Barcode.Go

Go バーコード生成ライブラリ

マニュアル

バージョン 1.0

有限会社 パオ・アット・オフィス

https://www.pao.ac/

目次

サーバーサイドで、18種のバーコードを自在に生成。

ユーザーズマニュアル

バージョン 1.0 — 2026年2月

有限会社 パオ・アット・オフィス

https://www.pao.ac/

1.1 Barcode.Goとは

物流倉庫のピッキングリスト、医療現場の検体ラベル、ECサイトの出荷伝票――。

バーコードはあらゆる業務システムの「最後の1ミリ」を担っています。

Barcode.Go は、そのバーコードを Pure Go で生成するライブラリです。

C言語バインディングも、外部コマンドも、一切不要。go　get　一発でインストールでき、1次元・2次元あわせて全18種 のバーコードを、PNG画像・JPEG画像・SVGベクター で出力できます。

Go　のシンプルさと高い並行性をそのまま活かせるため、REST APIサーバーに組み込めば、毎秒数千枚のバーコードを生成することも可能です。

```
qr := barcode.NewQRCode(barcode.FormatPNG)
qr.Draw("https://www.pao.ac/", 300)
base64, _ := qr.GetImageBase64()
```

たった3行で、QRコードが Base64 文字列になって返ってきます。

1.2 特長

特長	説明
Pure Go	外部依存ゼロ（golang.org/x/image のみ）。CGO不要、クロスコンパイルも自由自
サーバーサイド特化	net/http と組み合わせるだけで REST API バーコードサーバーが完成
PNG / JPEG / SVG	画面表示にはPNG、印刷にはSVG、サムネイルにはJPEG。用途で使い分け可能
18種のバーコード	1D・2D・GS1・郵便まで、業務で必要なバーコードを網羅
高い並行性	goroutine セーフ。各インスタンスが独立しているため、並行生成もロックフリー
豊富なカスタマイズ	色、テキスト、バー幅調整、均等割付まで細かく制御可能

1.3 対応バーコード一覧

1次元バーコード（11種類）

バーコード	コンストラクタ	どんなところで使われている？
Code39	NewCode39()	工場の部品ラベル、軍事規格（MIL-STD）にも採用
Code93	NewCode93()	Code39の高密度版。郵便・物流で活用
Code128	NewCode128()	物流の標準。ASCII全文字をエンコード可能
GS1-128	NewGS1_128()	医薬品・物流。ロット番号や有効期限をAIで管理
NW-7 (Codabar)	NewNW7()	宅配便の送り状、図書館の貸出管理でおなじみ
ITF	NewITF()	段ボール箱の集合包装用。数字ペアで高密度
Matrix 2of5	NewMatrix2of5()	工業用途。数字のみのシンプルな構成
NEC 2of5	NewNEC2of5()	日本の工業現場で使われるバリエーション
JAN-8 (EAN-8)	NewJAN8()	小さな商品用。ガムやキャンディーのパッケージに
JAN-13 (EAN-13)	NewJAN13()	日本の商品バーコードの標準。スーパーのレジで毎日活躍
UPC-A	NewUPC_A()	北米の商品コード。12桁
UPC-E	NewUPC_E()	UPC-Aの短縮版。小さなパッケージに

GS1 DataBar（3種類）

バーコード	コンストラクタ	どんなところで使われている？
GS1 DataBar 標準型	NewGS1DataBar14()	スーパーの青果・精肉売り場。重量や価格を直接エンコー
GS1 DataBar 限定型	NewGS1DataBarLimited()	小型商品向けのコンパクト版
GS1 DataBar 拡張型	NewGS1DataBarExpanded()	可変長データ対応。クーポンや特売情報も格納

郵便バーコード（1種類）

バーコード	コンストラクタ	どんなところで使われている？
郵便カスタマバーコード	NewYubinCustomer()	郵便物の住所バーコード。自動区分機で高速仕分け

2次元バーコード（3種類）

バーコード	コンストラクタ	どんなところで使われている？
QRコード	NewQRCode()	URL、決済、名刺交換——。日本発、世界で最も普及した
DataMatrix	NewDataMatrix()	電子部品の超小型マーキング。GS1ヘルスケアでも標準
PDF417	NewPDF417()	運転免許証、搭乗券。大容量データを1本に集約

2.1 PNG画像出力 — Base64でそのまま返せる

Draw() でバーコードを生成し、GetImageBase64() を呼ぶだけで Base64エンコードされたPNG画像が返ってきます。HTMLの <img> タグにそのまま埋め込めるので、REST APIの戻り値としてそのままクライアントに返せます。

```
qr := barcode.NewQRCode(barcode.FormatPNG)
qr.Draw("Hello World", 300)
base64, _ := qr.GetImageBase64()

// base64  "..."
//  JSON
```

PNGが向いている場面:

- REST APIでBase64文字列をクライアントに返す
- 画面上でのプレビュー表示
- 固定解像度での画像出力

2.2 SVGベクター出力 — 拡大しても美しい

出力形式を FormatSVG にするだけで、ベクター形式のSVG文字列が得られます。

どれだけ拡大しても線がぼやけないため、印刷用途に最適 です。

```
code := barcode.NewCode128(barcode.FormatSVG)
code.Draw("Hello-2026", 400, 100)
svg, _ := code.GetSVG()

// svg  "<svg xmlns="...">...</svg>"
// HTML
```

SVGが向いている場面:

