情報技術基礎 II: Web と情報アーキテクチャ (現代の情報技術 9)

久野 靖 (電気通信大学)

2020.5.12

今回の目標は次の通りです。

- 相対 URL や外部メディアの参照について学ぶ — ページに画像を使えるようになることで、見栄えのする (見てもらいやすい) サイトが作れます。
- Web サイトの構成および情報アーキテクチャの考え方について学ぶ — 定番なサイト構造を知っておくことで、労力をかけずにわかりやすいサイトが作れます。
- CSS によるページデザインについて学ぶ — CSS を用いてどのページでも統一された見え方のページ群が作れるようになり、また 1 箇所の変更で全部の表示を調整できます。

0 再掲: ファイルに HTML を作る

ここまでは「HTML 練習ページ」を使ってページを作って来ましたが、これではブラウザを止めたら内容が無くなってしまいますし、複数のページを作ることもできません。そこで、これまで作った HTML をファイルに保存して「普通の形で」ブラウザで表示してみましょう。保存場所としては「各自の USB メモリの中のフォルダ」を使用します。

(1) エクスプローラで作業に使うフォルダを表示 (まだ無ければ「新規フォルダ」機能で作る)。

(2) メニューの「アクセサリ」→「メモ帳」でメモ帳を開く。

(3) 「HTML 練習ページ」で作ってあったソースを貼り付けるか、自分で HTML を打ち込む。

(4) 「名前をつけて保存」で保存するが、保存時にファイルの種類を「すべての種類」にしてから、「mypage1.html」などの名前で (最後が「.html」) 保存。

(5) エクスプローラ上の HTML ファイルをドラグしてブラウザの窓にドロップすると開く。

このフォルダを保存するには、フォルダを USB メモリにドラグしてコピーするか、または最初から USB メモリ上にフォルダを作って作業してください。

慣れてきたら「練習ページ」は使わずに、直接メモ帳で打ち込んだり直したりしてもよいです。ただ、練習ページを使うとすぐチェックできおかしいところが分かるという利点はあります。

以下では基本的に今作業しているフォルダに色々なファイルを作ることにします。

1 外部ページ/外部メディアの参照

1.1 絶対 URL と相対 URL

これまで、Web ページを表示させたり HTML 中で a タグに記載する URL は次のようなものでした。

```
http://example.com/aa/bb/mypage.html
スキーム　ホスト　パス
```

このような (http:などの) スキームから始まる URL を**絶対 URL** と呼びます。すべてのサイトは絶対 URL で指定できますが、次のような場合には絶対 URL は不便です。

- 1 つのサイトが複数のページからできていて、相互にリンクで行き来するような場合。
- ページの HTML に加えて、そこで使用する画像などのファイルを参照している場合。

どのように不便かというと、(1) リンクなどで指定する URL が長くなり記述が繁雑であることと、(2) サイト内のファイルをまとめてよそのサーバに引越したときに、全部リンクを直さなければいけないことです (絶対 URL が書いてあるということは、リンクをたどると元のサーバから読みに行きますから、引越しの意味がありません)。

そこで Web では**相対 URL** も使うことができます。たとえば上の URL のファイル mypage.html にリンクを書くとき、相対 URL ではその置いてあるディレクトリ http://example.com/aa/bb/ が「起点」となります。そして、あとは Unix のパス名と同じようにそこを起点に対象とするファイルを指定します (.. も使えますが、ただしホストやスキームは変更できません)。表 1 に上の例のページを起点としたときの相対 URL とそれを解釈して絶対 URL に直したものの対照を示します。

表 1: 「http://example.com/aa/bb/」を起点とする相対 URL の解釈

相対 URL	絶対 URL
page2.html	http://example.com/aa/bb/page2.html
fig1.png	http://example.com/aa/bb/fig1.png
FIGS/fig1.png	http://example.com/aa/bb/FIGS/fig1.png
../FIGS/fig1.png	http://example.com/aa/FIGS/fig1.png
../../FIGS/fig1.png	http://example.com/FIGS/fig1.png
/aa/FIGS/fig1.png	http://example.com/aa/FIGS/fig1.png
./	http://example.com/aa/bb/
../	http://example.com/aa/

