打ち合わせ資料 2013/10/17

聞き取り調査の計画

実装する機能

コンテンツの提示
・データベースに入れず、普通にHTMLページを記述する
・参照時間、その中での参照割合、理解度を、ページごとにアンケートをとる。従って、認証をしてページを表示する

問題と解答
・ひたすらフォームを作る、結果はデータベースに残す。認証ログイン後が前提

アンケート
・ひたすらフォームを作る、結果はデータベースに残す。認証ログイン後が前提

以下、原稿です。

↓このスタイルの段落で、次のページになります。

Page1: Webページ制作についての基本

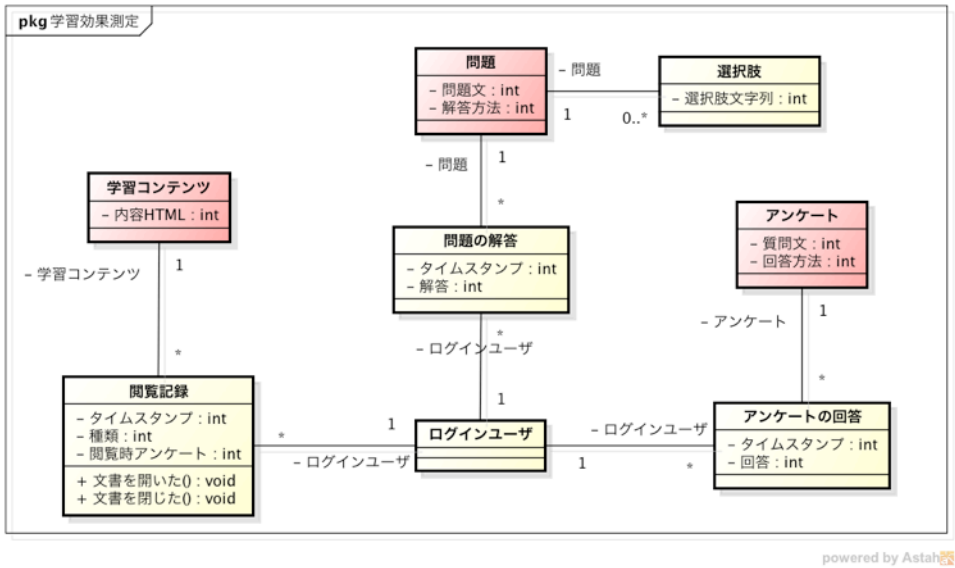
まず最初に、Webページの成り立ちについて、知っておいていただきたい知識をまとめておきます。次のページ以降の内容を読み進めるための前提知識をこのページで説明します。

WebサーバとWebクライアント

・Webページのコンテンツは、通常は「サーバ」というインターネット上で共有可能なコンピュータ上に保持されています。それらをネットワークを通じてデータをクライアントに転送した上で参照します。クライアントとして使われるアプリケーションは、一般には「Webブラウザ」と呼ばれます。

HTMLについて

・Webページのコンテンツは、HTMLという規格に従ってテキストで記述されたものが中心になっており、サーバ上ではたとえば拡張子が.htmlの1つのファイルで保存されています。そのファイルをWebブラウザが解釈することで、段落や表、



色の付いた文字などのレイアウトを施したWebとして表示できます。HTMLで記述されたファイルを「HTMLファイル」、HTMLの記述内容に従って表示された結果を「HTMLページ」と呼びます。

・HTMLによる記述の特徴は、表示する文字とは別に、「タグ」として書式やレイアウトなど、テキストをどのように見せるのかという記述も付加されていることです。また、タグはデータの区切り目を示すなど見せ方に限らない使い方もあります。

・タグは、たとえば abcdef のような書き方をします。開きタグ、閉じタグで囲まれるのが一般的で、divのような「タグ名」はHTMLで決められたものを利用します。また、alignのような「属性」をタグに付与できますが、属性名はタグごとに使える物が決まっています。属性名、イコールに続いて、属性の値がダブルクォート等で囲まれて記述され、値は状況に寄って変わります。

・Webページの表示では、テキスト以外に、画像やムービーも表示できます。これらは、通常はサーバでは別のファイルとして保存されたものになっており、HTMLページ内に特定の画像を表示するには、画像であれば画像を表示するIMGタグのSRC属性に、画像のファイル名やURIを記述します。

CSSとJavaScriptについて

・HTMLで表示されるものの書式設定は、CSSとして別途規格が定められており、多様な表現が可能になっています。CSSを利用する事で表現力が上がるだけでなく、まとめて設定したり、複数のHTMLファイルで設定情報を共有することがで

きます。前者は、idやclass属性、あるいはタグに対する書式設定ができることで実現されています。後者は、CSSの定義自体を独立したファイルに記述して、HTMLからCSSが記述されたファイルの内容を参照する仕組みで実現されています。

- HTMLでの記述やCSSによる記述は、その場でどうするかということ（例えば、画像を見せるとか、段落を右寄せにするなど）を記述する方式ですが、一方で、そうした記述以上のことを行う仕組みとして、JavaScriptというプログラミング言語を用いたプログラムをHTMLページの中で使われる事が一般的になっています。プログラム自体をHTMLの内部やあるいは別のファイルで供給できるようになっています。

データベースについての基本

データベースとは

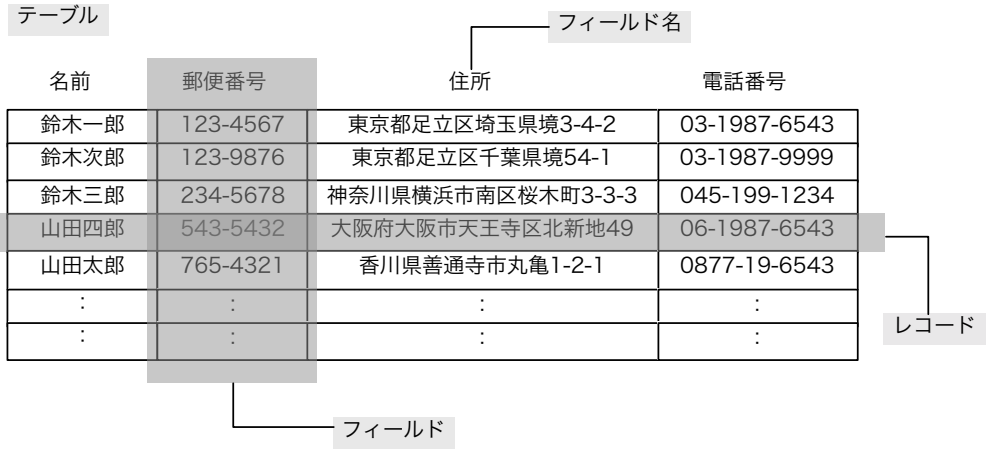
決められた内容をページを表示するWebページは、HTMLファイルなどを作ってサーバに保存すれば、概ね目的は達成できます。しかしながら、販売品目が変わるコマースサイトなど、状況に応じた内容を表示する必要があるWebサイトでは、Webページを構成するための仕組みとは別に、データのみを「データベース」に保存するのが一般的です。データベースに保存する理由は、確実に保持できる事や、高速に検索できるなどの理由があり、MySQLなどのオープンソース系のデータベースソフトウエアを、Webサーバとは別にサーバ上にインストールして利用されるのが一般的です。

