Engineer

セマンティックバージョニング

互換性を壊す変更をバージョン番号で示すための規則

どのようなものか

セマンティックバージョニングは、通常SemVerと略され、リリースにどのような変更が含まれるかを番号自体で示すための規則です。Tom Preston-Wernerが最初に書いたこの仕様は、現在バージョン2.0.0で、MAJOR.MINOR.PATCHという形式の三つの数値を使います。公開APIの互換性を壊すリリースではメジャー番号を上げ、後方互換性を保って機能を追加する場合はマイナー番号を上げ、APIを変えずにバグだけを修正する場合はパッチ番号を上げます。ある番号を上げると、その右側の番号はゼロに戻します。また、それぞれを文字列ではなく数値として比較するので、`1.10.0` は `1.9.0` の後になります。

すべての前提になる要件は一つです。番号が表すのは公開APIの変更だけなので、プロジェクトはコードか文書で公開APIを宣言しなければなりません。一度リリースしたバージョンの内容は変更してはならず、修正するなら新しいバージョンにします。メジャーバージョンがゼロの `0.y.z` は初期開発用で、何もかもいつでも変わる可能性があり、APIを安定していると見なすべきではありません。`1.0.0` は公開APIが定義される時点であり、それ以降の番号はAPIの変化に従います。

形式には、必要に応じて二種類のラベルを追加できます。プレリリースはハイフンの後に付け、対応する正式リリースより前に並びます。したがって、`1.0.0-alpha` 、`1.0.0-alpha.1` 、`1.0.0-beta` 、`1.0.0-rc.1` 、最後に `1.0.0` という順序になります。ビルドメタデータは、`1.0.0+20130313144700` のようにプラス記号の後に付けますが、バージョンの順序を決めるときには無視するため、その部分だけが違う二つのバージョンの順位は同じです。Gitのタグ `v1.2.3` のように先頭に付くvは、セマンティックバージョンの一部ではありません。仕様では、バージョン番号を示す一般的な方法と説明しています。Goではこれが必須で、正規のGoモジュールバージョンは、文字vにセマンティックバージョンを続けたものです。

概念を判断に生かす

「セマンティックバージョニングとは？」を実際の判断に生かしましょう。その概念を、開発または購入するシステムに結び付けて考えるお手伝いをします。 変更を検証し、リリースし、元に戻せる仕組みを整え、金曜の深夜ではなく火曜の午後にお客様へ届けられるようにします。

サービスを見る お問い合わせ →

用語の背後にある前提を検証する

解説とリンク先の実績を参考に、プロジェクトについての質問を整理してください。ツールを選ぶ前に、前提を一緒に検討できます。 GitHub ActionsでGIS SaaSのデプロイを自動化 — デプロイ、データ、報告を手作業から解放

GitHub ActionsでGIS SaaSのデプロイを自動化 お問い合わせ →

なぜ重要なのか

仕様は、依存関係地獄と呼ぶ問題から説明を始めます。依存関係の指定を厳しくしすぎると、あるパッケージを更新するには、それに依存するすべてのパッケージも新しいバージョンをリリースしなければならない、バージョンロックに陥ります。緩くしすぎると、互換性を期待できる確かな根拠もなく将来のリリースを受け入れる、仕様のいう「バージョンの無節操」が起きます。番号の意味に合意があれば、プロジェクトは受け入れる範囲を指定できます。3.1.0で初めて追加された関数を必要とするライブラリは、3.1.0以上4.0.0未満を受け入れ、要件を誰も編集することなく、その後の修正や機能追加を取り込めます。

パッケージマネージャーは、その約束に基づいて確認なしに動作します。Rustのパッケージマネージャーでは、`"1.2.3"` のような単純な指定はキャレット要件であり、1.2.3以上2.0.0未満を意味します。Cargoは `Cargo.lock` を書くとき、その範囲内の最大のバージョンを選びます。2.0.0を非互換と扱うため、2.0.0の公開後も1.x系にとどまります。npm向けのセマンティックバージョン処理ツールと名乗るnode-semverでは、`^1.2.3` は `>=1.2.3 <2.0.0-0` を意味し、`~1.2.3` はパッチ単位の変更だけ、つまり `>=1.2.3 <1.3.0-0` を許可します。Goは、この規則をインポートパスに組み込んでいます。メジャーバージョン2以降は、モジュールのパスの末尾に `/v2` などの接尾辞が必要なので、互換性のない二つのメジャーバージョンは、異なるインポートパスになります。そのうえでGoの最小バージョン選択は、各モジュールについて、いずれかの要件が求めるバージョンのうち最も高いものを選びます。それが、すべての要件を満たす最小のバージョンです。結果はロックファイルに保存されず、モジュール対応の各コマンドが毎回計算し直します。

