Ruby - Bug #20209

YJIT can leak memory by retaining objects with singleton class

01/24/2024 11:06 PM - alanwu (Alan Wu)

Status:	Closed		
Priority:	Normal		
Assignee:			
Target version:			
ruby -v:		**Backport:**	3.0: DONTNEED, 3.1: DONTNEED, 3.2: DONTNEED, 3.3: DONE

Description

We've received reports of YJIT causing memory leaks in production Rails apps by keeping objects that have singleton classes alive. The symptom is similar to #19436. We have found a workaround with https://github.com/ruby/ruby/pull/9693 and would like to have it in the next 3.3 point release.

Associated revisions

Revision 2cc7a56ec7830fd5efaf2bc449637fd831743714 - 01/24/2024 11:06 PM - alanwu (Alan Wu)

YJIT: Avoid leaks by skipping objects with a singleton class

For receiver with a singleton class, there are multiple vectors YJIT can end up retaining the object. There is a path in jit_guard_known_klass() that bakes the receiver into the code, and the object could also be kept alive indirectly through a path starting at the CME object baked into the code.

To avoid these leaks, avoid compiling calls on objects with a singleton class.

See: https://github.com/Shopify/ruby/issues/552

[Bug #20209]

Revision 2cc7a56ec7830fd5efaf2bc449637fd831743714 - 01/24/2024 11:06 PM - alanwu (Alan Wu)

YJIT: Avoid leaks by skipping objects with a singleton class

For receiver with a singleton class, there are multiple vectors YJIT can end up retaining the object. There is a path in jit_guard_known_klass() that bakes the receiver into the code, and the object could also be kept alive indirectly through a path starting at the CME object baked into the code.

To avoid these leaks, avoid compiling calls on objects with a singleton class.

See: https://github.com/Shopify/ruby/issues/552

[Bug #20209]

Revision 2cc7a56e - 01/24/2024 11:06 PM - alanwu (Alan Wu)

YJIT: Avoid leaks by skipping objects with a singleton class

For receiver with a singleton class, there are multiple vectors YJIT can end up retaining the object. There is a path in jit_guard_known_klass() that bakes the receiver into the code, and the object could also be kept alive indirectly through a path starting at the CME object baked into the code.

To avoid these leaks, avoid compiling calls on objects with a singleton class.

See: https://github.com/Shopify/ruby/issues/552

[Bug #20209]

Revision cdcabd8a44ee2f4a2b549a3460a5c77c2dffca36 - 03/14/2024 04:26 PM - alanwu (Alan Wu)

Backport 3.3: YJIT memory leak fix with additional CI fixes (#9841)

merge revision(s) 2cc7a56e,b0711b1,db5d9429: [Backport #20209]

```
    YJIT: Avoid leaks by skipping objects with a singleton class

    For receiver with a singleton class, there are multiple vectors YJIT can
    end up retaining the object. There is a path in jit_guard_known_klass()
    that bakes the receiver into the code, and the object could also be kept
    alive indirectly through a path starting at the CME object baked into
    the code.

    To avoid these leaks, avoid compiling calls on objects with a singleton
    class.

    See: https://github.com/Shopify/ruby/issues/552

    [Bug #20209]
    ---
     yjit/bindgen/src/main.rs       |  1 +
     yjit/src/codegen.rs            | 17 ++++++++++++++++
     yjit/src/cruby_bindings.inc.rs |  1 +
     yjit/src/stats.rs              |  2 ++
     4 files changed, 21 insertions(+)

    YJIT: Fix tailcall and JIT entry eating up FINISH frames (#9729)

    Suppose YJIT runs a rb_vm_opt_send_without_block()
    fallback and the control frame stack looks like:

    ```
    will_tailcall_bar [FINISH]
    caller_that_used_fallback
    ```

    will_tailcall_bar() runs in the interpreter and sets up a tailcall.
    Right before JIT_EXEC() in the `send` instruction, the stack will look like:

    ```
    bar [FINISH]
    caller_that_used_fallback
    ```

    Previously, JIT_EXEC() ran bar() in JIT code, which caused the `FINISH`
    flag to return to the interpreter instead of to the JIT code running
    caller_that_used_fallback(), causing code to run twice and probably
    crash. Recent flaky failures on CI about "each stub expects a particular
    iseq" are probably due to leaving methods twice in
    `test_optimizations.rb`.

    Only run JIT code from the interpreter if a new frame is pushed.
    ---
     test/ruby/test_optimization.rb | 11 ++++++++++
     vm_exec.h | 3 ++-
     2 files changed, 13 insertions(+), 1 deletion(-)

    YJIT: No need to RESTORE_REG now that we reject tailcalls

    Thanks to Kokubun for noticing.

    Follow-up: b0711b1cf152afad0a480ee2f9bedd142a0d24ac
    ---
     vm_exec.h | 1 -
     1 file changed, 1 deletion(-)
```

Revision cdcabd8a44ee2f4a2b549a3460a5c77c2dffca36 - 03/14/2024 04:26 PM - alanwu (Alan Wu)