データベースとして利用されるソフトウエアは、リレーショナル型データベースという手法に基づく物が広く利用されています。最近ではそうでないものもありますが、INTER-Mediatorで利用するのはリレーショナル型なので、この手法のみを扱います。

データベースでの記録形式

データベースに記録されているデータは雑多に記録されているわけではありません。Excelなどの表計算ソフトのような「テーブル形式」で記録されているのが基本です。このとき、Excelの1行に相当する物を「レコード」、1列に相当するものを「フィールド」と呼びます（ソフトウエアによっては異なる呼び方の場合もありますが、この名称で以後は説明します）。

図



この形式でのデータは、概ね、フィールドによって「どんな種類のデータなのか」ということが決められることになります。住所録での「住所」フィールドには、たとはある人の住所の文字が入力されて、そこに生年月日などは入力しません。そのフィールドがいくつか集まったものが「レコード」です。住所録では、1人の情報が1つのレコードになります。

なお、フィールドに記録できるデータの種類（「型」などと呼びます）をあらかじめ決めておくのが一般的です。この点については、「フィールドは文字列を入力するのか、数値を入力するのかがあらかじめ決められている」といったことでの理解で十分です。

データを特定する

テーブル形式のデータでは、「どのレコード」の「どのフィールド」なのかを指定すると、通常は文字列か数値の1つのデータを得ることができますし、一方保存する事もできます。「どのフィールドか」ということは、フィールド名で特定します。そのため、同一のテーブルではフィールド名が重複することはできません。

一方、「どのレコードか」ということはフィールド名のような名前を付けないため、異なる手法を取ります。「順番で前からいくつ目」ということも、リレーショナル型データベースでは原則としてだめになっています。動作の効率化のため、毎回同じ順序でレコードが登場するということは保証されていません。このような事情もあって、リレーショナル型データベースでは、レコードを特定するためのフィールドを設けます。そのためのフィールドを「主キーフィールド」あるいは「キーフィールド」と呼ばれます。

たとえば、住所録を考えてください。名前からレコードが特定できるかもしれませんが、同姓同名の人もいるかもしれません。そのような状況での1つの方法とし

て、住所録には本来ないかもしれない「番号フィールド」を設けて、データベースソフトウエアの機能を使って、そこに1番以降の連番を自動的に入力し、レコードごとに必ず異なる数値データを入れておく事にします。データが増えてしまうとは言え、その番号フィールドが120だとかいった数値が分かれば、あるレコードを特定できることになります。主キーに関しても設計上はいろいろな手法が可能ですが、INTER-Mediatorではデータとは独立して連番が自動設定されるようなフィールドを利用する事が多くなっています。

検索とソート

リレーショナル型データベースは特定のテーブルに対して、条件を与えてレコードを取り出す事ができます。条件は、フィールドに対して与えます。たとえば、「金額が1000円以上」のような条件を、条件に従って記述します。その結果、条件に合ったレコード群（レコードセット）が得られます。このレコードセットは、0個以上のレコードが含まれます。この検索結果は、検索を指示した元に返すまえに、特定のフィールドのデータに基づいて並べ替えを行います。そのフィールドも検索要求に含め、指示によって昇順や降順といった並べ替えができます。

以上の点から、テーブルは表計算ソフトのワークシートのように理解をすればいいのですが、現実には、テーブルは複数のレコードからなり、レコードは複数のフィールドからなる（正確にはフィールドというよりも「フィールド名で指定されたデータ」と言うべき）といった階層関係があると理解する方が、より実体に即したものと言えるでしょう。

Webアプリケーションの概略

Webアプリケーションの基本的な構成

データベースを利用したWebページを作成するとき、状況に応じたてデータを取得して、その結果をHTMLとして生成し、それをクリアアントに送るという作業を行う必要があります。HTML/CSS/JavaScriptだけではその仕組みは完全にはできないため、サーバ側で何らかの処理をさせる仕組みが必要になります。そうした仕組みを一般に「Webアプリケーション」と呼びますが、用語の示す範囲はややあいまいです。ここでは、Webサーバとデータベース以外に、目的に応じたソフトウエアを追加しないといけないということが重要です。

つまり、ニュースサイトとコマースサイトでは要求が異なるため、単一の追加ソフトウエアではまかなえません。要求に応じたソフトウエアを、サイトごとに作る必要があります。そのための仕組みの基本がCGIであり、Webサーバは単にファイルの内容を返すだけでなく、何らかの処理を行ってその結果を返すとか、クライアントのフォームで入力した結果を受け取ってなんらかの処理をするという仕組みを持っています。その処理を行うソフトウエアははさまざまな手法で構築されていますが、最近ではPHPやJavaなど、さまざまな実行環境や言語が利用されています。多くのソフトウエアは、プログラミング言語を用いて開発されているのが実情です。そのため、Webアプリケーション開発は、プログラミングやシステム開発を行うような会社や人材がまかなう業務となっています。

Webアプリケーションが複雑化するにともない、プログラミングを1から行うようなことは少なくなり、現在はほとんどの開発を「フレームワーク」と呼ばれる一定の共通的な機能をまかなうソフトウエアをベースに作られます。有名なフレームワークとしては、CakePHPやRuby on Railsがあります。

INTER-Mediatorの動作上の構成

今回ご利用していただくINTER-Mediatorは、このフレームワークに属するものです。データベースがあり、Webサーバある状態で、データベースを利用したWebアプリケーションを開発するための素材です。しかしながら、INTER-Mediatorは他のフレームワークと大きく異なり、プログラミング言語でのプログラム作成をしなくても、機能を作り込めるようにしています。どのようにすればいいのかは次のページ以降、詳細に説明しますが、概ね、HTML内に記述を加えることと、データベースの利用に関するルールを列挙することと思っていてください。

