couple of reasons:

1. There are few heuristics available for users to determine when GC should run, this means that in OOBGC scenarios, it's possible that major GC's are being run more than necessary. 2. The lack of any GC during a request means that lots of garbage objects have been created and not cleaned up, so the process is using more memory than it should - requiring major GC's run as part of OOBGC to do more work and therefore take more time.

This ticket attempts to address these issues by:

1. Provide GC.disable_major and its antonym GC.enable_major to disable and enable only major GC 2. Provide GC.needs_major? as a basic heuristic allowing users to tell when Ruby should run a Major GC.

These ideas were originally proposed by @ko1 (Koichi Sasada) and @byroot (Jean Boussier) in this rails issue

Disabling GC major's would still allow minor GC's to run during the request, avoiding the ballooning memory usage caused by not running GC at all, and reducing the time that a major takes when we do run it, because the nursery objects have been cleaned up during the request already so there is less work for a major GC to do.

This can be used in combination with GC.needs_major? to selectively run an OOBGC only when necessary

Implementation

This PR adds 3 new methods to the GC module

- GC.disable_major This prevents major GC's from running automatically. It does not restrict minors. When objspace->rgengc.need_major_gc is set and a GC is run, instead of running a major, new heap pages will be allocated and a minor run instead. objspace->rgengc.need_major_gc will remain set until a major is manually run. If a major is not manually run then the process will eventually run out of memory.

When major GC's are disabled, object promotion is disabled. That is, no objects will increment their ages during a minor GC. This is to attempt to minimise heap growth during the period between major GC's, by restricting the number of old-gen objects that will remain unconsidered by the GC until the next major.

When GC.start is run, then major GC's will be enabled, a GC triggered with the options passed to GC.start, and then disable_major will be set to the state it was in before GC.start was called.

- GC.enable_major This simply unsets the bit preventing major GC's. This will revert the GC to normal generational behaviour. Everything behaves as default again.

- GC.needs_major? This exposes the value of objspace->rgengc.need_major_gc to the user level API. This is already exposed in GC.latest_gc_info[:need_major_by] but I felt that a simpler interface would make this easier to use and result in more readable code. eg.

```
out_of_band do
  GC.start if GC.needs_major?
end
```

Because object aging is disabled when majors are disabled it is recommended to use this in conjunction with Process.warmup, which will prepare the heap by running a major GC, compacting the heap, and promoting every remaining object to old-gen. This ensures that minor GC's are running over the smallets possible set of young objects when GC.disable_major is true.

Benchmarks

We ran some tests in production on Shopify's core monolith over a weekend and found that:

Mean time spent in GC, as well as p99.9 and p99.99 GC times are all improved.

6cff5b11-2e21-40c1-bb84-d994e0e1798d
p99 GC time is slightly higher.

dc645cbe-9495-46f0-8485-24e790c42f32
We're running far fewer OOBGC major GC's now that we have GC.needs_major? than we were before, and we believe that this is contributing to a slightly increased number of minor GC's. raising the p99 slightly.

App response times are all improved

We see a ~2% reduction in average response times when compared againststandard GC (~7% p99, ~3% p99.9 and ~4% p99.99).

7a0d8fa5-77b2-469a-8a45-83b6f8ffefe9
This drops slightly to a ~3% reduction in average response times when compared against our normal OOBGC approach (~6% p99, ~2% p99.9 and ~3% p99.99).

dba38c07-0ed1-4dba-a5e6-b9df91024d73
EDIT: corrected a formula error in the original Average charts, numbers updated.

Associated revisions

Revision f543c68e1ce4abaafd535a4917129e55f89ae8f7 - 07/12/2024 01:43 PM - eightbitraptor (Matt V-H)

Provide GC.config to disable major GC collections

This feature provides a new method GC.config that configures internal GC configuration variables provided by an individual GC implementation.

Implemented in this PR is the option full_mark: a boolean value that will determine whether the Ruby GC is allowed to run a major collection while the process is running.