- ラベル印刷（拡大しても劣化しない）
- PDF生成時の高品質バーコード埋め込み
- ファイルサイズを小さく抑えたい場合

> ヒント: 同じバーコードオブジェクトで SetOutputFormat() を切り替えれば、PNG版とSVG版の両方を生成できます。プレビューはPNG、ダウンロードはSVG、という使い分けも簡単です。

2.3 バイト列出力 — ファイル保存やHTTPレスポンスに

GetImageMemory() を使えば、PNG/JPEG のバイト列（[]byte）を直接取得できます。ファイル保存はもちろん、http.ResponseWriter に直接書き込んでバイナリレスポンスとして返すことも可能です。

```
code := barcode.NewCode39(barcode.FormatPNG)
code.Draw("HELLO", 400, 100)
imageBytes := code.GetImageMemory()

//
os.WriteFile("barcode.png", imageBytes, 0644)

//  HTTP
w.Header().Set("Content-Type", "image/png")
w.Write(imageBytes)
```

2.4 カスタマイズ — 色もテキストも思いのままに

色を変える

前景色（バーの色）と背景色を自由に指定できます。

透明度（アルファ値）にも対応しているので、背景を透明にすることも可能です。

```
//
code.SetForegroundColor(0, 0, 128, 255)
code.SetBackgroundColor(255, 255, 240, 255)

//
code.SetBackgroundColor(0, 0, 0, 0)
```

テキスト表示を調整する

バーコード下部のテキスト（ヒューマンリーダブル）は、表示・非表示だけでなく、サイズや配置まで細かく調整できます。

```
code.ShowText = true                    //
code.SetTextFontScale(1.2)              //
code.SetTextVerticalOffsetScale(0.5)    //
code.TextEvenSpacing = true             // 1
```

> ヒント: TextEvenSpacing = true にすると、テキストが各バーの真下に揃って配置されます。見た目がすっきりするので、一般的な1Dバーコードではおすすめです。

バー幅を微調整する（印刷のにじみ対策）

実際に印刷すると、インクのにじみで黒バーが太くなることがあります。

バーコードリーダーの読み取り精度が落ちてしまう場合は、この機能で補正しましょう。

```
code.SetPxAdjustBlack(-1)  // 1px
code.SetPxAdjustWhite(1)   // 1px
```

幅ぴったり描画

指定した幅にバーコードをぴったり収めたい場合に使います。

```
code.SetFitWidth(true)   //
code.SetFitWidth(false)  //
```

導入はとてもシンプルです。Go Modules に対応しているので、go get 一発で完了します。

3.1 ダウンロード

https://www.pao.ac/barcode.go/#download

パッケージ	内容	こんな方に
Easy 2 Steps	QRコード生成の最小RESTサーバー	まずは動かしてみたい方
All-in-One	全18種対応のフル機能RESTサーバー	本格的に評価したい方
WASM版	ブラウザで動くデモ	Go以外の環境でも試したい方

3.2 go get でインストール

```
go get github.com/pao-company/barcode-go
```

これだけです。CGO不要なので、クロスコンパイル環境でもそのまま使えます。

3.3 ファイル構成

Easy 2 Steps サンプル

```
barcode_go_easy2steps/
├── main.go               # QR100
├── go.mod
├── templates/
│   └── index.html         # Web
└── barcode-go/            #
    ├── barcode.go
    ├── base.go
    ├── qr.go
    └── ...
```

起動方法

```
cd barcode_go_easy2steps
go run main.go
# → http://localhost:5700
```

ブラウザで開くと、QRコードの生成画面が表示されます。テキストを入力して「生成」ボタンを押すだけ。

> ヒント: サンプルはすべてGo標準ライブラリの net/http だけで構成されています。フレームワーク不要で動くため、Go初心者にも優しい設計です。

ここでは、コピー＆ペーストですぐ動くサンプルを紹介します。

4.1 QRコードをPNGで生成

```
package main

import (
    "fmt"
    "os"

    barcode "github.com/pao-company/barcode-go"
)

func main() {
    // Step 1: QR
    qr := barcode.NewQRCode(barcode.FormatPNG)
    qr.SetErrorCorrectionLevel(barcode.QREccM) // : Medium

    // Step 2:
    err := qr.Draw("https://www.pao.ac/", 300)
    if err != nil {
        fmt.Println("Error:", err)
        return
    }

    // Base64 — HTML
    base64, _ := qr.GetImageBase64()
    fmt.Println("Base64:", base64[:50], "...")

    //
    imageBytes := qr.GetImageMemory()
    os.WriteFile("qr.png", imageBytes, 0644)
    fmt.Println("Saved: qr.png")
}
```

4.2 1Dバーコードを SVGで生成

```
package main

import (
    "fmt"
    "os"

    barcode "github.com/pao-company/barcode-go"
)

func main() {
    code := barcode.NewCode128(barcode.FormatSVG)
    code.ShowText = true
    code.TextEvenSpacing = true

    err := code.Draw("Hello-2026", 400, 100)
    if err != nil {
        fmt.Println("Error:", err)
        return
    }

    svg, _ := code.GetSVG()
    os.WriteFile("code128.svg", []byte(svg), 0644)
    fmt.Println("Saved: code128.svg")
}
```