基本的に Unix のパス名のようなものだと考えていいのですが、(1) まったく「/」がなくても相対 URL であること、(2)「/」で始まるものも相対 URL であること (その Web サーバの公開用トップディレクトリからたどる意味になる)、などが違います。

なお、一番下の 2 つのようにパスが「/」で終わる場合はディレクトリを指しますが、そのときに何が表示されるかはサーバの設定次第で、多くのサーバではそこにある index.html というファイルの内容を返すように設定されています。

1.2 画像の使用

だいぶお待たせしましたが、ここで Web ページにおける画像の使用方法について説明します。なぜここまで待ったかというと、画像は HTML の中に直接は入れられないのでサーバ上に別のファイルとして置く必要があり、その参照はふつう相対 URL によるからです。

さて、まず画像ファイルの形式についてですが、ピクセル画像では **PNG**、**GIF**、**JPEG** のいずれかの形式、ベクトル画像では **SVG** 形式のものを使用してください。これらは多くのブラウザで共通にサポートされています (SVG は未サポートなブラウザがまだあります)。

とりあえず、画像を自分で描く方法をやってみます。「アクセサリ」→「ペイント」でペイントを起動し、画像サイズが大きすぎると使いにくいので「100x100」くらいに設定し、描きやすいように倍率を拡大してから絵を描いてみてください。最後に保存するとき「PNG」形式を選び、ファイル名を「なんとか.png」にしてください。

次に、Web ページで画像を使うやり方ですが、次の 3 種類があります。

(1) リンク先として — ...のようにリンク先として画像を指定した場合、リンクを選択するとブラウザ画面にその画像が単独で表示されます (図 1)。

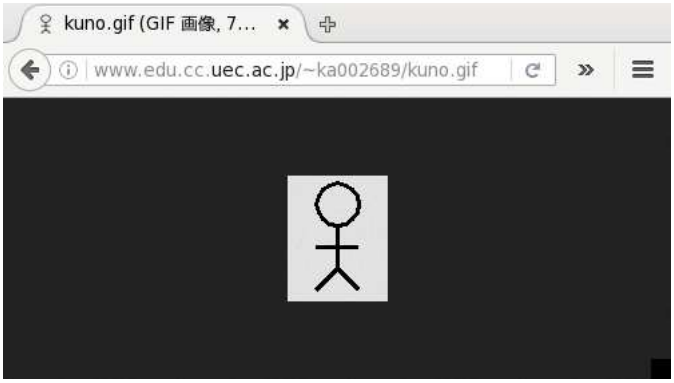

図 1: リンク先として画像を指定した場合

図 2: 埋め込み画像 (img 要素) として画像を指定した場合

(2) 埋め込み画像 — `<img src="fig1.png" alt="説明">` のようにして、**img 要素**を使うことで、ページのその場所に画像が埋め込めます (図 2。この要素はインライン要素です)。なお、`alt` 属性は画像が表示できない時に表示したり、目に障害がある人が読み上げブラウザを使っているときに読み上げたりするのに使います。

(3) 背景画像 — CSS で「`background-image: url(fig1.gif)`」などのように指定すると、その指定した要素の「背景模様」として指定した画像が繰り返し敷き詰められて表示されます (図 3)。ページ全体の背景にしたければ、セレクタで `body` や `html` を指定してください。なお、繰り返し敷き詰めたくなければ、同じセレクタに対して `background-repeat:` プロパティとして `no-repeat`(繰り返さない)、`repeat-x`(横方向のみ繰り返す)、`repeat-y`(縦方向のみ繰り返す) を指定できます。通常は `repeat`(縦横とも繰り返す) です。