It has the following semantics

This feature configures Ruby's GC to only run minor GC's. It's designed
to give users relying on Out of Band GC complete control over when a
major GC is run. Configuring full_mark: false does two main things:

- Never runs a Major GC. When the heap runs out of space during a minor
  and when a major would traditionally be run, instead we allocate more
  heap pages, and mark objspace as needing a major GC.
- Don't increment object ages. We don't promote objects during GC, this
  will cause every object to be scanned on every minor. This is an
  intentional trade-off between minor GC's doing more work every time,
  and potentially promoting objects that will then never be GC'd.

The intention behind not aging objects is that users of this feature
should use a preforking web server, or some other method of pre-warming
the oldgen (like Nakayoshi fork)before disabling Majors. That way most
objects that are going to be old will have already been promoted.

This will interleave major and minor GC collections in exactly the same
what that the Ruby GC runs in versions previously to this. This is the
default behaviour.

- This new method has the following extra semantics:

  - GC.config with no arguments returns a hash of the keys of the
    currently configured GC
  - GC.config with a key pair (eg. GC.config(full_mark: true) sets
    the matching config key to the corresponding value and returns the
    entire known config hash, including the new values. If the key does
    not exist, nil is returned

- When a minor GC is run, Ruby sets an internal status flag to determine
  whether the next GC will be a major or a minor. When full_mark: false this flag is ignored and every GC will be a minor.

  This status flag can be accessed at
  GC.latest_gc_info(:needs_major_by). Any value other than nil means
  that the next collection would have been a major.

  Thus it's possible to use this feature to check at a predetermined
  time, whether a major GC is necessary and run one if it is. eg. After
  a request has finished processing.

  ```
  if GC.latest_gc_info(:needs_major_by)
    GC.start(full_mark: true)
  end
  ```

[Feature #20443]

Revision f543c68e1ce4abaafd535a4917129e55f89ae8f7 - 07/12/2024 01:43 PM - eightbitraptor (Matt V-H)

Provide GC.config to disable major GC collections

This feature provides a new method GC.config that configures internal
GC configuration variables provided by an individual GC implementation.

Implemented in this PR is the option full_mark: a boolean value that
will determine whether the Ruby GC is allowed to run a major collection
while the process is running.

It has the following semantics

This feature configures Ruby's GC to only run minor GC's. It's designed
to give users relying on Out of Band GC complete control over when a
major GC is run. Configuring full_mark: false does two main things:

- Never runs a Major GC. When the heap runs out of space during a minor
  and when a major would traditionally be run, instead we allocate more
  heap pages, and mark objspace as needing a major GC.
- Don't increment object ages. We don't promote objects during GC, this
  will cause every object to be scanned on every minor. This is an
  intentional trade-off between minor GC's doing more work every time,
  and potentially promoting objects that will then never be GC'd.

The intention behind not aging objects is that users of this feature
should use a preforking web server, or some other method of pre-warming
the oldgen (like Nakayoshi fork)before disabling Majors. That way most
objects that are going to be old will have already been promoted.

This will interleave major and minor GC collections in exactly the same
what that the Ruby GC runs in versions previously to this. This is the
default behaviour.

- This new method has the following extra semantics:

  - GC.config with no arguments returns a hash of the keys of the
    currently configured GC
  - GC.config with a key pair (eg. GC.config(full_mark: true) sets
    the matching config key to the corresponding value and returns the
    entire known config hash, including the new values. If the key does
    not exist, nil is returned

- When a minor GC is run, Ruby sets an internal status flag to determine
  whether the next GC will be a major or a minor. When full_mark: false this flag is ignored and every GC will be a minor.

  This status flag can be accessed at
  GC.latest_gc_info(:needs_major_by). Any value other than nil means
  that the next collection would have been a major.

  Thus it's possible to use this feature to check at a predetermined
  time, whether a major GC is necessary and run one if it is. eg. After
  a request has finished processing.

  ```
  if GC.latest_gc_info(:needs_major_by)
    GC.start(full_mark: true)
  end
  ```

[Feature #20443]

Revision f543c68e - 07/12/2024 01:43 PM - eightbitraptor (Matt V-H)

Provide GC.config to disable major GC collections

This feature provides a new method GC.config that configures internal
GC configuration variables provided by an individual GC implementation.

