

Engineer

Rust

ett kompilerat språk som kontrollerar minnet åt dig



Vad det är

Rust är ett programspråk för det arbete som traditionellt görs i C och C++, där ett program sköter sitt eget minne och ofta länkas in i mjukvara som är skriven i andra språk. Kompilatorn översätter koden till maskinkod i förväg. Version 1.0 tillkännagavs den 15 maj 2015 som ett språk som förenar styrning av prestanda på låg nivå med garantier för säkerhet, nådda utan skräpsamlare eller körmiljö, och det är just det som låter ett bibliotek i Rust ersätta ett i C.

Minnet hanteras genom ägarskap, en uppsättning regler som kompilatorn kontrollerar. Varje värde har en enda ägare i taget, och när ägaren går ur scope släpps värdet och dess minne frigörs, vid en punkt som bestäms när programmet kompileras snarare än när en skräpsamlare nästa gång tittar efter. Kod som behöver ett värde utan att ta över det lånar en referens, och lånereglerna tillåter i varje ögonblick antingen en enda muterbar referens eller hur många oföränderliga som helst, och kräver att varje referens är giltig. Ett program som bryter mot en regel kompileras inte. Samma regler utesluter datakapplöpningar, så två trådar som skriver till samma data utan synkronisering är ett kompileringsfel snarare än ett fel som dyker upp då och då.

Rust saknar undantag. En funktion som kan misslyckas på ett väntat sätt returnerar ett `Result`, som rymmer antingen ett värde eller ett fel, och kompilatorn varnar som standard om anroparen lämnar resultatet oanvänt. En bugg som programmet inte kan återhämta sig från, som att läsa förbi slutet av en array, går i stället genom makrot `panic!`.

Statisk analys är försiktig, så kompilatorn avvisar en del program som hade fungerat, och visst arbete, som att styra hårdvara eller anropa ett annat språk, går inte att kontrollera alls. För de fallen finns `unsafe`, som låser upp fem extra förmågor, bl.a. att avreferera en rå pekare och att anropa en osäker funktion,

vilket som standard omfattar varje funktion som deklareras från C. Det stänger inte av lånekontrollen, den så kallade borrow checker. Det markerar de rader där programmeraren, inte kompilatorn, går i god för minnessäkerheten, så säkerheten i dessa operationer beror på att programmeraren upprätthåller deras kontrakt, även i den omgivande säkra koden. Rust-bokens råd är att hålla de blocken små och kapsla in dem i ett säkert gränssnitt, och det är så delar av standardbiblioteket är byggda.

En ny kompilator släpps var sjätte vecka. Ändringar som skulle bryta befintlig kod väntar på en utgåva, en edition, som kommer ungefär vart tredje år. Det finns fyra utgåvor, 2015, 2018, 2021 och 2024, och den senaste blev stabil den 20 februari 2025 med Rust 1.85.0. Varje programpaket anger sin utgåva i `Cargo.toml`, varje kompilator bygger alla utgåvor som kom före den, och programpaket med olika utgåvor länkas ihop, så ett projekt går vidare när det självt väljer. Cargo, språkets beroendehanterare och byggverktyg, följer med språket, rustfmt håller kodstilen enhetlig, och bibliotek från tredje part kommer som programpaket som kallas crates.

Använd begreppet i ett beslut

Använd ”Vad är Rust?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Den del användarna aldrig ser: där data lagras, vilka reglerna är och hur andra system får kommunicera med den.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Utformade och drev 6 ETL/ELT-pipelines — 6 ETL-pipelines över tre databasmotorer

Utformade och drev 6 ETL/ELT-pipelines Hör av er →

Varför det spelar roll

I C och C++ kompileras en pekare som används efter att dess minne har frigjorts, eller två trådar som skriver samma data, utan klagomål och fallerar senare, sporadiskt och ofta långt från orsaken. Rust gör sådana program till kompileringsfel, och eftersom kontrollerna körs när programmet byggs betalar det färdiga programmet ingenting för dem.

Eftersom Rust varken bär på en skräpsamlare eller en körmiljö kan kod i Rust ligga under mjukvara i andra språk. Allt som kan anropa C kan anropa en kärna i Rust genom ett C-gränssnitt, så en enda vältestad implementation kan betjäna

klienter på flera plattformar. Rust-boken räknar upp kommandoradsverktyg, webbtjänster, inbyggda enheter och stora delar av webbläsaren Firefox bland de användningar som finns i produktion.

Kostnaden betalas i förväg. Koden måste formas så att kompilatorn kan se att den är säker, vilket ofta innebär att bestämma vem som äger ett värde i stället för att skicka runt referenser fritt, och boken säger själv att ägarskap tar tid att vänja sig vid. När kompilatorn vägrar ett program som den inte kan verifiera är svaret oftast en annan struktur snarare än en väg runt kontrollen.

Stabiliteten är den andra halvan av överenskommelsen. Sedan 1.0 är ändringar som bryter befintlig kod i stort sett uteslutna, så en ändring som skulle kunna bryta gammal kod kommer bara genom en utgåva som ett programpaket självt väljer att gå över till.