4.3 REST APIサーバーで提供

Go の真骨頂——サーバーサイドでバーコードを生成し、APIとして提供する例です。

```
package main

import (
    "encoding/json"
    "log"
    "net/http"

    barcode "github.com/pao-company/barcode-go"
)

func main() {
    http.HandleFunc("/api/qr", func(w http.ResponseWriter, r *http.Request) {
        code := r.URL.Query().Get("code")
        if code == "" {
            code = "https://www.pao.ac/"
        }

        qr := barcode.NewQRCode(barcode.FormatPNG)
        qr.Draw(code, 300)
        b64, _ := qr.GetImageBase64()

        w.Header().Set("Content-Type", "application/json")
        json.NewEncoder(w).Encode(map[string]string{
            "base64": b64,
        })
    })

    log.Println("→ http://localhost:8080/api/qr?code=Hello")
    log.Fatal(http.ListenAndServe(":8080", nil))
}
```

curl http://localhost:8080/api/qr?code=Hello で JSON が返ってきます。フロントエンドから fetch するだけでバーコードが表示できます。

ここからは、バーコードの種類ごとに実践的なサンプルを紹介します。

各バーコードが「どんな場面で使われているか」も添えていますので、用途に合ったバーコードを選ぶ参考にしてください。

5.1 1次元バーコード — 物流・工業の定番

Code39 — 工場で最も古くから使われるバーコード

英数字と一部の記号を表現できます。スタート/ストップコード（*）で囲まれるのが特徴です。

```
code := barcode.NewCode39(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
code.ShowStartStop = true   // *HELLO123*
err := code.Draw("HELLO123", 400, 100)
```

入力可能: 数字（0-9）、英大文字（A-Z）、記号（- . $ / + % スペース）

Code93 — Code39の高密度版

Code39と同じ文字を、より狭いスペースでエンコードできます。さらにASCII全文字に対応。

```
code := barcode.NewCode93(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
err := code.Draw("Hello123", 400, 100)
```

Code128 — 物流業界の標準

ASCII全文字に対応し、数字は高密度でエンコードできるため、物流伝票で広く使われています。コードモードは通常Code128Auto にしておけば、最短幅になるよう自動で最適化されます。

```
code := barcode.NewCode128(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
code.CodeMode = barcode.Code128Auto  // AUTO / A / B / C
err := code.Draw("Hello123", 400, 100)
```

コードモード	定数	説明
AUTO	Code128Auto	自動で最短幅に最適化（おすすめ）
A	Code128CodeA	制御文字 + 数字 + 英大文字
B	Code128CodeB	数字 + 英大文字 + 英小文字 + 記号
C	Code128CodeC	数字のみ（2桁ずつ高密度エンコード）

> ヒント: AUTO モードでは、データの内容を解析して CODE-A / B / C を動的に切り替え、最短幅になるよう自動最適化します。特別な理由がなければ AUTO のままで問題ありません。

NW-7 (Codabar) — 宅配便の送り状でおなじみ

先頭と末尾にスタート/ストップコード（A/B/C/D）を付けるのがルールです。

```
code := barcode.NewNW7(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
code.ShowStartStop = true
err := code.Draw("A1234567A", 400, 100)
```

ITF — 段ボール箱でよく見るバーコード

Interleaved 2 of 5。バーとスペースを交互に使って2桁ずつエンコードするため、高密度です。入力は偶数桁である必要があります。

```
code := barcode.NewITF(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
err := code.Draw("123456", 400, 100)
```

5.2 2次元バーコード — 大容量データを小さな面積に

QRコード — 日本発、世界で最も使われている2Dコード

URL、テキスト、連絡先——なんでも格納できる万能選手です。日本語もそのままエンコードできます。

```
qr := barcode.NewQRCode(barcode.FormatPNG)
qr.SetStringEncoding("utf-8")
qr.SetErrorCorrectionLevel(barcode.QREccM)  // L(7%) / M(15%) / Q(25%) / H(30%)
qr.SetVersion(0)                             // 0=
err := qr.Draw("https://www.pao.ac/", 300)
```

誤り訂正レベル	定数	復元能力	こんなときに
L	QREccL	約7%	データ量を最優先したい
M	QREccM	約15%	一般的な用途（おすすめ）
Q	QREccQ	約25%	やや過酷な環境（汚れ・傷）
H	QREccH	約30%	ロゴを重ねたい場合にも

DataMatrix — 極小マーキングの世界標準

電子部品やヘルスケア製品の超小型マーキングに使われています。小さくても大容量。

```
dm := barcode.NewDataMatrix(barcode.FormatPNG)
dm.SetStringEncoding("utf-8")
dm.SetCodeSize(barcode.DxSzAuto)
dm.SetEncodeScheme(barcode.DxSchemeAutoBest)
err := dm.Draw("Hello World", 200)
```

PDF417 — 運転免許証にも使われている大容量コード

1次元バーコードを積み重ねたような構造で、テキスト・数字・バイナリの大量データを格納できます。

```
pdf := barcode.NewPDF417(barcode.FormatPNG)
pdf.SetStringEncoding("utf-8")
pdf.SetErrorLevel(barcode.PDF417ErrorLevel2)
pdf.SetColumns(3)
pdf.SetAspectRatio(3.0)
pdf.SetYHeight(3)
err := pdf.Draw("Hello World", 400, 100)
```

5.3 GS1系バーコード — 流通のインフラ

GS1-128 — AI（アプリケーション識別子）で情報を構造化

ロット番号、有効期限、重量——さまざまな情報をAIコードで構造化して格納します。

```
gs1 := barcode.NewGS1_128(barcode.FormatPNG)
gs1.ShowText = true
gs1.TextEvenSpacing = true
err := gs1.Draw("[01]04912345123459[10]ABC123", 500, 120)
```