Implemented in this PR is the option full_mark: a boolean value that
will determine whether the Ruby GC is allowed to run a major collection
while the process is running.

It has the following semantics

This feature configures Ruby's GC to only run minor GC's. It's designed
to give users relying on Out of Band GC complete control over when a
major GC is run. Configuring full_mark: false does two main things:

- Never runs a Major GC. When the heap runs out of space during a minor
  and when a major would traditionally be run, instead we allocate more
  heap pages, and mark objspace as needing a major GC.
- Don't increment object ages. We don't promote objects during GC, this
  will cause every object to be scanned on every minor. This is an
  intentional trade-off between minor GC's doing more work every time,
  and potentially promoting objects that will then never be GC'd.

The intention behind not aging objects is that users of this feature
should use a preforking web server, or some other method of pre-warming
the oldgen (like Nakayoshi fork)before disabling Majors. That way most
objects that are going to be old will have already been promoted.

This will interleave major and minor GC collections in exactly the same
what that the Ruby GC runs in versions previously to this. This is the
default behaviour.

- This new method has the following extra semantics:

  - GC.config with no arguments returns a hash of the keys of the
    currently configured GC

  - GC.config with a key pair (eg. GC.config(full_mark: true) sets the matching config key to the corresponding value and returns the entire known config hash, including the new values. If the key does not exist, nil is returned

- When a minor GC is run, Ruby sets an internal status flag to determine whether the next GC will be a major or a minor. When full_mark: false this flag is ignored and every GC will be a minor.

  This status flag can be accessed at GC.latest_gc_info(:needs_major_by). Any value other than nil means that the next collection would have been a major.

  Thus it's possible to use this feature to check at a predetermined time, whether a major GC is necessary and run one if it is. eg. After a request has finished processing.

  ```
  if GC.latest_gc_info(:needs_major_by)
    GC.start(full_mark: true)
  end
  ```

[Feature #20443]

Revision 17b71e6c2e530ad258943ceb442abf0343bc3e12 - 07/12/2024 02:53 PM - peterzhu2118 (Peter Zhu)

[DOC] Fix link in NEWS.md for [Feature #20443]

Revision 17b71e6c2e530ad258943ceb442abf0343bc3e12 - 07/12/2024 02:53 PM - peterzhu2118 (Peter Zhu)

[DOC] Fix link in NEWS.md for [Feature #20443]

Revision 17b71e6c - 07/12/2024 02:53 PM - peterzhu2118 (Peter Zhu)

[DOC] Fix link in NEWS.md for [Feature #20443]

History

#1 - 04/22/2024 04:45 PM - byroot (Jean Boussier)

