



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

---



# Self-dispatch

---

```
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

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

```
let shortcut = cat();
```

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

# Self-dispatch + object extension

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

```
let shortcut = cat();
```

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

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

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

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



# A brainteaser...

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

```
let shortcut = cat();
```

```
assert (shortcut.zzz() == "meow")
```

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

longcat's vtable

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()

# A brainteaser...

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

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

shortcut's vtable

0	ack	ret "ack"
1	meow	ret "meow"
2	zzz	ret self.meow()

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

longcat's vtable

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

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

```
let shortcut = cat();
```

```
assert (shortcut.zzz() == "meow")
```

shortcut's vtable

0	ack	ret "ack"
1	meow	ret "meow"
2	zzz	ret self.meow()

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

longcat's vtable

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

```
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 shortcut
};

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

shortcut's <b>backwaring</b> vtable		
0	ack	<b>backward to</b> longcat.ack()
1	meow	<b>backward to</b> longcat.meow()
2	zzz	<b>backward to</b> longcat.zzz()

shortcut's vtable		
0	ack	ret "ack"
1	meow	ret "meow"
2	zzz	ret self.meow()

longcat's vtable		
0	ack	<b>forward to</b> shortcut.ack()
1	lol	ret "lol"
2	meow	<b>forward to</b> shortcut.meow()
3	nyan	ret "nyan"
4	zzz	<b>forward to</b> shortcut.zzz()



# Self-dispatch + object extension + overriding

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

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

# Self-dispatch + object extension + overriding

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

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

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

longercat's vtable

0	ack	forward to shortcut.ack()
1	meow	ret "zzz"
2	zzz	forward to shortcut.zzz()



# Self-dispatch + object extension + overriding

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

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

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

shortcut's vtable

0	ack	ret "ack"
1	meow	ret "meow"
2	zzz	ret self.meow()

longercat's vtable

0	ack	forward to shortcut.ack()
1	meow	ret "zzz"
2	zzz	forward to shortcut.zzz()

# Self-dispatch + object extension + overriding

```
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");
```

shortcut's **backwaring** vtable

0	ack	<b>backward to</b> longcat.ack()
1	meow	<b>backward to</b> longcat.meow()
2	zzz	<b>backward to</b> longcat.zzz()

shortcut's vtable

0	ack	ret "ack"
1	meow	ret "meow"
2	zzz	ret self.meow()

longercat's vtable

0	ack	<b>forward to</b> shortcut.ack()
1	meow	ret "zzz"
2	zzz	<b>forward to</b> shortcut.zzz()

# Self-dispatch + object extension + overriding

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

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

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

# Self-dispatch + object extension + overriding

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

```
let shortcut = cat()  
assert (shortcut.zzz
```

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

# Self-dispatch + object extension + overriding

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

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

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

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



# 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 ↺ 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>