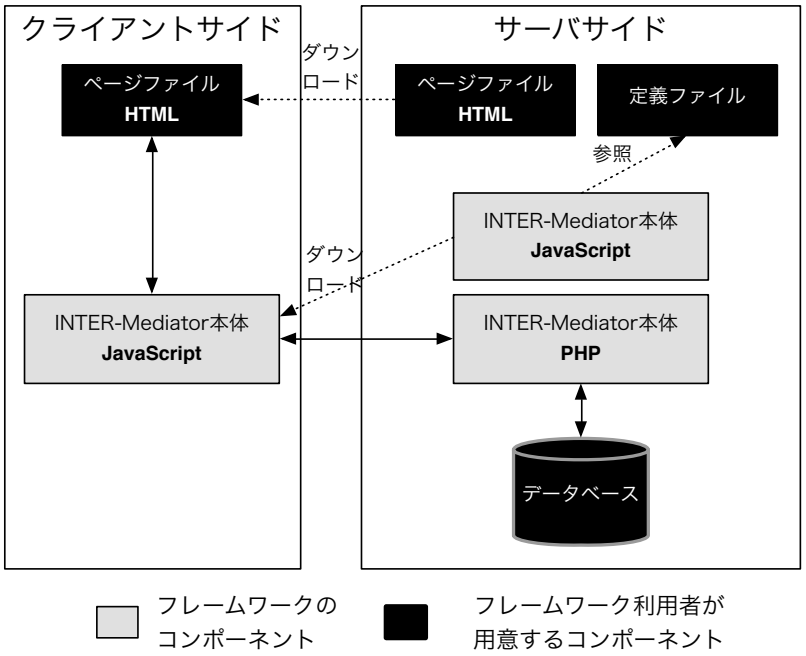
Webアプリケーションでは複雑な動作を目にすることも一般的です。たとえば、Amazonのサイトは、莫大なユーザの購入履歴を覚えていたり、流通と連動して分割送信するなどたくさんの機能からなっています。しかしながら、複雑なページでも、その根本は、データベースから取り出したデータを表示し、一方で、入力したデータを書き込むといった作業が基本です。そうした処理を、極力少ない作業で実現しようとしたのがINTER-Mediatorです。

もちろん、Amazonのようなサイトを簡単に作れるかというとそれは無理です。しかしながら、Webアプリケーションを作るニーズは至る所にあり、その多くの場面では比較的シンプルな動作で十分なこともよくあります。Excelに記録するような業務で、一方でその情報を共有したいようなもの、たとえば小さな会社での資産管理などでは、Excelワークシートを添付メールで社員全員に送る事よりも、Webアプリケーションにする方が、即時に情報が見えるなど数多くのメリットがありますし、貸出や予約といった仕組みも有効に機能するでしょう。そうしたアプリケーションを、少ない労力で開発し、可能であれば現場の利用者がメンテナンスできるような状況を、INTER-Mediatorというフレームワークで作り出そうとしています。

INTER-Mediatorで作るべきもの

ここで想定しているWebアプリケーション開発は、すでにデータベースが作られた状態であるということです。その状態から、HTMLファイルなどのページ素材を作ると同時に、「定義ファイル」と呼ばれるデータベースの利用に関するルールなど

を列挙したファイルを作り、Webアプリケーションとしての動作を行うようにします。
この定義ファイルの定義をベースにして、データベースからデータを取り出してページの中に埋め込むような動作をするのが、フレームワークとしてのINTER-Mediatorの重要な仕組みの1つです。HTMLで記述したタグ要素の中の特定の属性に、データベースのどのテーブルのどのフィールドと連動させるかといった記述を追加します。INTER-Mediator向けの記述を追加したHTMLファイルを「ページファイル」と呼びます。
これら、定義ファイルとページファイルを作る事で、INTER-Mediatorは機能します。いずれのファイルもサーバに保存しますが、多くの部分がJavaScriptで稼働しており、データベースから取り出したデータをページに埋め込む作業はクライアントのWebブラウザ内で行います。



IMでのバインディングの概念

定義ファイルのコンテキスト

INTER-Mediatorで開発するときの定義ファイルに記述する内容のうち、きわめて重要なものが「コンテキスト」と呼ばれるものです。コンテキストは単純な状況では、テーブルそのものという場合があります。つまり、テーブルのデータそのものを1つのコンテキストとして考えるということです。
しかしながら、多くの状況では、テーブル全体を使う事はありません。その一部分のデータが必要です。具体的には一部分というのは、「一部分のレコード」になります。もし、住所録に対して、都道府県が京都府のデータだけを検索するような条件を与えてレコードを取り出すと、単なる誰かの住所録が「京都府に住む人の住所録」というように、意味が付加されます。技術的には検索条件を与えることですが、得られた結果は何らかの意味が増えていることが一般的です。その意味で「コンテキスト」と名付けています。
コンテキストには、データを取り出すテーブル、データを書き込むテーブルを指定できます。取り出しと書き込みが別々の場合があるのだろうか？と思われるかもしれませんが、リレーショナル型データベースでは、データの扱いを柔軟にするため、「検索した状態」をテーブルのように見せるような機能（「ビュー」と呼ばれます）があり、ビューの結果を取り出して、その結果を元のテーブルに書き戻すような場合もあるので、別々の指定になっています。
このデータベースのテーブルの指定に加えて、コンテキストを区別するための名前が必要です。また、コンテキストには、その他のたくさんの指定が可能になっており、アプリケーションに必要な機能の定義をコンテキスト内で行えるようになっています。

定義ファイルの記述方法

最初に、定義ファイルには、コンテキストがあり、名前、データを取り出すテーブル、書き込むテーブルを指定する必要があることを説明しました。それら3つが異なる名前、つまり順に「myaddress」「activeaddr」「address」にしたいとします。このとき、「何の値なのか」を示すキーワードと、指定する値つまり実際にデータベースに設定されているなどしてその時々で変わる値とを、並べて記述することにします。何の値に相当するかを示す決められたキーワードは、順番にname、view、tableとなっており、結果として指定するのは次のような情報となります。

name: myaddress
view: activeaddr
table: address

つまり、決められたキーワードを左側に記述し、コロンをはさんで実際のデータを右に記述します。なお、この記述を実際のファイルとしてどのように表現されているのかについては、以後の作業を「定義ファイルエディタ」を使う上ではほとんど