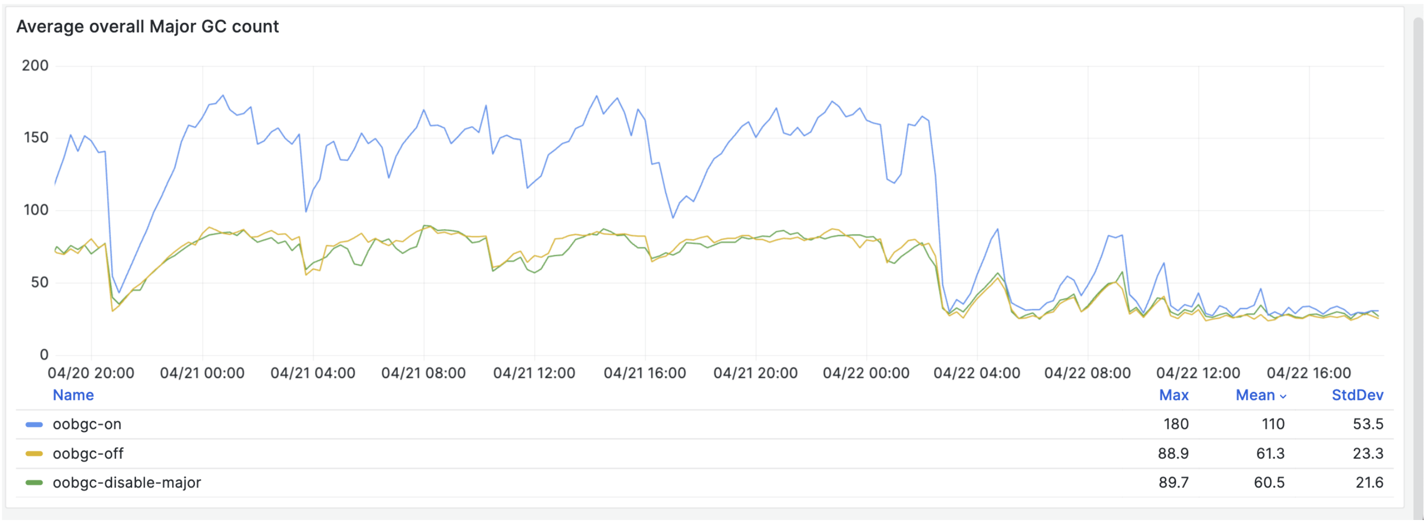
- File Capture d’écran 2024-04-22 à 18.41.52.png added

To add a little bit more context on @eightbitraptor's description. In some graph you see 3 groups:

- oobgc-off: which is workers without any Out of Band GC.
- oobgc-on: which is workers with our previous OOB GC implementation (once every 128 to 512 requests, 20% more every time)
- oobgc-disable-major: which is the new OOB GC implementation that only run GC when GC.need_major? returns true.

The new implementation not only improve latency in most case, it also reduce the capacity impact of having workers running major GC when it wasn't needed.

Here is the graph of GC.stat[:major_gc_count] over these 3 groups, and as you can see oobgc-disable-major runs major GC only about as often as no-OOBGC, whereas the previous implementation is triggering more often than actually needed, wasting server capacity.



Name	Max	Mean ⌄	StdDev
oobgc-on	180	110	53.5
oobgc-off	88.9	61.3	23.3
oobgc-disable-major	89.7	60.5	21.6

#2 - 04/22/2024 07:21 PM - eightbitraptor (Matt V-H)

- Description updated

#3 - 04/23/2024 08:31 AM - eightbitraptor (Matt V-H)

- Description updated

#4 - 04/23/2024 10:43 PM - nateberkopec (Nate Berkopec)

Regarding the interface:

```
GC.disable(major: true)
GC.disable(type: :major)
```

Should we consider these additional keyword arguments rather than adding a new method?

#5 - 04/24/2024 10:17 AM - eightbitraptor (Matt V-H)

nateberkopec (Nate Berkopec) wrote in #note-4:

> Regarding the interface:
>
> ```
> GC.disable(major: true)
> GC.disable(type: :major)
> ```
>
> Should we consider these additional keyword arguments rather than adding a new method?

I slightly prefer having a new method pair for this, however I don't object to changing it.

I do have a slight concern that GC.disable(major: true) could be read either as disabling major GC's or keeping majors enabled and disabling minors

So if we decide to use the keyword approach I prefer GC.disable(type: major)

#6 - 04/24/2024 11:57 AM - byroot (Jean Boussier)

> I slightly prefer having a new method pair for this

Same. it makes it easy to test for existence with respond_to? and alternative implementations can make them undefined methods like for Process.fork etc.

#7 - 04/24/2024 02:56 PM - shan (Shannon Skipper)

I wonder if "full_sweep" would be worth considering as an alternative to "major" to align with the existing GC.start(full_sweep: true) keyword argument? Or they could be aliased, but it seems nice to be consistent if I'm understanding it correctly that "major" and "full sweep" have equivalent meaning.

```
GC.start(full_sweep: true) # existing default
GC.enable_full_sweep
GC.disable_full_sweep
```

For checking if a full sweep or major is needed, would the addressing the Object convention mean GC.need_full_sweep? singular, Dir.exist? style? I wonder if "need" is the right word?