特殊文字	意味
[AI]	AIを角括弧で表記（テキスト表示で括弧表示）
{FNC1}	ファンクション1（可変長フィールドの区切り）

コンビニバーコード（標準料金代理収納）

公共料金の払込票に印字されているあのバーコードです。DrawConvenience() で生成します。

```
gs1 := barcode.NewGS1_128(barcode.FormatPNG)
gs1.ShowText = true
err := gs1.DrawConvenience(
    "{FNC1}9191234500000000000000452087500401310029500", 500, 150,
)
```

GS1 DataBar 標準型 — 青果・精肉売り場で活躍

重量や価格情報をコンパクトにエンコードできるバーコードです。

```
db := barcode.NewGS1DataBar14(barcode.FormatPNG, barcode.Omnidirectional)
db.ShowText = true
err := db.Draw("1234567890128", 200, 80)
```

シンボルタイプ	定数	説明
標準型	Omnidirectional	どの方向からでも読み取り可能
二層型	Stacked	省スペース
標準二層型	StackedOmnidirectional	二層かつ全方向対応

GS1 DataBar 拡張型 — クーポンや特売情報も格納可能

可変長データに対応し、多層型（スタック）にも対応しています。

```
db := barcode.NewGS1DataBarExpanded(barcode.FormatPNG, barcode.Unstacked)
db.ShowText = true
err := db.Draw("[01]90012345678908[10]ABC123", 400, 80)

//
dbStacked := barcode.NewGS1DataBarExpanded(barcode.FormatPNG, barcode.StackedExp)
dbStacked.SetNoOfColumns(4)
err = dbStacked.Draw("[01]90012345678908[10]ABC123", 300, 100)
```

5.4 商品・郵便バーコード — 身の回りのバーコード

郵便カスタマバーコード — 郵便物を高速仕分け

長さの異なる4種類のバー（ロング・セミアッパー・セミロウワー・タイミング）で住所情報を表現します。幅はバーの本数から自動計算されるため、高さだけを指定 します。

```
yubin := barcode.NewYubinCustomer(barcode.FormatPNG)
err := yubin.Draw("27500263-29-2-401", 25)
```

入力形式: 郵便番号7桁 + 住所表示番号（ハイフン区切り可）

JAN/EAN バーコード — スーパーのレジで毎日活躍

```
// JAN-13
jan13 := barcode.NewJAN13(barcode.FormatPNG)
jan13.ShowText = true
jan13.ExtendedGuard = true          //
jan13.TextEvenSpacing = false       //
err := jan13.Draw("491234567890", 300, 100)

// JAN-8
jan8 := barcode.NewJAN8(barcode.FormatPNG)
jan8.ShowText = true
jan8.ExtendedGuard = true
jan8.TextEvenSpacing = false
err = jan8.Draw("4901234", 200, 100)
```

チェックディジットは自動計算されるため、JAN-13なら12桁、JAN-8なら7桁を入力すればOKです。

> ヒント: JAN/UPCバーコードでは ExtendedGuard と TextEvenSpacing の組み合わせで見た目が変わります。商品バーコードらしい標準的な見た目にするには、ExtendedGuard=true + TextEvenSpacing=false の組み合わせがおすすめです。

UPC バーコード — 北米の商品コード

```
// UPC-A12
upcA := barcode.NewUPC_A(barcode.FormatPNG)
upcA.ShowText = true
upcA.ExtendedGuard = true
upcA.TextEvenSpacing = false
err := upcA.Draw("01234567890", 300, 100)

// UPC-E8
upcE := barcode.NewUPC_E(barcode.FormatPNG)
upcE.ShowText = true
upcE.ExtendedGuard = true
upcE.TextEvenSpacing = false
err = upcE.Draw("0123456", 200, 100)
```

ここからは、全メソッドの詳細なリファレンスです。

各メソッドのパラメータ、戻り値、デフォルト値を網羅しています。

6.1 共通メソッド（全バーコード）

すべてのバーコード型で使用できるメソッドです。

SetOutputFormat(format string)

出力形式を設定します。

パラメータ	型	説明
format	string	"png", "jpeg", "svg"

```
code.SetOutputFormat(barcode.FormatPNG)   // PNGBase64—
code.SetOutputFormat(barcode.FormatJPEG)  // JPEG
code.SetOutputFormat(barcode.FormatSVG)   // SVG
```

デフォルト: "png"（コンストラクタで指定）

SetForegroundColor(r, g, b, a uint8)

前景色（バーの色）を設定します。

パラメータ	型	説明
r	uint8	赤（0〜255）
g	uint8	緑（0〜255）
b	uint8	青（0〜255）
a	uint8	透明度（0=透明 〜 255=不透明）

```
code.SetForegroundColor(0, 0, 0, 255)      //
code.SetForegroundColor(0, 0, 128, 255)    //
code.SetForegroundColor(255, 0, 0, 128)    //
```

SetBackgroundColor(r, g, b, a uint8)

背景色を設定します。

```
code.SetBackgroundColor(255, 255, 255, 255)  //
code.SetBackgroundColor(255, 255, 240, 255)  //
code.SetBackgroundColor(0, 0, 0, 0)          //
```

SetPxAdjustBlack(adj int) / SetPxAdjustWhite(adj int)

