Rust at Sentry
7 Years Later

Armin @mitsuhiko Ronacher
What’s happening?
Who am I

- Armin Ronacher
- @mitsuhiko
- https://lucumr.pocoo.org/
- I love Open Source
- Flask, Insta, Jinja2, MiniJinja, …
What’s Sentry

- https://sentry.io/
- Error and Crash Monitoring
- Application Performance Monitoring
- Session Replays etc.
- Open Source (*)
- A Python Shop

*: some is BUSL licensed with a 3 year Apache 2 cliff
Errors and Crashes

Stack Trace

TypeError
i?.filter is not a function

mechanism: generic handled: true

```
/app/components/forms/fields/sentryMemberTeamSelectorField.tsx in ensureUserIds at line 37:21

32   const currentItem = form?.getValue(props.name) as string[] | null;
33
34   // Ensure the current value of the fields members is loaded
35   const ensureUserIds = useMemo(
36     () =>
37     currentItem?.filter(item => item.startsWith('member:')).map(user => user.slice(7)),

opened this line in GitHub

38   [currentItem]
39   );
40   useMembers({ids: ensureUserIds});
41
42   const {

Called from: ./node_modules/react-dom/cjs/react-dom.profiling.min.js in Hh.useMemo

/app/components/forms/fields/sentryMemberTeamSelectorField.tsx in SentryMemberTeamSelectorField at line 35:25

Called from: ./node_modules/react-dom/cjs/react-dom.profiling.min.js in Gh
```
Replays
## Profiles

### Transaction Details
- `base.dispatch.execute`: OrganizationReleasesEndpoint.get
- `serialize`: ReleaseSerializer
- `serialize.get.attrs`: ReleaseSerializer
- `smba_q_series`: Missing span instrumentation: Missing span instrumentation
- `http.c.l/a.qml`:

### Profile Details

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Self Time</th>
<th>Total Time</th>
<th>Bottom Up</th>
<th>Top Down</th>
<th>All Frames</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>384.10ms</td>
<td>384.10ms</td>
<td>12.4%</td>
<td>12.4%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>242.23ms</td>
<td>242.23ms</td>
<td>7.8%</td>
<td>7.8%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>190.59ms</td>
<td>190.59ms</td>
<td>6.4%</td>
<td>6.1%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>60.12ms</td>
<td>60.12ms</td>
<td>2.6%</td>
<td>2.6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>70.31ms</td>
<td>70.31ms</td>
<td>2.3%</td>
<td>2.3%</td>
<td></td>
</tr>
</tbody>
</table>

### System Frames
- `SocketIO.readinfo`
- `SocketIO.readinfo`
- `socketIO.readinfo`
- `socketIO.readinfo`
- `socketIO.readinfo`
- `socketIO.readinfo`
- `socketIO.readinfo`
- `socketIO.readinfo`
- `socketIO.readinfo`

### Transaction Details
- Model: unknown
- Manufacturer: unknown
- Classification: unknown
- OS: Linux
- OS Version: S.10.162
- Locale: unknown
### Traces

<table>
<thead>
<tr>
<th>Filter</th>
<th>Search for spans</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Web Vitals

- **Cumulative Layout Shift**
  - 0.139

- **First Contentful Paint**
  - 1.858 seconds

- **First Paint**
  - 1.858 seconds

- **Largest Contentful Paint**
  - 6.243 seconds

- **Time to First Byte**
  - 1.494 seconds

#### Request Time
- 461.600 milliseconds

#### Custom Performance Metrics

- **allresources.encoded**
  - 1.7 MiB

- **allresources.transfer**
  - 1.7 MiB

- **bundle_load**
  - 1.07s

- **connection.rtt**
  - 50.00ms
Why Rust?

- Initially personal interest
- Was really good for redistribution (sentry-cli)
- Was really nice to expose to Python
- Over time: we quite like it
- Predictable at runtime
- Tooling is really good
A Company’s Origin Story is a Legend

- Memory gets foggy over time
- Technology choices are less well informed and more incidental
- Is Jane Street really successful because of OCaml?
Rust @ Sentry Stats

- rust libraries + services: 180kLOC
- Sentry Python Monolith: 455kLOC
- Sentry TypeScript SPA: 612kLOC

Third most popular language by LOC
Why we picked it
Predictable Runtime Behavior

- Feels like Python
- No whacky memory behavior
  ○ (aside from suffering of fragmentation — hi jemallocator)
- CPU usage mostly stays predictable
- Performs well for a long time
Fits into Python

- Great at extension modules
- For us: cffi + milksnake (do not use!)
- Nowadays: PyO3 + maturin
Unexpected Wins
Rust is Outbound

- We quite actively contribute to external crates in Rust
- We rarely do so in Python
- Fork and depend on fork works well!
- Cargo as tooling changes behavior
Standardized Tooling

- One code style
- Almost universally embraced lints
- Rather well established patterns
- Jumping between code-bases feels natural
- Moving code between crates is trivial
- Painless compiler upgrades
The DX is Dope

- cargo
- rustup
- rust-analyzer
- docs (std + crate)
Types and Borrow Checker

- Modern Rust makes you a better programmer
- Types for the most part are helpful
- Borrow checker is not too annoying any more
- Makes you suspicious of a lot of Python code
Unexpected Issues
Why is there so much `memmove`?

- Large error types
- `String::clone` and friends
Large Result Types (Large Errors)

- The compiler sometimes is bad at optimizing result mapping

```rust
pub struct Error {
    repr: Box<ErrorRepr>,
}

struct ErrorRepr {
    kind: ErrorKind,
    detail: Option<Cow<'static, str>>,
    name: Option<String>,
    lineno: usize,
    span: Option<Span>,
    source: Option<Box<dyn std::error::Error + Send + Sync>>,
    #[cfg(feature = "debug")]
    debug_info: Option<crate::debug::DebugInfo>,
```
Shlemiel the Painter

- Work gets progressively harder
- Classic case: cstrings (strcat)
- But also OFFSET + LIMIT in SQL

Rust has a family of performance issues that are related

- Fear of lifetimes cause bad lookups
- String assigns become string clones
Shlemiel Paints the Entire Street For Every Dot

- Add an offset to \( N \) tokens, clone entire source for every token
Strings are ... not optimal

- Maybe we should use more `Arc<str>`?
- But `Arc<str>` is not particularly efficient
- String’s extra capacity is odd in public APIs
- Similar issue with `Vec<u8>` (broadcast to N sockets)
Errors

- Still no stack trace on `std::error::Error`
- Errors don’t have names (parsing Debug output)
Life Before Main / Registry

- We would love a supported `ctor`
- Or a way to register startup functions
From Actix to Running our own Show

- Started out with actix + actix-web
- Actor frameworks feel great
- Backpressure management is a giant pain and messy
- Moved from pre-tokio 1.0 to async/await
How I learned to love the async Bomb

- Use less async
- Use More Channels
- Embrace Backpressure
- (Cancellations are still hard)
Rust is Good For Us
Rust Community: Let's talk
Some Thoughts

- Nobody is perfect
- Building things is hard
- Good intentions can still result in bad outcomes
- Rust made it this far, let’s work on it together
- We all are more nuanced in Person