(edited to remove confusion)

#8 - 04/25/2024 12:59 AM - ko1 (Koichi Sasada)

Basically I like this idea. Some points.

- should not use "major" as a "major gc", so GC.disable_major should be GC.disable_major_gc and so on.
- I don't have strong opinion about GC.disable(major_gc: true) or GC.disable_major_gc
- "When major GC's are disabled, object promotion is disabled" what happens on oldgen->younggen references? points from the remembers set? I think we can promote this case because it makes minor gc faster (the promoted objects can not be freed until major gc, so the number of living objects is same).

#9 - 04/25/2024 01:12 AM - ko1 (Koichi Sasada)

- needs_major "s" should not be on method name (like File.exists -> File.exist)
- can you measure the memory consumption? It is a key compared with old OOBGC.

#10 - 04/25/2024 03:44 AM - duerst (Martin Dürst)

ko1 (Koichi Sasada) wrote in #note-8:

> Basically I like this idea. Some points.
>
> - should not use "major" as a "major gc", so GC.disable_major should be GC.disable_major_gc and so on.

Isn't the gc already very obvious from the class GC?

#11 - 04/25/2024 06:15 AM - byroot (Jean Boussier)

> what happens on oldgen->younggen references? points from the remembers set?

Yes.

> I think we can promote this case because it makes minor gc faster (the promoted objects can not be freed until major gc, so the number of living objects is same).

I understand your point, but I fear it could be counter-productive. We specifically stopped doing that in [Feature #19678] because there is many patterns in common Ruby code bases that are causing promotion.

I'd rather run **minor GC** out of band frequently, and **major GC** out of band very rarely, because the ratio of effectively permanent objects to ephemeral ones tend to be large in long running applications.

#12 - 05/09/2024 12:56 PM - matz (Yukihiro Matsumoto)

I am neutral on this proposal. However, I am concerned that there may not be a Major GC when GC made pluggable.

Matz.

#13 - 05/09/2024 08:29 PM - eightbitraptor (Matt V-H)

matz (Yukihiro Matsumoto) wrote in #note-12:

> I am neutral on this proposal. However, I am concerned that there may not be a Major GC when GC made pluggable.
>
> Matz.

I think this is still relevant when GC is pluggable. Ruby will always ship with the existing GC by default, and there will always be applications for which running standard OOBGC will be the best approach.

I would anticipate that this function would warn or become a no-op when a pluggable GC module is in use.

#14 - 05/12/2024 06:49 AM - byroot (Jean Boussier)

I think that same concern applies to some existing GC methods (e.g. GC.compact, GC.verify_compaction_reference) etc. So it would make sense that plugable GC would offer a way for the GCs to register behavior for these methods, and if they don't it either noop or acts as not implemented (respond_to? -> false etc).

#15 - 05/14/2024 07:23 AM - matz (Yukihiro Matsumoto)

As long as the pluggable GC issue is clearly stated somewhere, I have no objection.

Matz.

#16 - 06/03/2024 06:26 PM - ko1 (Koichi Sasada)

byroot (Jean Boussier) wrote in #note-11:

> > I think we can promote this case because it makes minor gc faster (the promoted objects can not be freed until major gc, so the number of living objects is same).
>
> I understand your point, but I fear it could be counter-productive. We specifically stopped doing that in [Feature #19678] because there is many patterns in common Ruby code bases that are causing promotion.
>
> I'd rather run **minor GC** out of band frequently, and **major GC** out of band very rarely, because the ratio of effectively permanent objects to ephemeral ones tend to be large in long running applications.

I understand the proposed code:

- if old object a refers to young object b, put a into a remember set.
- if the a->b reference leaves, a will be in a remember set.
- if the a->b reference was lost (a.b = nil for example), b will be free'ed and a is not in a remember set.

If there is no more permanent objects, it works well as #19678.
And on the web requests, most of objects will be free'ed after the request processing (other than cached objects).