黒バー / 白スペースの幅を微調整します。印刷時のにじみ補正に使います。

```
code.SetPxAdjustBlack(-1)  // 1px
code.SetPxAdjustWhite(1)   // 1px
```

デフォルト: 0

SetFitWidth(fit bool)

指定した幅にぴったり収めるかどうかを設定します。

デフォルト: false

> 仕組み: true の場合、バーの幅に小数ピクセルを使用して指定幅にぴったり収めます。false の場合は整数ピクセルのみ使用するため、指定幅より若干小さくなることがあります。

GetImageBase64() (string, error)

Base64エンコードされたデータURIを返します。

戻り値:

- PNG: "..." 形式
- JPEG: "..." 形式
- SVG: エラー（SVGモードでは GetSVG() を使用）

GetSVG() (string, error)

SVG文字列を返します（SVGモード時のみ）。

戻り値: "<svg xmlns="...">...</svg>" 形式

GetImageMemory() []byte

PNG/JPEGのバイト列を返します。ファイル保存や HTTP レスポンスに直接使用できます。

6.2 1次元バーコード共通メソッド

1次元バーコード（郵便カスタマバーコードを除く）で共通して使用できるフィールドとメソッドです。

Draw(code string, width, height int) error

バーコードを生成します。

パラメータ	型	説明
code	string	エンコードするデータ
width	int	画像の幅（px）
height	int	画像の高さ（px）

```
err := code.Draw("HELLO123", 400, 100)
```

ShowText bool

バーコード下部のテキスト表示を切り替えます。

デフォルト: true

TextEvenSpacing bool

テキストの均等割付を設定します。

```
code.TextEvenSpacing = true   // 1
code.TextEvenSpacing = false  //
```

デフォルト: true

> 使い分けのコツ: 一般的な1Dバーコード（Code39, Code128など）では true（均等割付）にすると、各文字がバーの真下に揃って読みやすくなります。一方、JAN/UPCバーコードでは false にして ExtendedGuard = true と組み合わせるのが、商品バーコードとしての標準的な見た目です。

SetTextFontScale(scale float64)

テキストのフォントサイズ倍率を設定します。

デフォルト: 1.0

SetTextVerticalOffsetScale(scale float64)

テキストの垂直オフセット倍率を設定します。値を小さくするとバーとテキストの間隔が狭くなります。

デフォルト: 1.0

SetMinLineWidth(width int)

最小線幅を設定します（ITF, Matrix2of5, NEC2of5 向け）。

デフォルト: 1

6.3 Code39

型: Code39 — 工業用途の定番バーコード

コンストラクタ: NewCode39(outputFormat string) *Code39

入力可能文字: 0-9, A-Z, - . $ / + %, スペース

固有フィールド

ShowStartStop bool

テキスト表示時にスタート/ストップコード（*）を表示するかどうか。

デフォルト: true

使用例

```
code := barcode.NewCode39(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
code.ShowStartStop = true
err := code.Draw("HELLO123", 400, 100)
```

6.4 Code93

型: Code93 — Code39の高密度版

コンストラクタ: NewCode93(outputFormat string) *Code93

入力可能文字: ASCII全文字（0x00〜0x7F）

固有フィールド: なし（共通フィールドのみ）

使用例

```
code := barcode.NewCode93(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
err := code.Draw("Hello123!@#", 400, 100)
```

6.5 Code128

型: Code128 — 物流の標準バーコード

コンストラクタ: NewCode128(outputFormat string) *Code128

入力可能文字: ASCII全文字（0x00〜0x7F）

固有フィールド

CodeMode int

定数	対応文字
Code128Auto	自動で最短幅に最適化（おすすめ）
Code128CodeA	制御文字 + 数字 + 英大文字 + 一部記号
Code128CodeB	数字 + 英大文字 + 英小文字 + 記号
Code128CodeC	数字のみ（2桁ずつ高密度エンコード）

デフォルト: Code128Auto

使用例

```
code := barcode.NewCode128(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
code.CodeMode = barcode.Code128Auto
err := code.Draw("Hello123", 400, 100)
```

6.6 GS1-128

型: GS1_128 — GS1標準準拠。物流・医療分野で使用

コンストラクタ: NewGS1_128(outputFormat string) *GS1_128

入力形式: AI（アプリケーション識別子）とデータの組み合わせ

固有メソッド

Draw(code string, width, height int) error

通常のGS1-128バーコードを生成します。

DrawConvenience(code string, width, height int) error

標準料金代理収納用（コンビニバーコード）を生成します。

使用例

```
// GS1-128
gs1 := barcode.NewGS1_128(barcode.FormatPNG)
gs1.ShowText = true
gs1.TextEvenSpacing = true
err := gs1.Draw("[01]04912345123459[10]ABC123", 500, 120)

//
gs1c := barcode.NewGS1_128(barcode.FormatPNG)
gs1c.ShowText = true
err = gs1c.DrawConvenience(
    "{FNC1}9191234500000000000000452087500401310029500", 500, 150,
)
```

6.7 NW-7（Codabar）

型: NW7 — 宅配便・図書館で使用

コンストラクタ: NewNW7(outputFormat string) *NW7

入力可能文字: 0-9, - $: / . +, スタート/ストップ: A B C D

固有フィールド

ShowStartStop bool

デフォルト: true

使用例

```
code := barcode.NewNW7(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
code.ShowStartStop = true
err := code.Draw("A1234567A", 400, 100)
```