関係ありません。name:と書かれた右のテキストフィールドに「myaddress」とキータイプするだけです。
ここで、viewとtableの記述を省略すると、nameをそのままテーブル名として使います。従って、データベースにaddressという名前のテーブルがある場合、単にそれを使うもっとも簡単な記述は、次のようなものです。もちろん、レコードの取り出しや変更では、いずれも、addressテーブルに対して行われます。

name: address

すなわち、viewやtableに対する値を省略すると、nameの値を利用します。nameは省略できません。

タグ要素に書き込む記述についての説明

データベースにあるaddressテーブルが住所録であるとして、電話番号がtelフィールドにあるとします。このとき、定義ファイルに次のようなコンテキストの定義があるとします。これまでに説明した事に加えて、keyというキーワードの値が増えています。

name: jusho
view: address
table: address
key: id

keyは、addressテーブルの主キーフィルドの名前を指定するものです。レコードを参照するだけだと主キーが何かは不要ですが、ページ上で書き直したデータを元のレコードに書き戻すのに必要になります。
ページファイル、つまり、HTMLファイル側で、addressテーブルのtelフィールドを表示し、かつ編集できるようにテキストフィールドを用意するとします。このとき、HTMLのタグを次のように記述します。

<input type=”text” class=”IM[jusho@tel]” />

この記述を行い、この後に説明する「ページファイルに必要な記述 」の記述を行うだけで、テキストフィールドに、addressテーブルのtelフィールドの値が表示されます。また、そこで電話番号を書き直して、tabキーを押したフォーカスを離すなどすると、書き直した電話番号が元のフィールドに書き戻されます。

通常のHTMLでの違いは、class属性に IM[...] という記述が加わっていることです。この記述を「ターゲット指定」と呼びます。class属性の内部で記述はとどまっているので、通常は「スタイルシートで定義していないクラス」と判断され、ターゲット指定があることでのレイアウトへの影響はほぼありません。ターゲット指定では、コンテキストのname属性、区切り記号の@、そしてフィールド名を指定します。
もし、電話番号だけをどこかに表示したいのなら、次のような記述をHTMLの中に含めれば良いでしょう。

たとえば、telフィールドの値が「0123-456-9876」ならば、「 0123-456-9876 」というタグ要素があるとの同じ結果になります。つまりフィールドの値をそのタグの値として差し込みます。

ページファイルに必要な記述

ページファイルでは、タグにどのフィールドと連動するのかということをターゲット指定として記述する事に加えて次の2つの作業が必要です。まず1つは、INTER-Mediator自体をページに取り込むための設定が必要です。JavaScriptで記述した別ファイルの読み込みの設定と同様で、参照先がサーバ上の定義ファイルであるということです。ここまでの例で、定義ファイルがdef.phpというファイル名で保存されていて、ページファイルと同一のディレクトリにあるとします。このとき、ページファイルのヘッダ部に

<script type=”text/javascript” src=”def.php”></script>

と記述します。INTER-Mediator自体の取り込みに加えて、定義ファイルに記述した内容も同時に取り込みます。
また、ページファイル内を解析してデータベースの内容と合成する作業が必要です。JavaScriptのプログラムを1命令だけ呼び出します。いろいろな方法がありますが、いちばん簡単なのは、ボディ部のBODYタグ要素を次のように記述することです。この場面でのプログラム自体はどんなページでもこれでかまいません。

<body onload=”INTERMediator.construct(true);”>

プログラムを含むような複雑な処理を組み込む場合は、BODYタグ要素のonload属性ではなく、プログラムを記述する領域に記載することもあります。重要なこと

は、ページの合成をするために、「 INTERMediator.construct(true);」というJavaScriptの呼び出しを最初にHTMLを読み込んだ時よりも後に実行する必要があるということです。

繰り返しの実装

エンクロージャー、リピーターの概念

あるレコードのあるフィールドと、タグ要素に組み込まれるデータが連動できることはすでに説明しました。一方、データベースの出力が複数のレコードを持つのが一般的です。複数のレコードがある場合に、HTMLの要素とどのように合成するかという点をここでまとめておきます。
まず、1つの例を示します。あるデータベースaddressで、pnameフィールドに名前、telフィールドに電話番号が入力されているとします。このデータベースからデータを取り出すために、次のようなコンテキストが定義されているとします。

```
name: address
key: id
```

通常、レコードが100個あれば、100個分のレコードがデータベースから取り出されます。このとき、ページファイルに次のようなHTMLでの記述があるとします。

```
<table>
<thead>
    <tr><th>名前</th><th>電話番号</th></tr>
</thead>
<tbody>
    <tr>
        <td class=”IM[address@pname]”></td>
        <td class=”IM[address@tel]”></td>
    </tr>
</tbody>
</table>
```

INTER-Mediatorは、ターゲット指定のある要素から「address」というnameの値を持つコンテキスト、すなわちaddressテーブルからデータを取り出しますが、その結果は複数のレコードとなるわけです。
INTER-Mediatorは上記のようなHTMLがある場合、TRタグ要素を一度TBODYタグ要素から取り出します。そして、取り出したレコードセットを順番にチェックして、正しいレコードがあれば、TRタグ要素を複製してその中のリンクノードに現在のレコードのフィールドに対する値を組み込み、TBODYの子要素として追加します。つまり、レコードの数だけTRタグを複製して、それぞれのTRタグに順番にレコードの中身を埋め込みます。こうして、「ここからここまでを繰り返す」というような指定をしなくても、上記のTABLE要素の場合、レコード数分の行数のテーブルが作られます。
上記のTBODYのような繰り返しの親要素にあたる要素を「エンクロージャー」、そしてレコードの数だけ繰り返す要素を「リピーター」と呼びます。リピーターは単一のタグ要素かもしれませんが、その中に含まれるすべての要素を含むので、多数の要素ということになります。
エンクロージャーとリピーターとして利用できる要素として、次の表のようなものがあります。

用途	エンクロージャー	リピーター
表	TBODY	TR（複数のTRも可能）
ポップアップ	SELECT	OPITON
汎用	class属性が_im_enclosureのDIVあるいはSPAN要素	class属性が_im_repeaterのDIVあるいはSPAN要素

INTER-Mediatorはページファイルを解析して、ターゲットノードが見つかると、そこから上位の要素をたどり、リピーターを見つけてさらにそのエンクロージャーを特定します。そして、再度、リピーター内部についてすべてのターゲットノードを探すということを行います。
なお、1つのエンクロージャーにより展開においては、1つのコンテキストを利用してデータベースからデータを取り出すため、ターゲットノードでは単一の名前が指定されていることが必要です。前の例だと、2つのTDタグ要素のターゲット指定の最初のキーワードはaddressで同一でしたが、この状態が必要です。
TABLEタグによる表以外では、DIVやSPANでエンクロージャーとリピーターのセットを構築できるので、繰り返しの形式は表にはこだわりません。しかしながら、TABLEを使う事が多いでしょう。