I understand the logic so no objection here.

#17 - 06/03/2024 06:40 PM - ko1 (Koichi Sasada)

Proposed code just ignore gc_aging while "disable_major_gc" but it can increase aging up to 2 and next major gc makes the age 2 object to old object. In other words permanent objects can promote on single major gc easily.

#18 - 06/03/2024 09:24 PM - eightbitraptor (Matt V-H)

Thanks @ko1 (Koichi Sasada). I've updated the documentation as per @matz (Yukihiro Matsumoto) request, so I'll merge and close this now.

#19 - 06/04/2024 08:21 AM - eightbitraptor (Matt V-H)

eightbitraptor (Matthew Valentine-House) wrote in #note-18:

> Thanks @ko1 (Koichi Sasada). I've updated the documentation as per @matz (Yukihiro Matsumoto) request, ~~so I'll merge and close this now.~~

I've discussed this further with @ko1 (Koichi Sasada) on Slack, who would like the names clarified at the next dev meeting. So this ticket will remain open for now.

#20 - 06/06/2024 08:40 AM - matz (Yukihiro Matsumoto)

First, If you clearly define what would happen when the (plugged) GC does not generational scanning, I am OK for it. @byroot (Jean Boussier) told me that calling those methods would raise NotImplementedError when GC does not provide generational GC.

For API, I don't like the name needs_major?. At least, it should be need_major? to follow Ruby's naming convention (no third-person singular present).

And in the developers' meeting, @byroot (Jean Boussier) proposed GC.config(full_mark: true). How do you think?

Matz.

#21 - 06/10/2024 11:23 AM - eightbitraptor (Matt V-H)

matz (Yukihiro Matsumoto) wrote in #note-20:

> First, If you clearly define what would happen when the (plugged) GC does not generational scanning, I am OK for it. @byroot (Jean Boussier) told me that calling those methods would raise NotImplementedError when GC does not provide generational GC.
>
> For API, I don't like the name needs_major?. At least, it should be need_major? to follow Ruby's naming convention (no third-person singular present).
>
> And in the developers' meeting, @byroot (Jean Boussier) proposed GC.config(full_mark: true). How do you think?
>
> Matz.

I really like the idea of having a set of config parameters to the GC. That seems like a much more robust way of providing custom features to the GC without having to make new methods every time.

So, based on the latest discussions, what I'd like to propose is this:

- Introduce GC.config, currently with a single key full_mark.
  - full_mark: true should be the default behaviour - ie. the GC will work as it currently does.
- Ruby code can set full_mark to false like this: GC.config(full_mark: false) - In this case, no major GC's will be run unless explicitly requested using GC.start(full_mark: true)
- A user can check whether a major GC is needed at any time by checking GC.latest_gc_info(:needs_major_by). Any value other than nil means that the GC would do a major on the next invocation.

I think this has a few benefits over the existing approach that I proposed in this ticket

1. Introduces a flexible way of providing config information to the GC. This will also allow pluggable GC's implemented in the future to define their own config keys without changing the interface of the GC module.
2. Does not introduce any new methods that are implementation specific to the current GC. This removes the decision about how future GC module

> needs to respond to these methods.

#22 - 07/08/2024 05:47 PM - ko1 (Koichi Sasada)

> Introduce GC.config, currently with a single key full_mark.

I'm okay to introduce it but not sure config or configure or other word? (English issue).

#23 - 07/08/2024 06:17 PM - byroot (Jean Boussier)

> but not sure config or configure or other word? (English issue).

Given GC.config returns the current configuration, I can't be .configure.

It could be .configuration, or any other concept. We can add it to the upcoming dev meeting to get a decision on that name.

#24 - 07/08/2024 07:22 PM - eightbitraptor (Matt V-H)

ko1 (Koichi Sasada) wrote in #note-22:

> > Introduce GC.config, currently with a single key full_mark.
>
> I'm okay to introduce it but not sure config or configure or other word? (English issue).