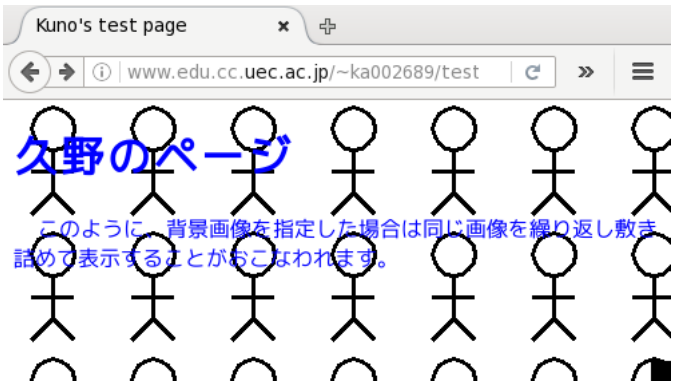

図 3: 画像を背景画像として指定した場合

演習 1 Web ページに画像を入れる演習として、次のものから 1 つ以上やってみなさい (題材の画像ファイルは各自工夫して入手のこと)。

a. 複数の画像をリンク先としたリストを含むページを作ってみる。なお、`<a href="..." target="名前">...` のように a タグに **target 属性**を指定するとどのような効果があるか試して検討してみること。

b. 画像を img 要素を使ってページ内に埋め込んでみなさい。このとき、この要素に対して CSS で枠や空白あけを指定して見やすくしてみなさい。さらに「`float: left`」「`float: right`」を指定するとどのような効果があるか試して検討してみること。

c. 画像を適当な要素の `background-image` として指定して効果を観察しなさい。複数の要素 (たとえば `body` とページの中にある `h1` や `p` など) に別々の背景画像を指定して、見た目だけでなくページ内容の読みやすさのためにはどういう配慮が必要か検討しなさい。画像のつぎ目の柄合わせのために位置を微調整する機能が CSS にあるか探してみるとなおよい。

2 情報アーキテクチャ

2.1 情報アーキテクチャとサイト構造

情報アーキテクチャ(information architecture) とは、広義には情報を分かりやすく伝え、受け手が情報を探しやすくするための技術全般を指します。また狭い意味では、Web サイトの構築に先立ち、情報を分類し、意味を整理し、サイト構造やサイトの使われ方を検討する分野を指す場合もあります。

なぜこのような分野が必要になったのでしょうか。もともと WWW はハイパーテキストから生まれており、どのページのどこにでも他のページへのリンクを埋め込めることが特徴でした。これを活かして、多数のページが互いにつながり合った構造が**ネットワーク構造**です (図 4 左)。しかし WWW でこのような構造を作ると、今居る場所が分からず (迷子)、混乱が生じることが分かってきました。そこで、ページを直線状につなげた**線形構造** (図 4 中)、ページを大分類→中分類→小分類のようにテーマに従って堀り下げていく**階層構造** (図 4 下) が生み出され使われるようになりました。

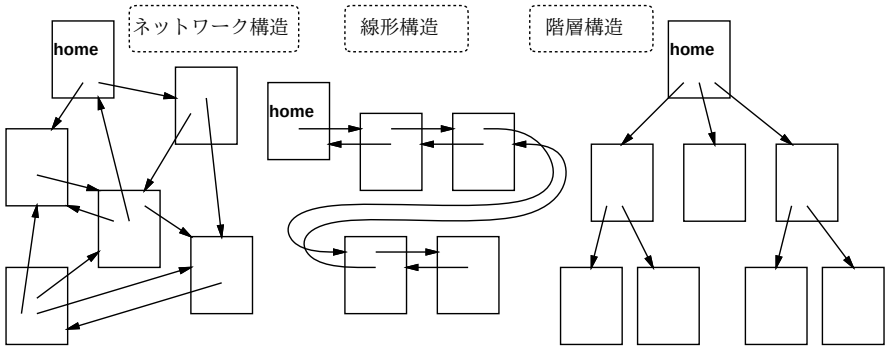

図 4: Web サイトの代表的な構造

線形構造はインタビュー記事のように流れが直線的で分量がさほど多くない場合は分かりやすいのですが、分量が多くなると先の方までたどって行くのが大変です。そのために目次を用意してそこまで飛べるようにするなどの工夫がなされます。

階層構造はファイルシステムのところでも述べたように、大量の情報を整理して納めるには適しています。そのかわり、すべての情報をひととおり見るようなことはやりにくいです。この対策として、情報はすべて一番下の「葉」のところにいれて、そこを線形構造で結ぶなどの工夫もあります。しかしそうすると、中間の分類を下までたどっていくのに手間がかかります。またそもそも、何を規準に階層構造に分類するべきか、ということも自明ではないかも知れません。