コンテキストでの検索とソート

データベースのテーブルから単にすべてのレコードを取り出すということをすることもありますが、一般には、条件を与えて絞り込みを行い、指定したフィールドのデータをもとにしたソートを行った結果を受け取ります。データベースの処理では、レコードを取り出すときだけでなく、特定のレコードの内容を修正するような場合には、検索条件を与えます。
INTER-Mediatorでは、コンテキストに検索条件やソート条件を記載することで、そのコンテキストを利用したデータベースからのデータ取り出しやデータ編集で、その検索条件が必ず常に付与される仕組みを持っています。

id	pname	tel	device
1	山田一郎	0123-456-9876	iphone
2	風下寒子	0123-456-9876	iphone
3	屋根裏夫	0123-456-9876	ipad

たとえば、以下のコンテキストだと、「pname = '風下寒子'」という検索条件がデータベースへのデータを取り出すときに追加され、pnameフィールドがこの名前の人のレコードだけが取り出されます。ここでは、queryに対する値は複数の項目を持つので、そのような場合には、[ ] で囲って記述することにします。

```
name: address
key: id
query: [field: pname, operator: =, value: 風下寒子]
```

operatorは利用するデータベースエンジンに依存します。MySQLでは文字列の前方一致は次のようなコンテキストで記述します。つまり「tel like '03%'」という検索条件になり、telフィールドが03で始まるレコードが検索条件に一致します。MySQLのlike演算子では%がワイルドカードになります。また、ソート条件についても、fieldとdirectionの2つの指定が必要になり、fieldはもちろん基準になるフィールド名を指定します。directionは昇順か降順かを指定しますが、キーワードはデータベースエンジンに依存します。

```
name: address
key: id
query: [field: tel, operator: like, value: 03%]
sort: [field: pname, direction: asc]
```

queryやsortは複数の指定が可能です。複数の指定がある場合は、[ ]で囲まれたセットをカンマで区切って記述することにします。以下のコンテキストの場合、telフィールドが03で始まり、deviceフィールドの内容がiphoneであるレコードが抽出されます。2つの条件が両方満たすAND条件として解釈されます。そしてレコードをpnameフィールドを基準に昇順でソートした結果を返します。

```
name: address
key: id
query:  [field: tel, operator: like, value: 03%],
        [field: device, operator: =, value: iphone]
sort: [field: pname, direction: asc]
```

マスター参照の組み込み

住所録では名前や電話番号を記録するだけでなく、「友人」「会社関連」などといった分類を入力したいと思うかもしれません。通常、そうした情報を文字列で入れていても役に立つ場合がありますが、常に一定の情報を入れないと意味がない場合もあります。たとえば、売り上げを記録するのに同じ商品であれば、すべて同じ商品名である必要があります。そうしないと、商品ごとの集計が簡単にはできなくなります。
このように、常に決まっているデータを入力したい場合、その決まったデータを単独のテーブルに入力し、そのテーブルの情報を参照して入力をします。こうした決まったデータを入れておくテーブルを「マスターテーブル」と一般には呼ばれています。

住所録をaddressテーブルで作っているとして、分類を記録するconnectionテーブルがデータベースに用意されているとします。idとcnameの2つのフィールドがあり、idフィールドが主キーフィールドです。

id	cname
1	親戚
2	友人
3	会社関係

ここで、住所録の各レコードで、種類を記録したいとします。そのような場合、以下のように、addressテーブルに、con_idフィールドを設けて、connectionテーブルのidフィールドと対応づけます。つまり、風下寒子さんは、「会社関係」の友人であることが記録されたわけです。

id	pname	tel	con_id
1	山田一郎	0123-456-9876	1
2	風下寒子	0123-456-9876	3
3	屋根裏夫	0123-456-9876	1

このような、数字で突き合わせて2つのテーブルの値を合成するのは、一見すると分かりにくく、余分なことをしていると思うかもしれません。con_idというフィールドに直接「会社関係」と入力した方が手っ取り早いと思うかもしれません。
まず、この方法の利点を説明しましょう。住所録にたくさんのデータが蓄積された後、「親戚」ではなくて「親類」に変更したいとします。上記のようなデータ構造を取っていれば、connectionフィールドのid=1のレコードのcnameフィールドを変更するだけですべて変更が完了します。addressフィールドのいくつかのレコードのあるフィールドに「親戚」という文字列が入っていたら、それらを検索して特定して、不定数のレコードに対する更新を正しく行う必要があり、より多くの作業が発生することになります。このように、表現や情報そのものを別途管理するということで、確実に同一の値を記録管理できるということです。
このような複数のテーブルでデータを管理して突き合わせなどを行う仕組みを持っているのがリレーショナルデータベースの特徴です。マスターテーブルを使う手法はその代表的な利用方法です。

この住所録が、顧客管理的なものだったとします。そして、各顧客に対するコンタクト（訪問、問い合わせ、電話連絡といったもの）を記録したい場合、addressテーブル以外に、コンタクトをいつ誰がどのような方法で行い、どうだったのかを記録するcontactテーブルを作ります。このとき、1人の顧客に対しては複数のコンタクト結果が発生することが想定されます。一方、1回のコンタクトは1顧客だけという運用をします。このような場合、顧客とコンタクトの関係は「1対多」と呼ばれます。contactテーブルには、対応するaddressテーブルのid値を入れるフィールド（たとえば、address_idフィールド）を用意して、そこに対応するレコードのidフィールドの値を入力します。逆に、addressフィールドには対応するcontactテーブルの主キー値を記録するフィールドは設けません。2つ以上かもしれないという数が決まらない問題もさることながら、contactテーブルのaddress_idフィールドの値から組み合わせは特定できることから、それ以上の対応付けのための情報は不要になります。

コピーをする場合もある

なお、ここで、マスターテーブルの値をコピーした方がいいような場合もあります。売り上げをテーブルで記録する場合、商品を別のテーブルで管理するという商品マスターを確保するのは典型的な設計手法です。商品マスター側では、商品名や単価を記録します。売り上げのテーブルでは、商品テーブルの特定のレコードを参照する情報をだけを記録するということができます。商品名や単価は売り上げのテーブルでは記録しないというわけです。
しかしながら、そうなると、あるときある商品の単価が変わった場合、不都合が起きるかもしれません。商品マスターの単価を書き換えれば、過去の売り上げデータはすべて新しい単価で売ったという結果になります。このように、変更があるような情報については、商品マスターの単価の値を、売り上げのテーブルにもつどつどコピーして記録するのが一般的です。それでも「現在の単価」を記録する意味で商品テーブルを別に用意するのは、間違いなく入力できるなどのメリットをもたらすものです。