I prefer config for the following reasons

- configure is a verb and therefore implies taking an action. as @byroot (Jean Boussier) says, this doesn't fit with returning the existing configuration when no arguments are passed.
- config is already a familiar term to developers (config files, config object in Rails etc)
- configuration is just the un-abbreviated form of config. It means the same thing but it takes longer to type.

#25 - 07/11/2024 10:04 AM - ko1 (Koichi Sasada)

On dev-meeting, there is no objection on GC.config().
Could you write the more doc of it? For example, returning value is unclear.

About keys:

- Matz prefer to add prefix for the keys, like GC.config(gengc_full_mark: true).
- I think GC.config(gengc_full_mark: true) is not clear always do full marking or not (gengc_full_mark: false is clear that full marking is prohibited). gengc_allow_full_mark: true/false? too much?

#26 - 07/11/2024 08:21 PM - eightbitraptor (Matt V-H)

ko1 (Koichi Sasada) wrote in #note-25:

> On dev-meeting, there is no objection on GC.config().
> Could you write the more doc of it? For example, returning value is unclear.
>
> About keys:
>
> - Matz prefer to add prefix for the keys, like GC.config(gengc_full_mark: true).
> - I think GC.config(gengc_full_mark: true) is not clear always do full marking or not (gengc_full_mark: false is clear that full marking is prohibited). gengc_allow_full_mark: true/false? too much?

Thank you for discussing this. I will ship this with GC.config as the method name. I have changed the config key to rgengc_allow_full_mark, because despite being long, it is descriptive and unambiguous. I have also fully documented the semantics of GC.config, as well as the behaviour of this key.

#27 - 07/12/2024 01:50 PM - eightbitraptor (Matt V-H)

- Status changed from Open to Closed

Applied in changeset git|f543c68e1ce4abaafd535a4917129e55f89ae8f7.

Provide GC.config to disable major GC collections

This feature provides a new method GC.config that configures internal
GC configuration variables provided by an individual GC implementation.

Implemented in this PR is the option full_mark: a boolean value that
will determine whether the Ruby GC is allowed to run a major collection
while the process is running.

It has the following semantics

This feature configures Ruby's GC to only run minor GC's. It's designed
to give users relying on Out of Band GC complete control over when a
major GC is run. Configuring full_mark: false does two main things:

- Never runs a Major GC. When the heap runs out of space during a minor
  and when a major would traditionally be run, instead we allocate more
  heap pages, and mark objspace as needing a major GC.
- Don't increment object ages. We don't promote objects during GC, this
  will cause every object to be scanned on every minor. This is an
  intentional trade-off between minor GC's doing more work every time,
  and potentially promoting objects that will then never be GC'd.

The intention behind not aging objects is that users of this feature
should use a preforking web server, or some other method of pre-warming
the oldgen (like Nakayoshi fork)before disabling Majors. That way most
objects that are going to be old will have already been promoted.

This will interleave major and minor GC collections in exactly the same
what that the Ruby GC runs in versions previously to this. This is the
default behaviour.

- This new method has the following extra semantics:

  - GC.config with no arguments returns a hash of the keys of the
    currently configured GC
  - GC.config with a key pair (eg. GC.config(full_mark: true) sets
    the matching config key to the corresponding value and returns the
    entire known config hash, including the new values. If the key does
    not exist, nil is returned

- When a minor GC is run, Ruby sets an internal status flag to determine
  whether the next GC will be a major or a minor. When full_mark: false this flag is ignored and every GC will be a minor.

  This status flag can be accessed at
  GC.latest_gc_info(:needs_major_by). Any value other than nil means
  that the next collection would have been a major.

  Thus it's possible to use this feature to check at a predetermined
  time, whether a major GC is necessary and run one if it is. eg. After
  a request has finished processing.

  ```
  if GC.latest_gc_info(:needs_major_by)
    GC.start(full_mark: true)
  end
  ```

[Feature #20443]

Files

Capture d'écran 2024-04-22 à 18.41.52.png	279 KB	04/22/2024	byroot (Jean Boussier)