こうした仕組みがあっても、番号が保証になるわけではありません。これはパッケージの公開者がする約束であり、何を互換性を壊す変更と見なすかという、その人たちの判断次第です。そのため、Cargoは変更をメジャー、マイナー、互換性を壊す可能性があるものに分類した、詳細な互換性ガイドを公開しています。その大半は、昨日までコンパイルできたコードが、今日コンパイルできなくなる

変更を扱っています。ガイド自体も、ほぼどの変更にも実行時の挙動へ影響するリスクがあり、その変更を非互換と見なすかは、通常、保守担当者の判断になると述べています。

実際の活用例

架空のRustライブラリtallyのリリースを追ってみます。0.4.2は、まだ初期開発の段階です。1.0.0で、公開APIが安定したと宣言します。1.0.1では丸め処理のバグを修正し、パッチリリースになります。1.1.0では `parse_strict` 関数を追加し、従来の `parse` を非推奨にします。仕様は、非推奨化にはマイナーリリースを要求し、関数を削除する前に少なくとも一度、そのようなリリースを出すよう求めています。2.0.0では `parse` を削除するため、メジャーリリースになります。2.1.0-rc.1は、次のマイナーバージョンのプレリリースです。

tallyと、他の三つの架空のクレートに依存するアプリケーションは、`Cargo.toml` に要件を書きます。コメントは、Cargoが各行から読み取る範囲を示しています。

```
[dependencies]
tally = "1.0.1"         # >=1.0.1, <2.0.0: takes 1.1.0, never 2.0.0
ledger = "0.4.2"        # >=0.4.2, <0.5.0: 0.5.0 counts as breaking
stamp = "~1.2.3"        # >=1.2.3, <1.3.0: patch releases only
quill = "=2.1.0-rc.1"   # exactly this pre-release
```

`ledger` の行は、仕様ではなくツールの慣習を示しています。仕様では1.0.0未満はいつでも変わり得るとする一方、Cargoは、左から最初のゼロでない番号を、互換性を壊す変更を表すものとして扱います。Cargoは、要件にプレリリースが明記されていない限り、それを候補から除外するため、`"2.0"` という要件ではtallyのリリース候補は選ばれません。最終行の厳密な `=` 形式は、プレリリースに依存しながら、プレリリース間の非互換変更を避けるための指定です。Cargoの文書も、こうした場合には、バージョンを完全に指定する要件が妥当になり得るとしています。依存関係の解決で確定したバージョンは `Cargo.lock` に記録され、`cargo update` で、これらの範囲内のバージョンへ更新できます。`--locked` を指定すると、ロックファイルの変更が必要なビルドは、代わりにエラーを返します。

番号が扱わないのは、APIの外側にある事柄すべてです。2026年9月に実施した644個のRustクレートのライセンス監査では、ビルドのたびに、各クレートのライセンスを許可リストに照らして検査します。依存関係グラフの奥で日常的なバージョン更新が起きるだけでも、アプリケーションをどの条件で配布できるか

が、誰も全文を読まないロックファイルの中で、気付かれずに変わり得るためです。検査が機能することを確かめるため、意図的に二度失敗させました。一つはクレートのライセンス式を書き換えるテストです。これは、バージョン間で依存先の利用条件が変わる場合に相当し、クレート自体は新しくないため、人のレビューでは見落とされます。もう一つは、監査を再生成せずに、固定したコアのリビジョンを変更するテストです。セマンティックバージョンは、1.4.0が1.0.0を対象に書かれたコードで動くことを意図しているとは示せます。しかし、引き続き配布してよいかどうかは示せません。

問題が起きるところ

マイナーリリースで互換性を壊します。 うっかり起きることも、何を非互換と見なすかの考え方が違って起きることもあります。Cargoのガイドには、無害に見えてそうではない変更が挙げられています。公開enumにバリアントを追加すると、最初から `#[non_exhaustive]` が付いていない限り、すべてのバリアントを網羅するマッチを行うコードが壊れます。また、globインポートが関わるまれな状況では、新しい公開項目を追加するだけでもコンパイルエラーが起きます。誤りに気付いた場合に仕様が示す対処は、互換性を回復する新しいマイナーバージョンをリリースし、公開済みのリリースを変更することは決してせず、どのバージョンに問題があったか利用者に伝えることです。利用する側では、コミットされたロックファイルと、更新のたびに実行するテストによって、本番で発覚する前に検出します。