Så ser det ut i praktiken

I portföljen är Rust kärnan under en native meddelandeklient för macOS: den vältestade kärnan i protokollet är skriven i Rust och exponerar ett C-gränssnitt, och klienten, 11 141 rader Swift i 53 filer, länkar in den som ett statiskt arkiv kompilerat från en fastlåst uppströmsrevision. Låsningen spelar roll eftersom C-gränssnittet är kontraktet: en kärna i rörelse ändrar kontraktet utan att en enda rad Swift ändras. Allt som C-gränssnittet returnerar har en ägarskapsregel, och ett misstag i någon av dem ger en läcka eller en krasch utan en enda kompilatorvarning på vägen, så en tunn wrapper äger varje pekare och varje sträng som korsar gränsen, och inget lager ovanför den ser en rå pekare.

Att länka in den kärnan i en applikation innebär att leverera allt som den är beroende av, och det är ämnet för en licensgranskning av 644 Rust-crates. Varje crate deklarerar sin licens som ett uttryck, och granskningen kontrollerar varje uttryck i hela beroendegrafen mot en tillåtelselista vid varje bygge, där OR läses som ett val mellan licenser och AND som en skyldighet att följa båda. Allt som inte matchar får bygget att fallera. Eftersom en kontroll som aldrig har fallerat ser likadan ut som en som inte kan fallera, fick den fallera med flit, två gånger: en gång genom att en crates licens skrevs om till ett villkor som listan avvisar, och en gång genom att den fastlåsta kärnrevisionen flyttades utan att granskningen genererades om. Den läser bara deklarerade metadata och är ingen juridisk rådgivning.

Den här webbplatsen har ingen egen källkod i Rust. Den möter språket på avstånd, genom Ruff, som är skrivet i Rust och kontrollerar varje Python-skript som en commit rör.

Var det går fel

Det mesta besväret med Rust sitter vid kanterna av det som kompilatorn kontrollerar: aritmetik i ett optimerat bygge, gränsen mot ett annat språk och kod som kommer in med ett beroende.

Aritmetik slår runt i ett releasebygge. En räknare som fick panik vid ett överspill under testningen blir i tysthet ett litet tal i produktion. I debugbyggen kontrollerar Rust heltalsaritmetik efter överspill och får panik. Ett bygge med `--release` utelämnar kontrollen och slår runt, så 256 i en `u8` blir 0, och boken betraktar det som ett fel att förlita sig på det omslaget. Där ett värde kan spilla över ska beteendet väljas uttryckligen: `checked_*` returnerar `None`, `saturating_*` stannar vid gränsen och `wrapping_*` slår runt med avsikt.

```
fn add_stock(on_hand: u8, delivered: u8) -> Option<u8> {
    // None när summan inte får plats, i debug- och releasebyggen lika
    on_hand.checked_add(delivered)
}
```

Minnesbuggar kommer tillbaka vid gränsen mot C. Ett program vars Rust-sida byggs utan en enda varning läcker minne eller kraschar inne i ett främmande anrop. Funktioner som deklareras i ett block med `extern` är osäkra att anropa eftersom det andra språket inte upprätthåller någon av Rusts regler, så vem som frigör varje pekare är en konvention som kompilatorn inte kan se. Ge en enda wrapper ansvaret för varje pekare och sträng som korsar gränsen, skriv ned varje ägarskapsregel bredvid anropet och exponera bara ett säkert gränssnitt ovanför det.

En panik når C och processen avbryts. En Rust-funktion som anropas från C får panik, och i stället för ett fel stannar hela programmet. Sedan Rust 1.81, som släpptes den 5 september 2024, avbryter en panik som försöker lämna en funktion deklarerad med det vanliga gränssnittet `"C"` hela processen. Gör fel till returvärden innan de når gränsen, och deklarera en funktion med `"C-unwind"` bara där koden på andra sidan förväntar sig att stacken rullas upp.

En rutinmässig uppdatering ändrar licensen för hela bygget. En versionshöjning tre nivåer ned drar in en crate vars licens produkten inte kan godta, och ingenting i bygget säger ifrån. Varje crate deklarerar sin egen licens, och skyldigheterna läggs ihop över hela grafen. Kontrollera varje crates uttryck mot en tillåtelselista vid varje bygge, låt allt som inte matchar få bygget att fallera och bryt kontrollen med flit en gång för att bevisa att den slår till.

Läs mer

- The Rust Programming Language — den officiella boken, från installation via ägarskap och felhantering till osäker kod.
- What Is Ownership? — de tre reglerna för ägarskap och hur minnet frigörs utan en skräpsamlare.
- Unsafe Rust — de fem osäkra förmågorna, råa pekare och att anropa C genom `extern`.
- The Manifest Format — fälten `edition`, `rust-version` och `license` i `Cargo.toml`.
- Announcing Rust 1.0 — de ursprungliga målen och det stabilitetslöfte som började med 1.0.

Ordlista

Välj utifrån era faktiska förutsättningar

Beskriv arbetsflödet, befintliga gränssnitt och användarna som behöver förändringen. Vi kan definiera en testbar leverans och en överlämning som gör att teamet kan underhålla lösningen.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Rust?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/rust/