このような構造の問題や、個々のページの使いやすさの問題まで含めて、どのようにしたらよいかを考えて実現するのが、情報アーキテクチャの分野であるといえるでしょう。

なおついでですが、図で home と書いてあるのは一連のページの入口となるページです。本来はこのようなページを「ホームページ」と呼ぶはずですが (またはブラウザの home ボタンを押したときに移動するページの意味もある)、日本ではなぜか WWW 自体のことを「ホームページ」と呼ぶ習慣があるので、どのような意味でこの語を使っているのか、そのつど注意が必要です。

2.2 ページナビゲーション

ナビゲーション (navigation、航行) とは一般には進路を制御して目的地に向かう作業をいいますが、Web の場合は「行きたいページに行かせてくれるための機能全般」を指します。

とくに、前節で述べた「線形構造」「階層構造」の場合は、どのページにも同じような形のリンクが必要になります。具体的には、線形構造であれば各ページに「前へ」「次へ」が必要ですし、階層構想では中間のページにはそれぞれの「子供」に行くリンク、そしてトップ以外の全てのページには「上へ」行くリンクが必要です。また、**パンくずリスト** (crumb list) と呼ばれる、一番上から現在のページまでの経路のリストもあると位置が分かりやすくなります (図 5 の囲んだ部分)。

図 5: パンくずリストの例

このような、サイトの設計に応じてどのページでも同じように使えるリンクのことを**ナビゲーションリンク**と呼びます。ナビゲーションリンクは頻繁につかうので、ページ内でナビゲーションリンクのありかがすぐ分かることは使いやすさのために重要です。このため、ナビゲーションリンクを集めた領域である「ナビゲーションバー」を作成し、どのページでも同じ場所 (ページの先頭、末尾など) に配置するという工夫は多くのサイトでなされています。

演習 2 ページ構成に関する次のテーマから 1 つ以上やってみなさい。

a. 線形構造のサイトの例として「浦島太郎」のストーリーをもとにしたサイトを作る。入口ページのほかに「亀を助ける」「竜宮城に連れていってもらう」「乙姫と楽しくすごす」「乙姫に別れを告げる」「故郷に戻ってみると」の 5 ページを作り線形構造につなぐ。ページ内容は自由だが「前へ」「次へ」のリンクは必須とする (最初と最後は適宜修正)。浦島太郎を知らない場合はクラスメートに説明してもらうか、別のストーリーに変えてもよい。

b. 階層構造のサイトの例として「食材」のサイトを作る。入口ページのほかに「魚介類」「魚」「あじ」「しゃけ」「貝」「あさり」「しじみ」「肉」「牛肉」「鶏肉」「野菜」「キャベツ」「たまねぎ」の 13 以上を作る (個々の食材は別のものにしてもよいし、追加してもよい)。ページ内容は自由だが、すべてのページに「上へ」のリンク、そして分類のページにはその中の各種別へのリンクが必須とする。

c. この課題は a または b をやった後にそれに加えて選ぶこと。a の線形構造をやった場合は、入口ページに「目次」を追加しなさい。b の階層構造をやった場合は、個々の食材のページだけを順番に線形構造でながめて行けるようなリンクを追加するか、またはパンくずリストを追加しなさい。

3 CSS と HTML によるページレイアウト

3.1 ページの構成要素のグループ化

今回の Web ページ制作では CSS を使ってページ内容の配置を行います。また、CSS では (既に学んだように) 背景色、文字色の配色設定も行えます。ここではこれらを実際に (簡単なものですが) やってみましょう。

まず、CSS でレイアウトする場合、各ページの内容はいくつかの「部分」に分けて作成することになります。たとえば次のような部分が考えられます (デザインに応じて、これらのうちから 2〜3 個を用意するのが普通ですが、さらにこれら以外のものを加えることもあるかも知れません)。

- トップバナー — ページの先頭に置き、タイトルやロゴなどを目立つように配置する。
- メイン (本体) — ページの本体つまり主たる内容を入れる。
- フッター — ページの最後に置き、著作権表示やナビゲーションバーなどを入れることが多い。
- 左サイドバー — ページの左端に細長く置き、ナビゲーションバーやその他各ページで共通に使う機能、リンクを置くのに使う。
- 右サイドバー — ページの機能が多くて左サイドバーだけでは縦長になりすぎる (スクロールしないと見えなくなる) 場合に、右にもサイドバーを置くことがある。