6.8 ITF（Interleaved 2 of 5）

型: ITF — 集合包装用バーコード

コンストラクタ: NewITF(outputFormat string) *ITF

入力可能文字: 0-9 のみ（偶数桁必須）

固有フィールド: なし

使用例

```
code := barcode.NewITF(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
err := code.Draw("123456", 400, 100)
```

6.9 Matrix 2of5

型: Matrix2of5 — 工業用の数字専用バーコード

コンストラクタ: NewMatrix2of5(outputFormat string) *Matrix2of5

入力可能文字: 0-9 のみ

固有フィールド: なし

使用例

```
code := barcode.NewMatrix2of5(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
err := code.Draw("1234567890", 400, 100)
```

6.10 NEC 2of5

型: NEC2of5 — 日本の工業用途向け

コンストラクタ: NewNEC2of5(outputFormat string) *NEC2of5

入力可能文字: 0-9 のみ

固有フィールド: なし

使用例

```
code := barcode.NewNEC2of5(barcode.FormatPNG)
code.ShowText = true
code.TextEvenSpacing = true
err := code.Draw("1234567890", 400, 100)
```

6.11 JAN-8（EAN-8）

型: JAN8 — 小型商品用の8桁バーコード

コンストラクタ: NewJAN8(outputFormat string) *JAN8

入力: 数字7桁（チェックディジットは自動計算）

固有フィールド

ExtendedGuard bool

ガードバーの拡張。true にすると商品バーコードとしての標準的な外観になります。

デフォルト: true

テキスト表示パターン

ExtendedGuard	TextEvenSpacing	見た目
true	false	商品バーコードの標準スタイル。 ガードバーが長く伸び、
true	true	ガードバーが長く伸び、テキストは均等割付
false	false	フラットバー + テキスト中央寄せ
false	true	フラットバー + テキスト均等割付

使用例

```
code := barcode.NewJAN8(barcode.FormatPNG)
code.ShowText = true
code.ExtendedGuard = true
code.TextEvenSpacing = false
err := code.Draw("4901234", 200, 100)
```

6.12 JAN-13（EAN-13）

型: JAN13 — 日本の標準的な商品バーコード（13桁）

コンストラクタ: NewJAN13(outputFormat string) *JAN13

入力: 数字12桁（チェックディジットは自動計算）

固有フィールド

ExtendedGuard bool

6.11 JAN-8 と同じです。デフォルト: true

> JAN-13の特徴:
拡張ガードバー有効時、先頭1桁がバーコード左側にプレフィックスとして表示されます。日本の商品バーコードの「49」や「45」で始まるおなじみの見た目です。

使用例

```
code := barcode.NewJAN13(barcode.FormatPNG)
code.ShowText = true
code.ExtendedGuard = true
code.TextEvenSpacing = false
err := code.Draw("491234567890", 300, 100)
```

6.13 UPC-A

型: UPC_A — 北米の商品コード（12桁）

コンストラクタ: NewUPC_A(outputFormat string) *UPC_A

入力: 数字11桁（チェックディジットは自動計算）

固有フィールド

ExtendedGuard bool

6.11 JAN-8 と同じです。デフォルト: true

使用例

```
code := barcode.NewUPC_A(barcode.FormatPNG)
code.ShowText = true
code.ExtendedGuard = true
code.TextEvenSpacing = false
err := code.Draw("01234567890", 300, 100)
```

6.14 UPC-E

型: UPC_E — UPC-Aの短縮版（8桁）。小型商品用

コンストラクタ: NewUPC_E(outputFormat string) *UPC_E

入力: 数字6桁（チェックディジットは自動計算）

固有フィールド

ExtendedGuard bool

6.11 JAN-8 と同じです。デフォルト: true

使用例

```
code := barcode.NewUPC_E(barcode.FormatPNG)
code.ShowText = true
code.ExtendedGuard = true
code.TextEvenSpacing = false
err := code.Draw("0123456", 200, 100)
```

6.15 GS1 DataBar 標準型

型: GS1DataBar14 — 生鮮食品向けのコンパクトバーコード

コンストラクタ: NewGS1DataBar14(outputFormat string, symbolType SymbolType14) *GS1DataBar14

入力: 数字 8〜13桁（チェックディジットは自動計算）

シンボルタイプ

定数	説明
Omnidirectional	標準型（どの方向からでも読み取り可能）
Stacked	二層型（省スペース）
StackedOmnidirectional	標準二層型

使用例

```
code := barcode.NewGS1DataBar14(barcode.FormatPNG, barcode.Omnidirectional)
code.ShowText = true
err := code.Draw("1234567890128", 200, 80)
```

6.16 GS1 DataBar 限定型

型: GS1DataBarLimited — 先頭桁が0または1に限定されたコンパクト版

コンストラクタ: NewGS1DataBarLimited(outputFormat string) *GS1DataBarLimited

入力: 数字 8〜13桁（先頭桁は0または1のみ）

使用例

```
code := barcode.NewGS1DataBarLimited(barcode.FormatPNG)
code.ShowText = true
err := code.Draw("0123456789012", 200, 60)
```

6.17 GS1 DataBar 拡張型

型: GS1DataBarExpanded — 可変長データ対応

コンストラクタ: NewGS1DataBarExpanded(outputFormat string, symbolType ExpandedSymbolType) *GS1DataBarExpanded

入力: AI + データの組み合わせ

シンボルタイプ

定数	説明
Unstacked	一層型
StackedExp	多層型（スペースが限られる場合に）

固有メソッド

SetNoOfColumns(columns int)

