Some pieces of the Rust object system: 

extension, overriding, and self

Lindsey Kuper
Mozilla Research
August 18, 2011
Me and how I got here
Me and how I got here

- Graduated college (CS and music) in 2004
Me and how I got here

- Graduated college (CS and music) in 2004
- Web development at a (failed) startup, 2004–2006
Me and how I got here

- Graduated college (CS and music) in 2004
- Web development at a (failed) startup, 2004–2006
- Perl plumbing at a publishing company, 2006–2008
Me and how I got here

- Graduated college (CS and music) in 2004
- Web development at a (failed) startup, 2004–2006
- Perl plumbing at a publishing company, 2006–2008
  - but in 2007, I moved in with a couple of Haskell hackers...
Me and how I got here

- Graduated college (CS and music) in 2004
- Web development at a (failed) startup, 2004–2006
- Perl plumbing at a publishing company, 2006–2008
  - but in 2007, I moved in with a couple of Haskell hackers...
- Ph.D. student at Indiana studying PL since fall 2008
Me and how I got here

- Graduated college (CS and music) in 2004
- Web development at a (failed) startup, 2004–2006
- Perl plumbing at a publishing company, 2006–2008
  - but in 2007, I moved in with a couple of Haskell hackers...
- Ph.D. student at Indiana studying PL since fall 2008
  - and then I saw a job posting for Rust...
What’s Rust?

a systems language
pursuing the trifecta:
safe, concurrent, fast
You’re working on the *what* system?!
You’re working on the what system?!

- I was intrigued by the idea of a classless object model and flexible prototype-style objects
You’re working on the what system?!

- I was intrigued by the idea of a classless object model and flexible prototype-style objects
- and was told, “None of that’s implemented yet; go for it!”
You’re working on the *what* system?!

- I was intrigued by the idea of a classless object model and flexible prototype-style objects
  - and was told, “None of that’s implemented yet; go for it!”
  - No object extension, method overriding, or self-dispatch
You’re working on the *what* system?!

- I was intrigued by the idea of a classless object model and flexible prototype-style objects
  - and was told, “None of that’s implemented yet; go for it!”
  - No object extension, method overriding, or self-dispatch
- During my internship, I implemented those things
You’re working on the *what* system?! 

- I was intrigued by the idea of a classless object model and flexible prototype-style objects
  - and was told, “None of that’s implemented yet; go for it!”
  - No object extension, method overriding, or self-dispatch
- During my internship, I implemented those things
  - and learned that they interact with each other in interesting ways
Self-dispatch

```javascript
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.zzz() == "meow");
```
Self-dispatch + object extension

```plaintext
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.zzz() == "meow");
```
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.ddd() == "meow");

let longcat = obj() {
    fn lol() -> str {
        ret "lol";
    }
    fn nyan() -> str {
        ret "nyan";
    }
    with shortcut
    assert (longcat.zzz() == "meow");
A brainteaser...

On my first attempt, this returned "lol". Why?
A brainteaser...

```c
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.zzz() == "meow");
```

```c
let longcat = obj() {
    fn lol() -> str {
        ret "lol";
    }
    fn nyan() -> str {
        ret "nyan";
    }
    with shortcat
    }
assert (longcat.zzz() == "meow");
```
A brainteaser...

```
obj cat() {  
  fn ack() -> str {  
    ret "ack";
  }
  fn meow() -> str {  
    ret "meow";
  }
  fn zzz() -> str {  
    ret self.meow();
  }
}

let shortcat = cat();
assert (shortcat.zzz() == "meow");
```

```
let longcat = obj() {  
  fn lol() -> str {  
    ret "lol";
  }
  fn nyan() -> str {  
    ret "nyan";
  }
  with shortcat
  
  assert (longcat.zzz() == "meow");
```

<table>
<thead>
<tr>
<th>shortcut’s vtable</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ack</td>
<td>lol</td>
<td>meow</td>
<td>nyan</td>
<td>zzz</td>
</tr>
<tr>
<td></td>
<td>&quot;ack&quot;</td>
<td>&quot;lol&quot;</td>
<td>&quot;meow&quot;</td>
<td>&quot;nyan&quot;</td>
<td>self.meow()</td>
</tr>
<tr>
<td></td>
<td>forward to shortcat.ack()</td>
<td>ret &quot;lol&quot;</td>
<td>forward to shortcat.meow()</td>
<td>ret &quot;nyan&quot;</td>
<td>forward to shortcat.zzz()</td>
</tr>
</tbody>
</table>
How to fix it

```javascript
let longcat = obj() {
  fn lol() -> str {
    ret "lol";
  }
  fn nyan() -> str {
    ret "nyan";
    with shortcut
    assert (longcat.zzz() == "meow");
  }
}

let shortcut = cat();
assert (shortcut.zzz() == "meow");
```

<table>
<thead>
<tr>
<th>longcat's vtable</th>
<th>shortcut's vtable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ack</td>
<td></td>
</tr>
<tr>
<td>1 lol</td>
<td></td>
</tr>
<tr>
<td>2 meow</td>
<td></td>
</tr>
<tr>
<td>3 nyan</td>
<td></td>
</tr>
<tr>
<td>4 zzz</td>
<td></td>
</tr>
</tbody>
</table>

0 ack [forward to] shortcut.ack()
1 lol ret "lol"
2 meow [forward to] shortcut.meow()
3 nyan ret "nyan"
4 zzz [forward to] shortcut.zzz()
How to fix it

```plaintext
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let longcat = obj() {
    fn lol() -> str {
        ret "lol";
    }
    fn nyan() -> str {
        ret "nyan";
    }
    with shortcat
}

assert (longcat.zzz() == "meow");
```