Backport 3.3: YJIT memory leak fix with additional CI fixes (#9841)

merge revision(s) 2cc7a56e,b0711b1,db5d9429: [Backport #20209]

```
    YJIT: Avoid leaks by skipping objects with a singleton class
```

```
    For receiver with a singleton class, there are multiple vectors YJIT can
    end up retaining the object. There is a path in jit_guard_known_klass()
    that bakes the receiver into the code, and the object could also be kept
    alive indirectly through a path starting at the CME object baked into
    the code.

    To avoid these leaks, avoid compiling calls on objects with a singleton
    class.

    See: https://github.com/Shopify/ruby/issues/552

    [Bug #20209]
    ---
     yjit/bindgen/src/main.rs       |  1 +
     yjit/src/codegen.rs            | 17 ++++++++++++++++
     yjit/src/cruby_bindings.inc.rs |  1 +
     yjit/src/stats.rs              |  2 ++
     4 files changed, 21 insertions(+)

    YJIT: Fix tailcall and JIT entry eating up FINISH frames (#9729)

    Suppose YJIT runs a rb_vm_opt_send_without_block()
    fallback and the control frame stack looks like:

    ```
    will_tailcall_bar [FINISH]
    caller_that_used_fallback
    ```

    will_tailcall_bar() runs in the interpreter and sets up a tailcall.
    Right before JIT_EXEC() in the `send` instruction, the stack will look like:

    ```
    bar [FINISH]
    caller_that_used_fallback
    ```

    Previously, JIT_EXEC() ran bar() in JIT code, which caused the `FINISH`
    flag to return to the interpreter instead of to the JIT code running
    caller_that_used_fallback(), causing code to run twice and probably
    crash. Recent flaky failures on CI about "each stub expects a particular
    iseq" are probably due to leaving methods twice in
    `test_optimizations.rb`.

    Only run JIT code from the interpreter if a new frame is pushed.
    ---
     test/ruby/test_optimization.rb | 11 ++++++++++
     vm_exec.h | 3 ++-
     2 files changed, 13 insertions(+), 1 deletion(-)

    YJIT: No need to RESTORE_REG now that we reject tailcalls

    Thanks to Kokubun for noticing.

    Follow-up: b0711b1cf152afad0a480ee2f9bedd142a0d24ac
    ---
     vm_exec.h | 1 -
     1 file changed, 1 deletion(-)
```

Revision cdcabd8a - 03/14/2024 04:26 PM - alanwu (Alan Wu)

Backport 3.3: YJIT memory leak fix with additional CI fixes (#9841)

merge revision(s) 2cc7a56e,b0711b1,db5d9429: [Backport #20209]

```
    YJIT: Avoid leaks by skipping objects with a singleton class

    For receiver with a singleton class, there are multiple vectors YJIT can
    end up retaining the object. There is a path in jit_guard_known_klass()
    that bakes the receiver into the code, and the object could also be kept
    alive indirectly through a path starting at the CME object baked into
    the code.

    To avoid these leaks, avoid compiling calls on objects with a singleton
    class.
```

```
    See: https://github.com/Shopify/ruby/issues/552

    [Bug #20209]
    ---
     yjit/bindgen/src/main.rs       |  1 +
     yjit/src/codegen.rs            | 17 ++++++++++++++++
     yjit/src/cruby_bindings.inc.rs |  1 +
     yjit/src/stats.rs              |  2 ++
     4 files changed, 21 insertions(+)

    YJIT: Fix tailcall and JIT entry eating up FINISH frames (#9729)

    Suppose YJIT runs a rb_vm_opt_send_without_block()
    fallback and the control frame stack looks like:

    ```
    will_tailcall_bar [FINISH]
    caller_that_used_fallback
    ```

    will_tailcall_bar() runs in the interpreter and sets up a tailcall.
    Right before JIT_EXEC() in the `send` instruction, the stack will look like:

    ```
    bar [FINISH]
    caller_that_used_fallback
    ```

    Previously, JIT_EXEC() ran bar() in JIT code, which caused the `FINISH`
    flag to return to the interpreter instead of to the JIT code running
    caller_that_used_fallback(), causing code to run twice and probably
    crash. Recent flaky failures on CI about "each stub expects a particular
    iseq" are probably due to leaving methods twice in
    `test_optimizations.rb`.

    Only run JIT code from the interpreter if a new frame is pushed.
    ---
     test/ruby/test_optimization.rb | 11 ++++++++++
     vm_exec.h | 3 ++-
     2 files changed, 13 insertions(+), 1 deletion(-)

    YJIT: No need to RESTORE_REG now that we reject tailcalls

    Thanks to Kokubun for noticing.

    Follow-up: b0711b1cf152afad0a480ee2f9bedd142a0d24ac
    ---
     vm_exec.h | 1 -
     1 file changed, 1 deletion(-)
```

History

#1 - 02/05/2024 10:18 PM - alanwu (Alan Wu)

Note, there was an additional fix to stabilize CI after this. I opened a GitHub PR with everything bundled together:
https://github.com/ruby/ruby/pull/9841

#2 - 03/20/2024 12:58 PM - naruse (Yui NARUSE)

- Backport changed from 3.0: DONTNEED, 3.1: DONTNEED, 3.2: DONTNEED, 3.3: REQUIRED to 3.0: DONTNEED, 3.1: DONTNEED, 3.2: DONTNEED, 3.3: DONE

ruby_3_3 cdcabd8a44ee2f4a2b549a3460a5c77c2dffca36 merged revision(s) 2cc7a56e,b0711b1,db5d9429.