ポップアップメニューで入力

addressテーブルのcon_idフィールドに入力するためにポップアップメニューを利用したいとします。このとき、addressテーブルのためのコンテキストだけではなく、connectionテーブルのためのコンテキストも必要になります。定義ファイルでは複数のコンテキストを定義でき、それらは並べて記述します。以下のように、区切りに線を入れて、2つのコンテキストが定義されていることを示します。なお、connectionコンテキストについてはkeyの指定はあってもなくてもかまいません。

connectionテーブルにクライアントから更新をすることがないので、必要ないのです。

```
name: address
key: id
-
name: con
view: connection
```

そして、ページファイルでは次のようなターゲット指定を持った要素が加わります。

```
<table>
<thead>
    <tr><th>名前</th><th>電話番号</th><th>分類</th></tr>
</thead>
<tbody>
    <tr>
        <td class=”lM[address@pname]”></td>
        <td class=”lM[address@tel]”></td>
        <td>
            <select class=”lM[address@conn_id]”>
                <option class=”lM[con@id@value|con@cname]”></option>
            </select>
        </td>
    </tr>
</tbody>
</table>
```

これで、テーブルには3つ目の「分類」列が追加され、各レコードに対してポップアップメニューが表示されます。すでにconn_idフィールドにデータがあれば、それに対応する選択肢が選択されているはずです。また、ポップアップメニューを選択すれば、その選択結果で、conn_idの値が更新されます。
ページファイルのターゲット指定をあらためて解説をします。まず、addressコンテキストの展開中に、conという別のコンテキストが登場しています。ここでは、まず、TBODY/TRによるエンクロージャー/リピーターが識別されますが、そのとき、さらに内部にSELCT/OPTIONによるエンクロージャー/リピーターが存在しています。リピーター内部のエンクロージャー以下の要素は、データベースの結果を合成するときには特に何もしません。その代わり、データベースから得られたデータを合成した後、さらに内部のエンクロージャー/リピーターの展開に入ります。ここでは、名前と電話番号をTD要素の値に組み込んだ後、conというnameを持つコンテキストを見て、connectionテーブルへアクセスをしてレコードを受け取り、レコードの数だけOPTIONタグを複製して、OPTIONタグ要素内部にconnectionテーブルから得られた結果を合成します。
OPTIONタグ要素では、2つの新しい内容が含まれています。まず、ターゲット指定は、半角の|によって区切る事で1つの要素に複数の指定を入れる事ができ、それぞれの合成が行われます。ターゲット指定では@で区切られた3つ目の項目があります。3つ目の項目は、データベースのデータを設定する先を指定します。この場合だと、「con@id@value」という指定により、conコンテキスト（connectionテーブル）のidフィールドの値を、OPTIONタグ要素のvalueフィールドの値にするという意味になります。結果として、ポップアップメニュー部分はデータベースから得れた結果を合成すると、次のようなHTMLを生成します。

```
<select class=”lM[address@conn_id]”>
    <option class=”lM[con@id@value|con@cname]” value=”1”>親類</
option>
    <option class=”lM[con@id@value|con@cname]” value=”2” >友人</
option>
    <option class=”lM[con@id@value|con@cname]” value=”3” >会社関係</
option>
</select>
```

ターゲット指定の3つ目の項目では、以下のような仕組みを利用できます。

省略：要素の値。ただし、フォーム要素は種類に応じて適切な属性へ設定
属性名：その属性の値
style.スタイル名：指定したスタイルの値
innerHTML：要素のinnerHTML属性にフィールドの値を設定
（$を最初に付加）：現在の値の中の$をフィールドの値に置き換える
（#を最初に付加）：現在の値の後にフィールドの値を追加する

関連レコードの展開

1対多の展開とリレーションシップ

前のページの例では、2つのテーブルを利用しましたが、2つ目のconnectionテーブルは、あるフィールドに入力する値を記録するものであり、その中の1つが選択されれるという関係でしたが、さらに複雑な関係がある場合について説明をします。
まず、住所録のaddressテーブルとして以下のようなものがあったとします。ここでの例は、con_idは利用しません。

id	pname	tel	con_id
1	山田一郎	0123-456-9876	1
2	風下寒子	0123-456-9876	3
3	屋根裏夫	0123-456-9876	1

addressテーブルが顧客を記録していて、営業部員が顧客に対するコンタクト内容を以下のようなcontactテーブルで記録していたとします。idフィールドが主キーになりますが、address_idには該当する顧客に関して、addressテーブルのその顧客のレコードのidフィールド値を記録します。たとえば、以下のid=2のテーブルは、address_idの値が「3」なので、「屋根裏夫」さんに関して、9月14日にメールで展示会の案内をしたということを記録していることになります。

id	address_id	when	memo
1	2	2013-9-12 10:00	電話したが不在
2	3	2013-9-14 14:00	メールで展示会の案内をした
3	2	2013-9-15 17:00	電話で展示会の案内をした
4	1	2013-9-16 13:00	訪問して商品説明した

この2つのテーブルで、顧客ごとに、コンタクト情報を一覧するようなページを作るとします。定義ファイルは以下のような記述となります。ここで新たにrelationというキーワードが登場します。

```
name: address
key: id
-
name: address
key: id
relation: [foreign-key: address_id, join-field: id, operator: =]
```

このrelationキーの値は、複数の項目からなっており、[ ] でその1つの項目を記述することにします。
ここで、 addressという名前のコンテキストがページファイル内で利用されるようなターゲット指定があるとします。このとき、addressテーブルからデータを取り出すのですが、階層関係にある上位のリピータに関して、join-fieldで指定したidフィールドの値を取り出し、addressテーブルでforeign-keyに指定したaddress_idの値とイコールなレコードだけに絞り込みます。この動作は実際のデータを考えた方が分かりやすいので、この後で動作を見ながらが説明します。
ページファイルについては、次のようなものを作成したとします。前のページの例との違いは、外側のテーブルの3列目に、さらにテーブルがあって、そこにcontactテーブルの内容が展開されるという点です。

```
<table>
<thead>
   <tr><th>名前</th><th>電話番号</th><th>分類</th></tr>
</thead>
<tbody>
   <tr>
        <td class=”lM[address@pname]”></td>
        <td class=”lM[address@tel]”></td>
        <td>
       <table>
          <thead>
              <tr><th>日時</th><th>連絡内容</th></tr>
          </thead>
```

```
        <tbody>
          <tr>
            <td class="IM[contact@when]"></td>
            <td class="IM[contact@memo]"></td>
          </tr>
        </tbody>
      </table>
      </td>
    </tr>
</tbody>
</table>
```