何年も0.xのままです。 仕様上、メジャーバージョンがゼロの間は、安定したAPIという約束はありません。FAQは率直に、「本番環境で使われているソフトウェアなら、おそらくすでに1.0.0にしておくべきでしょう」と述べています。ツールは、独自の慣習でこの隙間を埋めています。Cargoとnode-semverは、左から最初のゼロでない番号を非互換変更の番号とするため、 `^0.2.3` が受け入れるのは、その後の0.2.xリリースだけであり、 `^0.0.3` は一切の更新を受け入れません。何年も0.xのままのライブラリには、それに依存する利用者がいるのに、明確な約束がありません。APIが頼りにされるようになったら、1.0.0をリリースします。変えるのは番号であって、コードではありません。

番号から読み取れる以上のことを読み込みます。 仕様がいう互換性は、公開APIについてのものです。Cargoのガイドは、クレートが必要とするRustの最低バージョンを上げる変更を、互換性を壊す可能性があるものに分類し、一般にはマイナー変更として扱うよう勧めています。そのため、マイナー更新でも、古いツールチェーンではビルドできなくなることがあります。 `package.rust-version` を設定すれば、必要に応じてCargoのリゾルバーが、そのクレートの古いバージョン

を試せるようになります。ライセンス条件、性能、APIが文書化していない挙動は、そもそも番号の対象外です。マイナーリリースだからと推測せず、ライセンス監査とテストで別々に確認します。

一つのビルドに二つのメジャーバージョンが入ります。 二つの依存先が、同じクレートの非互換なバージョン、例えば0.6と0.7を必要とすると、Cargoは両方をビルドします。名前が一致していても、コンパイラーはそれらの型を別物として扱います。通常はコンパイルエラーになりますが、実行時にしか分からないこともあります。Cargoの文書には、オブジェクトを生成したライブラリは1.0.0を使っているのに、呼び出し側は2.0.0の型へダウンキャストするため、失敗する例があります。`cargo tree -d` は、重複するバージョンと、それぞれがどこから来るかを一覧にします。ライブラリの作者は、Cargoの文書がsemver trickと呼ぶ方法で、メジャーリリースの影響を和らげられます。新しいメジャーバージョンを公開した後、古いメジャーバージョンのポイントリリースを出し、新しい方の型を再エクスポートします。

参考資料

- Semantic Versioning 2.0.0 — 仕様とFAQです。誤って互換性を壊す変更をリリースした場合の対処も含みます。
- SemVerの互換性 — Rustライブラリのどの変更がメジャー、マイナー、互換性を壊す可能性のあるものに当たるかを、例付きで説明するCargo Bookのガイドです。
- 依存関係の指定 — デフォルト、キャレット、チルダ、比較によるCargoの要件構文と、プレリリースの照合方法を説明しています。
- 依存関係の解決 — Cargoによるバージョンの選択、ロックファイルに保存される内容、一つのグラフに非互換な二つのバージョンが入る危険を説明しています。
- node-semver — npm向けのセマンティックバージョン処理ツールです。キャレット、チルダ、プレリリース範囲の規則を説明しています。
- Goモジュールリファレンス — メジャーバージョンの接尾辞、インポート互換性の規則、最小バージョン選択を説明しています。

用語集

実際の制約に合わせて選ぶ

デプロイの流れ、運用上のリスク、緊急対応の担当者をお知らせください。検査、復旧手段、明確な責任の所在を伴う変更の範囲を定められます。

料金を見る お問い合わせ →

用語を実務に結び付ける

「セマンティックバージョニングとは？」がお客様の組織にどう当てはまるかをご相談ください。現在のシステムの具体例が一つあれば、話を始められます。

サービスを見る

welcome@engineer.company +45 50 10 61 84 デンマーク、コペンハーゲン

このページを持ち歩く

連絡先カード

心を込めて丁寧に制作： D.

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

著作権 © 2025 – 現在 · Engineer ApS · すべての権利を留保します。

印刷元： https://engineer.company/ja/glossary/semantic-versioning/