分かりやすく無難な方法として、ページを縦/横に区切りながらこれらの要素をはめ込んで行く**ブロックレイアウト**と呼ばれる方法があります (図 6)。これに対し、縦横の配置にこだわらずもっと自由なデザインで配置する方法もありますが、デザインの難易度は高くなります。以下ではブロックレイアウトを前提とします。

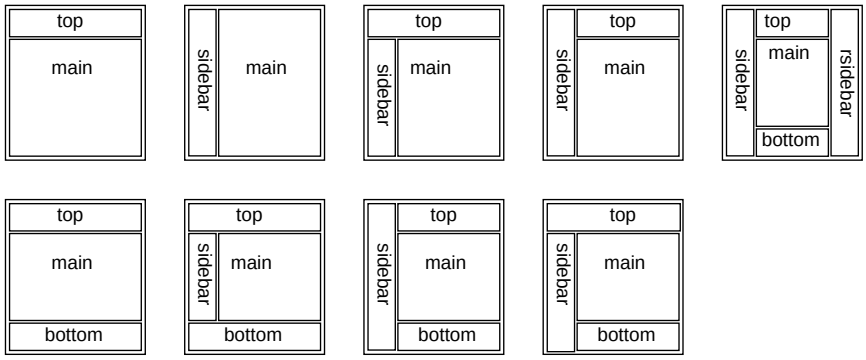

図 6: 標準的なブロックレイアウト

これまで HTML ページはいきなり見出し等から始まっていましたが、レイアウトを行う場合はその各ブロックをそれぞれ何らかの要素で囲む必要があります。

HTML 4(1 つ前のバージョンの HTML) までではそのような目的に使えるものが `div` しかなかったので、すべて `div` で囲み、`id` または `class` 属性を指定して、それぞれ「`#id` 名」または「`.` クラス名」を CSS のセレクタで指定しました。それでは分かりづらいので、HTML 5 (現在のバージョン) になったときに、目的別に次のような要素が追加されています (これらで不足する場合は従来の方法を併用します)。

- `<section>...</section>` — 1 つの本文セクションを表す。
- `<article>...</article>` — 1 つの本文記事を表す。
- `<aside>...</aside>` — 本文とは別の内容を表す。
- `<nav>...</nav>` — ナビゲーション部分を表す。
- `<header>...</header>`、`<footer>...</footer>` — ヘッダ、フッタを表す。

3.2 CSS グリッドによるレイアウト

実際に図 6 のような配置を実現するのには、以前は CSS のさまざまな「技」を駆使する必要があり面倒だったのですが、今は **CSS グリッド**と呼ばれる機能を使って簡単に構成できるようになっています。それには、たとえば図 6 の最後の配置を実現するのであれば、ページ全体 (body 要素の内側) を 1 つの div 要素にした上で、次の CSS を指定します (ページ全体を囲む div 要素に `class="main"` を指定することにします)。

```
.main { display: grid; width: 100vw; height: 100vh;
        grid-template-areas: "a a" "b c" "b d";
        grid-template-columns: 8em 1fr;
        grid-template-rows: 6em 1fr 3em }
```