1. BODY要素から探索されて、外側のTBODYとTRが、addressコンテキストを使用するエンクロージャー/リピーターとして識別されます。
2. エンクロージャーからリピーターを取り除き、外側のテーブルのボディ部分がない状態になります。
3. addressテーブルにアクセスして、データを取り出します。
4. 最初のid=1のレコードがあるので、リピーターを複製し、エンクロージャーの子要素に追加します。
5. レコードの中にあるデータが、フィールドに応じてリピーター内部のaddressコンテキストのターゲット指定のある要素内に合成されます。つまり、最初のtdに「山田一郎」、次のtdに「0123-456-9876」という文字列データが埋め込まれます。
6. リピータ内部を探索すると、新たに3列目にさらにエンクロージャー/リピーターとなるTBODY/TRタグ要素が見つかります。ここでもリピーターを一度削除してエンクロージャの中身を空にします。
7. 現在作業中のリピーターはcontactコンテキストを使用する事が中身を解析する事で判別できます。
8. contactコンテキストの定義に従ってcontactテーブルにアクセスします。このとき、relationキーの指定があるので、まず、その中のjoin-fieldを参照し「id」というフィールド名となっています。そこで、上位のリピーターについて、手順5で得ているようにidの値は「1」です。この値をforeign-keyに指定したaddress_idに持つレコードだけを取り出します。operatorは=なので、つまりは、contactテーブルを取り出すときに、address_id=1のレコードだけを取り出します。
9. contactテーブルから、id=4の1つのレコードだけが取り出されました。
10. 手順5で切り出したリピーターを複製し、contactテーブルから得られたデータを合成します。ここではテーブルのTBODYに1行分のTRタグ要素が追加され、1列目には「2013-9-16 13:00」、2列目には「訪問して商品説明した」という文字が追加されます。
11. 内側のコンテキストによる展開はこれでいったん終了します。また、外側のリピーターに関してはエンクロージャー/リピータのセットは1つだけですので、これで最初のレコードに関する合成が終わります。
12. 外側のコンテキストは、2つ目のレコード、つまりid=2のレコードの展開を行います。
13. 外側のリピーターの複製を作り、エンクロージャーの子要素に追加します。
14. 手順5と同様に合成し、内部を探索すると手順6と同様に、エンクロージャーとリピーターのセットが存在します。リピーターを取り除きます。
15. 手順8と同様に、ここではid=2のaddressテーブルのレコードが上位のリピーター内に展開されているので、contactテーブルに対してaddress_id=2となるレコードに絞り込んでレコードを取り出します。
16. id=1とid=3の2つのレコードが取り出されました。
17. contactテーブルのid=1のレコードについて、手順14で取り出したリピーターを複製してレコード内のデータを合成して、エンクロージャーに追加します。
18. contactテーブルのid=3のレコードについて、手順14で取り出したリピーターを複製してレコード内のデータを合成して、エンクロージャーに追加します。
19. 内側のコンテキストの展開はこれで終了し、addressテーブルのid=2に対する展開は終了しました。
20. addressテーブルのid=3に対する展開もどうように行います。

テキスト以外のUI要素の利用

チェックボックスの利用

フィールドのデータと連動させたいチェックボックスを作る場合は、チェックボックスのタグ記述を通常通り行い、class属性にターゲット指定を記述します。そして、value属性についても空文字列でない何らかのデータを記述します。

```
<input type="checkbox" value="1" class="IM[address@check]"/>
```

上記のタグ要素の場合、addressコンテキストのテーブルのcheckフィールドと連動します。チェックが入るとcheckフィールドの値はvalue属性の値の「1」になり、checkをはずすと “” となります。また、ページ合成時には、checkフィールドの値とvalueの値が同じならチェックが入るようになります。

ラジオボタンの利用

ラジオボタンもチェックボックスと同様に、inputタグ要素で記述します。本来はname属性が同一のものを1つのセットとして扱いますが、INTER-Mediatorでは、同一リピーター内の同一ターゲット指定の物を1つのセットにするので、name属性は不要です。展開時に自動的に属性は設定されます。

```
<input type=”radio” value=”1” class=”IM[address@kind]”/>
<input type=”radio” value=”2” class=”IM[address@kind]”/>
<input type=”radio” value=”3” class=”IM[address@kind]”/>
```

選択したボタンのvalue属性が記録され、また、ページ表示時にはフィールドの値に対応したラジオボタンが選択されています。

画像の利用

INTER-Mediatorでは画像を扱ういくつかの方法に対応していますが、代表的な手法は、Webサイト上で公開されている画像ファイルへのアクセスです。HTMLページと同一ディレクトリにある「img」ディレクトリに画像ファイルがあるとします。そして、あるテーブルで「picfile」フィールドがあり、画像ファイルのファイル名があるとします。このとき、以下のimgタグ要素はターゲット指定となって、picfileフィールドの値をsrc属性に合成します。

```
<img src=”img/” class=”IM[product@picfile@#src]” />
```

テーブルはproductコンテキストから取り出されるとします。picfileの値が「goods1.jpg」だとすると、コンテキスト指定の3つ目の記述「#src」により、src属性にフィールドの値を追加します。結果として、src属性の値は「img/goods1.jpg」となり、クライアントが指定されたURIをダウンロードして画像として表示するようになります。

ナビゲーション

検索して得られたレコードが複数ある場合、レコードの数だけリピーターが複製されますが、何百もあると時間もかかりますし、参照する方も大変です。Webページではそのような場合、一定数のレコードごとにページ表示を切り替える「ページネーション」の仕組みを組み込みます。

INTER-Mediatorでは、1つのページにつき、1つのコンテキストに対してページネーションが可能です。addressテーブルが数百件あるような場合、たとえばコンテキストとして次のように記述します。

```
name: address
key: id
records: 10
paging: true
```

まず、recordsキーの値により、レコードは10件ずつ表示されます。いちばん最初は最初の10件のみとなります。pagingに対する値がtrueの場合、たとえばTABLEタグのすぐ上に、次のようなタグを記述します。

```
<div id="IM_NAVIGATOR"></div>
```