多層型のセグメント数（列数）を設定します。偶数推奨。デフォルト: 2

使用例

```
//
db := barcode.NewGS1DataBarExpanded(barcode.FormatPNG, barcode.Unstacked)
db.ShowText = true
err := db.Draw("[01]90012345678908[10]ABC123", 400, 80)

//
dbStacked := barcode.NewGS1DataBarExpanded(barcode.FormatPNG, barcode.StackedExp)
dbStacked.SetNoOfColumns(4)
err = dbStacked.Draw("[01]90012345678908[10]ABC123", 300, 100)
```

6.18 郵便カスタマバーコード

型: YubinCustomer — 日本郵便の住所バーコード

コンストラクタ: NewYubinCustomer(outputFormat string) *YubinCustomer

入力: 郵便番号7桁 + 住所表示番号（ハイフン区切り可）

固有メソッド

Draw(code string, height int) error

他のバーコードと異なり、幅は自動計算 されるため高さのみ指定します。

> 注意: テキスト関連フィールド（ShowText, TextEvenSpacing 等）は使用できません。SetForegroundColor(), SetBackgroundColor() は使用可能です。

使用例

```
code := barcode.NewYubinCustomer(barcode.FormatPNG)
err := code.Draw("27500263-29-2-401", 25)
```

6.19 QRコード

型: QRCode — 日本発、世界で最も普及している2次元バーコード

コンストラクタ: NewQRCode(outputFormat string) *QRCode

入力: 数字、英数字、バイナリ、漢字（Shift-JIS）

固有メソッド

Draw(code string, size int) error

パラメータ	型	説明
code	string	エンコードするデータ
size	int	画像サイズ（px、正方形）

SetStringEncoding(encoding string)

値	説明
"utf-8"	UTF-8（おすすめ）
"shift-jis"	Shift-JIS（レガシー環境との互換性が必要な場合）

デフォルト: "utf-8"

SetErrorCorrectionLevel(level int)

定数	復元能力	こんなときに
QREccL	約7%	データ量優先
QREccM	約15%	一般的な用途（おすすめ）
QREccQ	約25%	汚れ・傷への耐性が必要
QREccH	約30%	最高品質。ロゴ重ね時にも

デフォルト: QREccM

SetVersion(version int)

QRコードのバージョン（セルの数）を指定します。0（自動）～ 40。

デフォルト: 0（データに応じた最小バージョンを自動選択）

SetEncodeMode(mode string)

定数	説明
QRModeBinary	バイトデータ（デフォルト、おすすめ）
QRModeNumeric	数字のみ（最高効率）
QRModeAlphaNumeric	英数字
QRModeKanji	漢字（Shift-JIS）

デフォルト: QRModeBinary

使用例

```
qr := barcode.NewQRCode(barcode.FormatSVG)
qr.SetStringEncoding("utf-8")
qr.SetErrorCorrectionLevel(barcode.QREccM)
qr.SetVersion(0)
qr.SetFitWidth(true)
err := qr.Draw("https://www.pao.ac/", 300)
svg, _ := qr.GetSVG()
```

6.20 DataMatrix

型: DataMatrix — 超小型マーキングの世界標準

コンストラクタ: NewDataMatrix(outputFormat string) *DataMatrix

入力: ASCII文字、バイナリデータ、GS1データ（{FNC1} で開始）

固有メソッド

Draw(code string, size int) error

パラメータ	型	説明
code	string	エンコードするデータ
size	int	画像サイズ（px、正方形）

SetStringEncoding(encoding string)

"utf-8"（デフォルト）または "shift-jis"

SetCodeSize(size int)

主な定数	説明
DxSzAuto	自動（おすすめ）
DxSz10x10 ～ DxSz144x144	正方形
DxSz8x18, DxSz8x32 等	矩形

デフォルト: DxSzAuto

SetEncodeScheme(scheme int)

定数	説明
DxSchemeAutoBest	自動選択（おすすめ）
DxSchemeAscii	ASCII
DxSchemeC40	英数字
DxSchemeText	テキスト（小文字優先）
DxSchemeX12	ANSI X12 EDI
DxSchemeEdifact	EDIFACT
DxSchemeBase256	バイナリ

デフォルト: DxSchemeAutoBest

GS1-DataMatrix

GS1データを格納する場合は、先頭に {FNC1} を付けます。

```
dm.Draw("{FNC1}0100012345678905{FNC1}10ABC123", 200)
```

使用例

```
dm := barcode.NewDataMatrix(barcode.FormatSVG)
dm.SetStringEncoding("utf-8")
dm.SetCodeSize(barcode.DxSzAuto)
dm.SetEncodeScheme(barcode.DxSchemeAutoBest)
dm.SetFitWidth(true)
err := dm.Draw("Hello World", 200)
svg, _ := dm.GetSVG()
```

6.21 PDF417

型: PDF417 — 大容量2次元バーコード。運転免許証・搭乗券に使用

コンストラクタ: NewPDF417(outputFormat string) *PDF417

入力: テキスト、数字、バイナリ

固有メソッド

Draw(code string, width, height int) error

パラメータ	型	説明
code	string	エンコードするデータ
width	int	画像の幅（px）
height	int	画像の高さ（px）

SetStringEncoding(encoding string)

"utf-8"（デフォルト）または "shift-jis"

SetErrorLevel(level int)