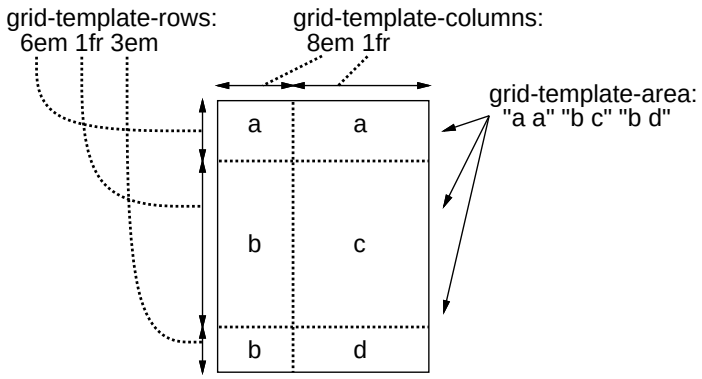

図 7: グリッド指定の意味

まず 1 行目で、この範囲をグリッドで配置することを指定し、またこの要素をブラウザの画面全体 (幅 100%、高さ 100%) と指定します。次の `grid-template-areas` で、図 7 のように、横 2 カラム、縦 3 行のグリッドを作り、そのそれぞれのます目に入る要素に a, b, c 等の名前をつけています (同じ名前をつけたものは複数のグリッドにまたがって配置される)。指定の`"a a" "b c" "b d"`と図 7 の各セルの記号が対応していることを確認してください。[1] その次の `grid-template-columns`、`grid-template-rows` は各グリッドの幅と高さの指定で、「`1fr`」と指定されている部分が直接大きさを指定した部分の「残り」になります。

なお、長さの指定においては単位として「`em`(文字 m の幅)」「`ex`(文字 x の高さ) など文字サイズを基準にした指定値を使う方がよいでしょう。というのは、「`px`」(ピクセル数) や「`cm`」などの指定だとその大きさに固定されてしまうのに対し、文字サイズ指定なら読み手がブラウザのメニューで文字を大きくしたとき対応して長さが大きくなって釣合いが保てるからです。

これらでグリッドを定義したあとは、それぞれの要素に grid-area として先の a, b, c 等の名前を指定すれば、その要素が指定したグリッドの位置にはめ込まれます。実際に見てみましょう。

```
<!DOCTYPE html>
<html><head>
<meta charset="utf-8">
<title>sample</title>
<style type="text/css">
body { margin: 0px }
.main { display: grid; width: 100vw; height: 100vh;
        grid-template-areas: "a a" "b c" "b d";
```

[1] 1 つの文字列が横 1 行を表し、文字列が 3 つあることで縦 3 行、文字列の中に名前が 2 つずつあることで横 2 カラムであることを表します。

```
        grid-template-columns: 8em 1fr;
        grid-template-rows: 6em 1fr 3em }
header { grid-area: a; background: rgb(200,240,180); padding: 1em }
aside  { grid-area: b; background: rgb(180,220,220); padding: 1em }
section { grid-area: c; background: rgb(210,255,240); padding: 1em }
footer { grid-area: d; background: rgb(200,240,180); padding: 1em }
address { text-align: right }
</style>
</head><body><div class="main">
<header><h1>A Sample Page</h1></header>
<section><p>A</p><p>B</p><p>C</p></section>
<aside><p>a</p><p>b</p><p>c</p></aside>
<footer><address>Today's Informatoin Technology</address></footer>
</div></body></html>
```

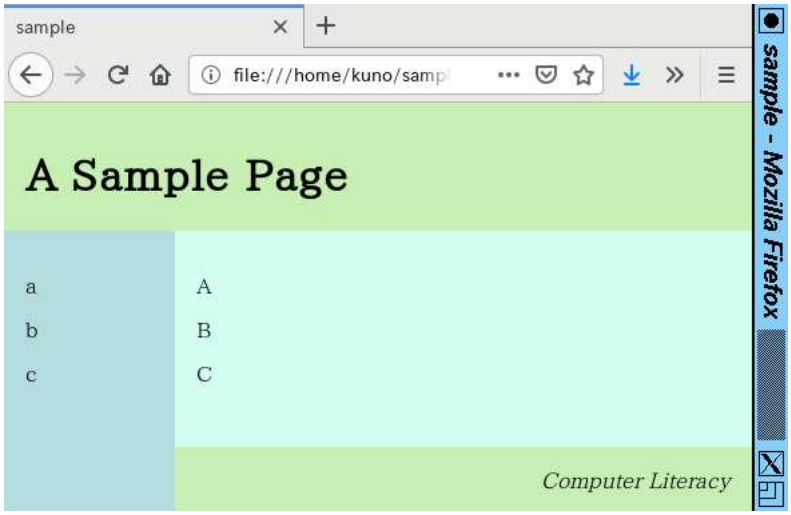

図 8: ブロックレイアウトの例

この例を表示したようすを図 8 に示します。この例ではバナー、本体、サイドバーの内容は仮に入れたものであり、本来ならこれらの部分には、見出し、段落、箇条書き、リンクなどのタグを使って、それぞれの内容を記述していきます。