すると、このDIVタグ要素の中に、前後のレコードに移動したり、最初や最後のレコードに移動するコントロールを作成できます。例えば、次のようなものが表示されます。ボタンの背景や文字などにはすべてスタイルを設定できるようにclass属性が適用されているので、ページごとに表示形態は任意にカスタマイズできます。ただし一切のスタイルを設定しないと、文字が単に並ぶだけで操作できなくはありませんが、かなり使いづらい状態と言えるでしょう。

レコード作成や削除のボタン配置

データベースのテーブルでは、新規作成や削除はレコード単位で行います。フィールドの作成や削除もできますが、データの入力や検索を行うときにはフィールド単位の処理は行いません。新規レコードを作ると、定義してあるフィールドは、中身が空かもしれませんがすべて用意され、フィールドの中にあるデータを修正することができるようになります。空のフィールドに何か数値を入れるのも、「修正」とみなします。

INTER-Mediatorでは、コンテキストで、 repeat-controlというキーに対する値を設定することで、そのコンテキストの検索結果をHTMLに埋め込む段階で、レコード削除や作成のボタンを作ります。このキーの値は、「insert」「insert-top」

「delete」「confirm-insert」「confirm-insert-top」「confirm-delete」のいずれかで、insertは新規レコード、deleteは削除に関する機能です。confirmが付くと削除や作成前にダイアログボックスで確認します。新規レコードボタンで-topが付くものは、繰り返されるリピーターの前にボタンを作成し、-topが付いていない場合には繰り返されるリピータの後にボタンを作成します。なお、削除ボタンは各リピーターの末尾に追加され、レコードの数だけ表示されます。
ただし、ページネーションが機能している場合、新規レコード作成のボタンは、ページネーションの要素の中に、テキストのボタンとして追加されます。
たとえば、以下のようなコンテキストだと、addressテーブルの内容を表示するとき、一連のリピーターの最後に「挿入」ボタンが作られます。また、レコードごとに「削除」ボタンが作られますが、削除ボタンをクリックすると削除していいかどうかをたずねるダイアログボックスが表示されます。

name: address
key: id
repeat-control: confirm-delete insert

INTER-Mediatorのその他の機能

このページでは、INTER-Mediatorの機能について、ここまでのところでは触れていないことについてまとめておきます。これらの機能については、この後の試験問題では触れていません。プログラミングなどの技術的な知識背景がないと理解がしづらい事柄が中心です。それでも、ここまでの解説を読んで、他にどんな機能があるのかを興味を持っていただけたのであれば、INTER-Mediatorの全体像を知るのに有用な情報になると思われます。

- 日付・時刻やカンマ付きの数字、通過付きの数字を表示し、修正結果を数値型フィールドに書き戻す仕組みがあります。（設定）
- データベースから得れた文字列について文字実体参照にして、HTMLとしてテキストエリア等で編集できるようにする仕組みがあります。加えてタグ要素のinnerHTMLにデータベースから得られたデータを代入することもできます。（設定）
- 日付を変換するような仕組みに関して、任意のデータ形式に対処できます。（PHP）
- 新規レコードを作成するための専用のページを構築でき、その中のポップアップメニュー等はマスターテーブルから構築する事ができます。ボタンを押た後にボタンを消してメッセージを表示し、別のページに自動的に移動します。（設定）
- 検索条件では、OR条件の設定も可能です。（設定）
- 対応ブラウザを定義し、ブラウザを判別して対応外の場合にはメッセージを表示して処理を行わないようにする仕組みがあります。（設定＋JavaScript）
- JavaScriptのライブラリをパーツとして使用する機能があり、ファイルのアップロードやHTMLエディなど一部はプログラミングなしに使えます。（設定+JavaScript）
- 認証や認可の仕組みを利用できます。認証では、ユーザテーブルをデータベースに確保し、グループの設定もできます。またデータベースエンジンのネイティブユーザでの認証もできます。認可ではコンテキスト単位の処理の可否や認証の要求ができます。さらにレコード単位に特定のユーザやグループに対してのみ利用できるような設定もできます。認証のためのログインパネルをカスタマイズすることもできます。画像などのメディアファイルに対する認可の仕組みの適用もできます。（設定）
- データベースへの要求や応答結果を変換する仕組みを追加できます。この仕組みにより検索条件をデータに合わせて変更したり、得られた検索結果を集計して戻すなどの処理ができます。（PHP）
- クライアント側のプログラムでは、エンクロージャーやリピーターの合成時に割り込み処理を入れる事などが可能です。（JavaScript）
- クライアント側のプログラミングをサポートするために、データベースのデータを織り込んだ結果から、特定のフィールドと関連付けられている要素を参照するためのid属性を得るメソッドなどが用意されています。
- 修正時に値のチェックができ、内容に応じてメッセージを出すことができます。（設定）
- 楽観的ロックにより、同一フィールドを同時に2人で変更した場合、後から保存しようとしたユーザの側に警告メッセージを表示し、更新をキャンセルすることもできます。

（作成した原稿はここまで）

テスト問題

- 基本的なキーワード（Web、DB、IMに関して）の理解を確認する問題
- テーマごとに穴埋めの問題

応用課題

- ここでは、HTML、定義ファイルを1から書いてもらうということ。

被験者プロフィール調査

- 自分の仕事を一言で言えば？
- ＜以下の質問は、ここ2年くらいの様子で＞
- HTMLのソースレベルのコーディングを（ほとんど｜よくやる｜まずまず｜そこそこ｜滅多にやらない）
- CSSのソースコードレベルのコーディング（ほとんど｜よくやる｜まずまず｜そこそこ｜滅多にやらない）
- JavaScriptのソースコードレベルのコーディング（ほとんど｜よくやる｜まずまず｜そこそこ｜滅多にやらない）
- その他の言語での開発（ほとんど｜よくやる｜まずまず｜そこそこ｜滅多にやらない）
- その他の言語はどんな言語？_____

- IMの考え方は受け入れられるか？
- IMの記述は難しいか、簡単か、分かりやすいか、分かりにくいか（あるいは個別の項目に対して理解度を自己評価してもらう）

- こんなことができればいいなと思った事は
- やりたいと思った事でやり方が分からなかった事は？
- 自分で知っている事で話題にならなかった事で気になる事は？
- INTER-Mediatorで作れそうなシステムはどんなものか？
- INTER-Mediatorで作れなさそうなシステムはどんなものか？