**shortcat’s backwarding vtable**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
<td>backward to longcat.ack()</td>
<td>backward to longcat.meow()</td>
<td>backward to longcat.zzz()</td>
</tr>
</tbody>
</table>

**longcat’s vtable**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ack</td>
<td>forward to shortcat.ack()</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lol</td>
<td></td>
<td>ret &quot;lol&quot;</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>meow</td>
<td></td>
<td>forward to shortcat.meow()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nyan</td>
<td></td>
<td></td>
<td>ret &quot;nyan&quot;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>zzz</td>
<td></td>
<td></td>
<td>forward to shortcat.zzz()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Self-dispatch + object extension + overriding

```plaintext
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.zzz() == "meow");
```
Self-dispatch + object extension + overriding

```plaintext
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.aaa() == "meow");
```
Self-dispatch + object extension + overriding

```javascript
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let longercat = obj() {
    fn meow() -> str {
        ret "zzz";
    }
    with shortcut
}

assert (longercat.zzz() == "zzz");

let shortcut = cat();
assert (shortcut.zzz() == "meow");
```
Self-dispatch + object extension + overriding

```
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let shortcat = cat();
assert (shortcat.zzz() == "meow");

let longercat = obj() {
    fn meow() -> str {
        ret "zzz";
    }
    with shortcat
    {
        assert (longercat.zzz() == "zzz");
    }
}
```

**longercat's vtable**

<table>
<thead>
<tr>
<th>Index</th>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ack</td>
<td>forward to shortcat.ack()</td>
</tr>
<tr>
<td>1</td>
<td>meow</td>
<td>ret &quot;zzz&quot;</td>
</tr>
<tr>
<td>2</td>
<td>zzz</td>
<td>forward to shortcat.zzz()</td>
</tr>
</tbody>
</table>
Self-dispatch + object extension + overriding

```rust
let longercat = obj() {
    fn meow() -> str {
        ret "meow"
    }
    with shortcut
    assert (longercat.zzz() == "zzz")
}

let shortcut = cat();
assert (shortcut.zzz() == "meow")
```

<table>
<thead>
<tr>
<th>shortcut's vtable</th>
<th>longercat's vtable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 ack ret &quot;ack&quot;</td>
<td>0 ack forward to shortcut.ack()</td>
</tr>
<tr>
<td>1 meow ret &quot;meow&quot;</td>
<td>1 meow ret &quot;zzz&quot;</td>
</tr>
<tr>
<td>2 zzz ret self.meow()</td>
<td>2 zzz forward to shortcut.zzz()</td>
</tr>
</tbody>
</table>
Self-dispatch + object extension + overriding

```python
obj cat() {  
    fn ack() -> str {  
        ret "ack";
    }
    fn meow() -> str {  
        ret "meow";
    }
    fn zzz() -> str {  
        ret self.meow();
    }
}
```

```python
let longercat = obj() {  
    fn meow() -> str {  
        ret "zzz";
    }
    with shortcutcat
    assert (longercat.zzz() == "zzz");
}
```

### shortcutcat’s vtable

<table>
<thead>
<tr>
<th>0</th>
<th>ack</th>
<th>backward to longercat.ack()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>meow</td>
<td>backward to longercat.meow()</td>
</tr>
<tr>
<td>2</td>
<td>zzz</td>
<td>backward to longercat.zzz()</td>
</tr>
</tbody>
</table>

### longercat’s vtable

<table>
<thead>
<tr>
<th>0</th>
<th>ack</th>
<th>forward to shortcutcat.ack()</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>meow</td>
<td>ret “zzz”</td>
</tr>
<tr>
<td>2</td>
<td>zzz</td>
<td>forward to shortcutcat.zzz()</td>
</tr>
</tbody>
</table>
Self-dispatch + object extension + overriding

```plaintext
let longercat = obj() {
  fn meow() -> str {
    ret "meow";
  }
  with shortcut
    assert (longercat.zzz() == "zzz");
}

let shortcut = cat();
assert (shortcut.zzz() == "meow");
```

```plaintext
obj cat() {
  fn ack() -> str {
    ret "ack";
  }
  fn meow() -> str {
    ret "meow";
  }
  fn zzz() -> str {
    ret self.meow();
  }
}
```
```plaintext
Self-dispatch + object extension + overriding

```
```javascript
obj cat() {
    fn ack() -> str {
        ret "ack";
    }
    fn meow() -> str {
        ret "meow";
    }
    fn zzz() -> str {
        ret self.meow();
    }
}

let longercat = obj() {
    fn meow() -> str {
        ret "zzz";
    }
    with shortcut
}

assert (longercat.zzz() == "zzz");

let shortcut = cat();

assert (shortcut.zzz() == "zzz");

let evenlongercat = obj() {
    fn meow() -> str {
        ret "zzzzzzz";
    }
    with longercat
}

assert (evenlongercat.zzz() == "zzzzzzz");
```
Go check it out!

http://rust-lang.org
Life goal achieved!

@ryanqnorth
Ryan North

@shaver @lindsey @pcwalton as near as i can tell all the best people are at Mozilla, measuring "bestness" by "good at twitter" at least!

15 Aug via web  Favorite  t3 Undo Retweet  Reply
Questions?

Thanks to:
Graydon Hoare and everyone on the Rust team
Dave Herman and all of Mozilla Research

Me: lkuper@cs.indiana.edu; @lindsey
Rust: http://rust-lang.org

Photo by jamesrbowe on Flickr. Thanks!