3.3 CSS のマージンとパディング

それぞれの要素内を見やすく配置するのは、やはり CSS の機能を使って行います。このとき知っておく必要があるのは次のプロパティ程度です (マージンとパディングの関係は図 9 を見てください)。少なくともグリッドに使った要素にパディングを指定しないと、要素がグリッドの端にぴったりくっついてしまいます。

- `margin`: 上 右 下 左 — 要素の「外側の」アキを上/右/ 下/左の順で指定。数値が 2 個の時は「上下」「左右」をそれぞれ指定、1 個のときは 4 つとも同じ値を指定。
- `padding`: 上 右 下 左 — 要素の「内側の」アキを指定。指定方法は margin と同様。
- `background`: 色 — 背景色の指定は必要に応じて。

演習 3 ブロックレイアウトの例をそのまま動かしてみなさい。うまく動いたら、`grid-template-area` を変更して配置を修正したり、サイドバーを右側にしたり、ブロックの数を増減してみなさい。

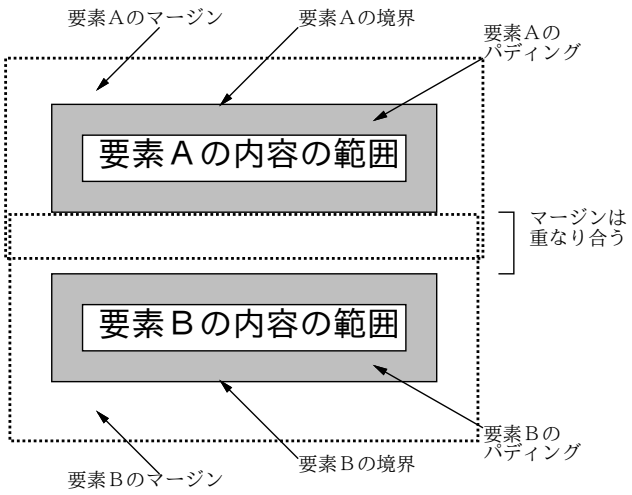

図 9: マージンとパディングの関係

最終的には、自分が自分のサイトを作るときに使いたいと思う色づかいと配置のレイアウトを構成してみなさい。

本日の課題 9A

本日の課題は「演習 1」「演習 2」に含まれる小問か「演習 3」(合計で 7 個) の中から 1 つ以上を選択し、その内容を説明するレポートを作成することです。レポートは紙で提出なので、手書きでもコンピュータで作成でも構いませんが、HTML のソースやページの表示をおこなった様子を含めてください。これらはブラウザ等ソフトの印刷機能や画面プリント機能で出力された紙をはさんでもよいですし、画面キャプチャをレポートの中に図として貼りつけてもかまいません。

レポートは次回授業前日一杯までに教務部の指示に従って提出してください。

レポートには次の内容がこの順で含まれること。

- 表紙。題名「現代の情報技術 9A」、学籍番号と氏名、提出日付を書く。さらに、グループで協力してやった場合はグループの他のメンバー全員の学籍番号と氏名を書く。
- 課題の再掲を書く (どんな課題であるかをレポートを読む人が分かる程度に要約する)。
- レポート本体の内容 (やったこととその結果) を書く。今回の場合、作成した HTML とその説明、画面表示とその説明が含まれること。
- 考察 (課題をやった結果自分が新たに分かったことや考えたこと) を書く。
- 以下のアンケートに対する回答。

  Q1. HTML によるページの記述はどれくらい知っていましたか。今回やってみてどうでしたか。

  Q2. CSS による表現の指定はどれくらい知っていましたか。今回やってみてどうでしたか。

  Q3. リフレクション (今回の課題で分かったこと)・感想・要望をどうぞ。

なお、課題はグループで協力してやって構いません。その場合も、(メンバー氏名を明記した上で) レポートは必ず各自で執筆してください。レポート文面が同一 (コピー) と認められた場合は同一であると認めた全員について点数にペナルティを科すことがあります。