

Engineer

Rust

et kompilert språk som sjekker minnebruken for deg



Hva det er

Rust er et programmeringsspråk for arbeidet som tradisjonelt gjøres i C og C++, der et program styrer sitt eget minne og ofte lenkes inn i programvare skrevet i andre språk. Kompilatoren lager maskinkode på forhånd. Versjon 1.0 ble kunngjort den 15. mai 2015 som et språk som forener kontroll over ytelse på lavt nivå med garantier for trygg minnebruk, oppnådd uten en søppelinnsamler og uten et kjøretidsmiljø, og det er nettopp det som lar et Rust-bibliotek tre inn i stedet for et C-bibliotek.

Minnet styres gjennom eierskap (ownership), et sett regler som kompilatoren sjekker. Hver verdi har én eier om gangen, og når eieren går ut av virkeområdet sitt (scope), blir verdien droppet og minnet frigjort, på et punkt som ligger fast når programmet kompileres, og ikke når en søppelinnsamler neste gang ser etter. Kode som trenger en verdi uten å ta den, låner en referanse, og lånereglene tillater til enhver tid enten én referanse som kan endre verdien, eller et hvilket som helst antall som bare leser den, og de krever at hver referanse er gyldig. Et program som bryter en regel, kompilerer ikke. De samme reglene utelukker datakappløp (data races), så to tråder som skriver til de samme dataene uten synkronisering, gir en kompileringsfeil og ikke en feil som dukker opp av og til.

Rust har ikke unntak (exceptions). En funksjon som kan feile på en forventet måte, returnerer et `Result`, som holder enten en verdi eller en feil, og kompilatoren advarer som standard hvis den som kaller, lar resultatet være ubrukt. En feil som programmet ikke kan komme seg etter, som å lese forbi slutten av en array, går i stedet gjennom makroen `panic!`.

Statisk analyse er konservativ, så kompilatoren avviser noen programmer som ville gått fint, og noe arbeid, som å styre maskinvare eller kalle et annet språk, kan ikke sjekkes i det hele tatt. For de tilfellene finnes `unsafe`, som låser opp fem ekstra evner, blant dem å dereferere en rå peker og å kalle en funksjon merket som usikker, noe som som standard gjelder hver funksjon deklarert fra C. Det slår ikke av lånesjekkeren (borrow checker); det markerer linjene der programmereren, og ikke kompilatoren, går god for at minnet brukes trygt, så sikkerheten til disse operasjonene avhenger av at programmereren opprettholder kontraktene deres, også i den omkringliggende trygge koden. Rust-boken råder til å holde slike blokker små og pakke dem inn i et trygt grensesnitt, og slik er deler av standardbiblioteket bygd.

Det kommer en ny kompilator hver sjette uke. Endringer som ville ødelagt eksisterende kode, venter på en utgave (edition), og en ny utgave kommer omtrent hvert tredje år. Det finnes fire av dem: 2015, 2018, 2021 og 2024, og den nyeste ble stabil den 20. februar 2025 med Rust 1.85.0. Hver pakke oppgir utgaven sin i `Cargo.toml`, hver kompilator bygger alle utgavene som kom før den, og pakker med ulike utgaver lenkes sammen, så et prosjekt går over når det selv velger det. Cargo, verktøyet for avhengigheter og bygging, følger med språket, rustfmt holder kodestilen konsekvent, og biblioteker fra tredjeparter kommer som pakker som kalles crates.

Bruk begrepet i en beslutning

Bruk «Hva er Rust?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Delen brukerne aldri ser: hvor dataene ligger, hvilke regler som gjelder, og hvordan andre systemer får lov til å snakke med den.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Designet og driftet 6 ETL/ELT-pipelines — 6 ETL-pipelines på tvers av tre databasemotorer

Designet og driftet 6 ETL/ELT-pipelines Ta kontakt →

Hvorfor det er viktig

I C og C++ kompilerer en peker som brukes etter at minnet den peker på, er frigjort, eller to tråder som skriver til de samme dataene, uten protester, og så feiler programmet senere, av og til og ofte langt fra årsaken. Rust gjør slike programmer til kompileringsfeil, og fordi sjekkene kjører når programmet bygges, betaler det ferdige programmet ingenting for dem.

Uten en søppelinnsamler og uten et kjøretidsmiljø å ta med seg kan Rust-kode ligge under programvare skrevet i andre språk. Alt som kan kalle C, kan kalle en kjerne i Rust gjennom et C-grensesnitt, så én godt testet implementasjon kan betjene klienter på flere plattformer. Rust-boken nevner kommandolinjeverktøy, nettjenester, innebygde enheter og store deler av nettleseren Firefox blant bruksområdene i produksjon.

Kostnaden betales på forhånd. Koden må formes slik at kompilatoren kan se at den er trygg, og det betyr ofte å bestemme hvem som eier en verdi, i stedet for å sende referanser fritt rundt. Boken sier selv at eierskap tar tid å venne seg til. Når kompilatoren avviser et program den ikke kan verifisere, er svaret som regel en annen struktur og ikke en vei rundt sjekken.

Stabilitet er den andre halvdelen av avtalen. Siden 1.0 har endringer som bryter eksisterende kode, i stor grad vært utelukket, så en endring som kunne ødelagt gammel kode, kommer bare gjennom en utgave som en pakke velger å ta i bruk.