定数	訂正能力
PDF417ErrorLevel0	最小
PDF417ErrorLevel1	低
PDF417ErrorLevel2	標準（おすすめ）
PDF417ErrorLevel3 〜 PDF417ErrorLevel8	高〜最大

デフォルト: PDF417ErrorLevel2

SetColumns(columns int)

列数。0=自動、1〜30で指定。デフォルト: 0

SetRows(rows int)

行数。0=自動、3〜90で指定。デフォルト: 0

SetAspectRatio(ratio float64)

縦横比。1.0〜10.0。デフォルト: 3.0

SetYHeight(height int)

Y方向の高さ係数。1〜10。デフォルト: 3

使用例

```
pdf := barcode.NewPDF417(barcode.FormatSVG)
pdf.SetStringEncoding("utf-8")
pdf.SetErrorLevel(barcode.PDF417ErrorLevel2)
pdf.SetColumns(4)
pdf.SetRows(0)
pdf.SetAspectRatio(3.0)
pdf.SetYHeight(3)
pdf.SetFitWidth(true)
err := pdf.Draw("Hello World", 400, 100)
svg, _ := pdf.GetSVG()
```

Barcode.Go は Go の WASM ターゲット（GOOS=js GOARCH=wasm）でコンパイルすることで、ブラウザ上でも動作します。

ビルド方法

```
cd wasm
GOOS=js GOARCH=wasm go build -o barcode.wasm .
```

Windows の場合:

```
set GOOS=js
set GOARCH=wasm
go build -o barcode.wasm .
set GOOS=
set GOARCH=
```

必要ファイル

```
wasm/
├── barcode.wasm     ← Go WASM
├── wasm_exec.js     ← Go  JS $GOROOT/misc/wasm/
└── index.html       ←
```

wasm_exec.js は Go のインストールディレクトリに含まれています。

使い方

```
<script src="wasm_exec.js"></script>
<script>
const go = new Go();
WebAssembly.instantiateStreaming(fetch("barcode.wasm"), go.importObject)
    .then(result => {
        go.run(result.instance);
        // JavaScript  Go
        const base64 = drawQR("https://www.pao.ac/", 300);
        document.getElementById("barcode").src = base64;
    });
</script>
```

> ヒント: WASM版のダウンロードパッケージには、すぐに試せるデモ HTML が含まれています。

Go バージョン

項目	要件
Go	1.21 以降

依存ライブラリ

パッケージ	用途
golang.org/x/image/font/opentype	TrueType フォント描画（テキスト表示）

上記以外はすべて Go 標準ライブラリのみを使用しています。CGO は不要です。

対応OS（WASM版除く）

Go がサポートするすべてのOS / アーキテクチャで動作します。

OS	アーキテクチャ
Windows	amd64, arm64
macOS	amd64 (Intel), arm64 (Apple Silicon)
Linux	amd64, arm64, arm

WASM版の対応ブラウザ

ブラウザ	対応バージョン
Google Chrome	57 以降
Mozilla Firefox	53 以降
Safari	11 以降
Microsoft Edge	16 以降

使用許諾

Barcode.Go
の使用について、利用者様と有限会社パオ・アット・オフィス（以下「弊社」）は、以下の各項目に同意するものとします。

1. 使用許諾書

この使用許諾書は、利用者様がお使いのパソコンにおいて Barcode.Go を使用する場合に同意しなければならない契約書です。

2. 同意

利用者様が Barcode.Go を使用する時点で、本使用許諾書に同意されたものとします。

3. ライセンスの購入

製品版を使用して開発を行う場合、1 台の開発用コンピュータにつき 1 ライセンスの購入が必要です。お客様環境等、開発コンピュータでないマシンでの使用にはライセンスは不要です（ランタイムライセンスフリー）。

4. 著作権

Barcode.Go の著作権は、いかなる場合においても弊社に帰属いたします。

5. 免責

Barcode.Go
の使用によって、直接的または間接的に生じたいかなる損害に対しても、弊社は補償賠償の責任を負わないものとします。

6. 禁止事項

Barcode.Go
およびその複製物を第三者に譲渡・貸与することはできません。開発ツールとしての再販・再配布を禁止します。ただし、モジュールとして組み込みを行い再販・再配布する場合は問題ございません。

7. 保証の範囲

弊社は Barcode.Go の仕様を予告なしに変更することがあります。利用者様への情報提供は弊社 Web サイトにて行います。

8. 適用期間

本使用許諾条件は利用者様が Barcode.Go を使用した日より有効です。

ライセンス

Barcode.Go は有限会社パオ・アット・オフィスの製品です。

試用版:
生成されるバーコードに「SAMPLE」の透かしが表示されます。機能制限はありません。すべてのバーコード種類・設定を自由にお試しいただけます。

製品版: 透かしなしでバーコードを生成できます。ライセンスの詳細は弊社Webサイトをご確認ください。

お問い合わせ

有限会社 パオ・アット・オフィス

Webサイト	https://www.pao.ac/
製品ページ	https://www.pao.ac/barcode.go/
メール	info@pao.ac

関連製品

製品	対応環境
Barcode.net	.NET（C#, VB.NET）
Barcode.jar	Java
Barcode.php	PHP
Barcode.wasm	JavaScript / TypeScript（ブラウザ）
Barcode.py	Python
Barcode.Flutter	Flutter / Dart

Barcode.Go ユーザーズマニュアル

バージョン 1.0 — 2026年2月