Slik ser det ut i praksis

I porteføljen er Rust kjernen under en native meldingsklient for macOS: den godt testede kjernen til protokollen er skrevet i Rust og eksponerer et C-grensesnitt, og klienten, 11 141 linjer Swift fordelt på 53 filer, lenker den inn som et statisk arkiv kompilert fra en låst oppstrømsrevisjon. Låsingen betyr noe fordi C-grensesnittet er kontrakten: en kjerne i bevegelse endrer kontrakten uten at én linje Swift endres. Alt C-grensesnittet returnerer, har en eierskapsregel, og å ta feil av én gir en lekkasje eller en krasj uten noen kompilatoradvarsel imellom. Derfor eier en tynn wrapper hver peker og hver streng som krysser grensen, og ikke noe lag over den ser en rå peker.

Å lenke den kjernen inn i en applikasjon betyr å levere alt den er avhengig av, og det er temaet for en lisenskontroll av 644 Rust-crates. Hver crate deklarerer lisensen sin som et uttrykk, og sjekken går gjennom hvert uttrykk i hele avhengighetsgrafen mot en tillatelsesliste ved hvert bygg. Den leser OR som et valg mellom lisenser og AND som en plikt til å overholde begge, og alt uten treff får bygget til å feile. Fordi en sjekk som aldri har feilet, ser lik ut som en som ikke kan feile, ble den med vilje gjort til å feile to ganger: én gang ved å skrive om lisensen til en crate til et vilkår listen avviser, og én gang ved å flytte den låste kjernerevisjonen uten å generere lisenskontrollen på nytt. Den leser bare deklarerte metadata og er ikke juridisk rådgivning.

Dette nettstedet har ingen egen Rust-kildekode. Det møter språket på avstand, gjennom Ruff, som er skrevet i Rust og sjekker hvert Python-skript en commit rører.

Hvor det går galt

Det meste av trøbbelet med Rust ligger i kantene av det kompilatoren sjekker: aritmetikk i et optimalisert bygg, grensen mot et annet språk, og kode som kommer inn med en avhengighet.

Aritmetikken går rundt i et utgivelsesbygg. En teller som stoppet med en panikk ved overflyt under testing, blir i det stille til et lite tall i produksjon. I feilsøkingsbygg (debug builds) sjekker Rust heltallsaritmetikk for overflyt og går i panikk. Et bygg med `--release` utelater sjekken og går rundt, så 256 i en `u8` blir 0, og boken regner det som en feil å stole på at verdien går rundt. Der en verdi kan flyte over, bør du velge oppførselen eksplisitt: `checked_*` returnerer `None`, `saturating_*` stopper ved grensen, og `wrapping_*` går rundt med vilje.

```
fn add_stock(on_hand: u8, delivered: u8) -> Option<u8> {
    // None when the sum does not fit, in debug and release alike
    on_hand.checked_add(delivered)
}
```

Minnefeilene kommer tilbake ved C-grensen. Et program der Rust-siden bygger uten én eneste advarsel, lekker minne eller krasjer inne i et kall til et annet språk. Funksjoner som er deklarert i en `extern` -blokk, er usikre å kalle fordi det andre språket ikke håndhever noen av reglene til Rust, så hvem som frigjør hver peker, er en konvensjon kompilatoren ikke kan se. Gi én wrapper ansvaret for hver peker og hver streng som krysser, skriv ned hver eierskapsregel ved siden av kallet, og eksponer bare et trygt grensesnitt over den.

En panikk når C, og prosessen avbrytes. En Rust-funksjon som kalles fra C, går i panikk, og i stedet for en feilmelding stopper hele programmet. Siden Rust 1.81, utgitt den 5. september 2024, blir prosessen avbrutt når en panikk prøver å forlate en funksjon deklarert med det vanlige `"C"` -grensesnittet. Gjør feil om til returverdier før de når grensen, og deklarer en funksjon med `"C-unwind"` bare der koden på den andre siden forventer at stakken blir avviklet (unwinding).

En rutinemessig oppdatering endrer lisensen for hele bygget. En versjonsoppgradering tre nivåer ned tar inn en crate med en lisens produktet ikke kan godta, og ingenting i bygget sier fra. Hver crate deklarerer sin egen lisens, og forpliktelsene legges sammen over hele grafen. Sjekk uttrykket til hver crate mot en tillatelsesliste ved hvert bygg, la alt uten treff feile, og bryt sjekken med vilje én gang for å bevise at den slår ut.

Les mer

- The Rust Programming Language — den offisielle boken, fra installasjon via eierskap og feilhåndtering til usikker kode.
- What Is Ownership? — de tre eierskapsreglene, og hvordan minnet frigjøres uten en søppelinnsamler.
- Unsafe Rust — de fem usikre evnene, rå pekere og kall til C gjennom `extern`.
- The Manifest Format — feltene `edition`, `rust-version` og `license` i `Cargo.toml`.
- Announcing Rust 1.0 — de opprinnelige målene, og løftet om stabilitet som begynte med 1.0.

Ordliste

Velg ut fra deres faktiske rammer

Beskriv arbeidsflyten, eksisterende grensesnitt og brukerne som trenger endringen. Vi kan definere en testbar leveranse og en overlevering som gjør teamet i stand til å vedlikeholde løsningen.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er Rust?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

welcome@engineer.company +45 50 10 61 84 København, Danmark

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/rust/