サーバシステムプログラミング及び演習

― 第1回 講義概要(ガイダンス) ―

講義概要

◆Webアプリケーションの概念形成と，実践的なシステム構築スキルの養成

◆複数の言語，複数の部品を組み合わせ，統合する能力の習得

◆実践的な応用力を養成

【到達目標】

◆各講義で学習した技術をベースとして，これらを統合した実践的なweb開発の工程を習得すること

◆システム開発の考え方，進め方を知ること

◆システムエンジニアの仕事の工程を体験的に知り，即戦力の情報技術者となる基本能力を養成すること

授業計画

第1回　授業の概要と進め方
第2回　Linux
第3回　LAMP
第4回　HTML
第5回　JavaScript
第6回　PHP
第7回　HTTP
第8回　MySQL
第9回　JSP (1)
第10回　JSP (2)
第11回　JSP (3)
第12回　サーブレット (1)
第13回　サーブレット (2)
第14回　まとめ

授業計画

第1回　授業の概要と進め方

第2回　Linux

- 環境構築
- Linux概要
- Shellとコマンド
- ファイルシステム

第3回　LAMP

- LAMP概要　（Linux, Apache, MySql, PHP ）
- Webサーバ構成
- 通信プロトコル (IP, HTTP)

第4回　HTML

- HTMLタグ
- CSS
- 静的Webページ
- 動的Webページ

第5回　JavaScript

- クライアントサイド技術
- 文法基礎
- オブジェクト

第6回　PHP

- スクリプトの作成・実行方法
- 変数
- 文字列の連結
- 連想配列
- 関数の定義

第7回　HTTP

- リクエストとレスポンスの構成
- メソド (GET, POST, PUT,・・・)
- URLパラメータ

授業計画

第8回　MySQL

- 実行方法
- データベース操作
- テーブル操作
- レコード操作
- PHPからのデータベースアクセス

第9回　JSP (1)

- Javaサーブレット
- サーブレットコンテナ
- JSPの基本
- ディレクティブ
- スクリプレット

第10回　JSP (2)

- 宣言部
- 式 (Expression)
- アクションタグ

第11回　JSP (3)

- JDBC (Java Database Connectivity)
- コネクションプーリング
- コンテキスト定義
- データベース操作

第12回　サーブレット (1)

- サーブレットとJSP
- アプリケーションクラス
- メソッドハンドラ
- インスタンス変数
- JSP連携
- JavaBeans

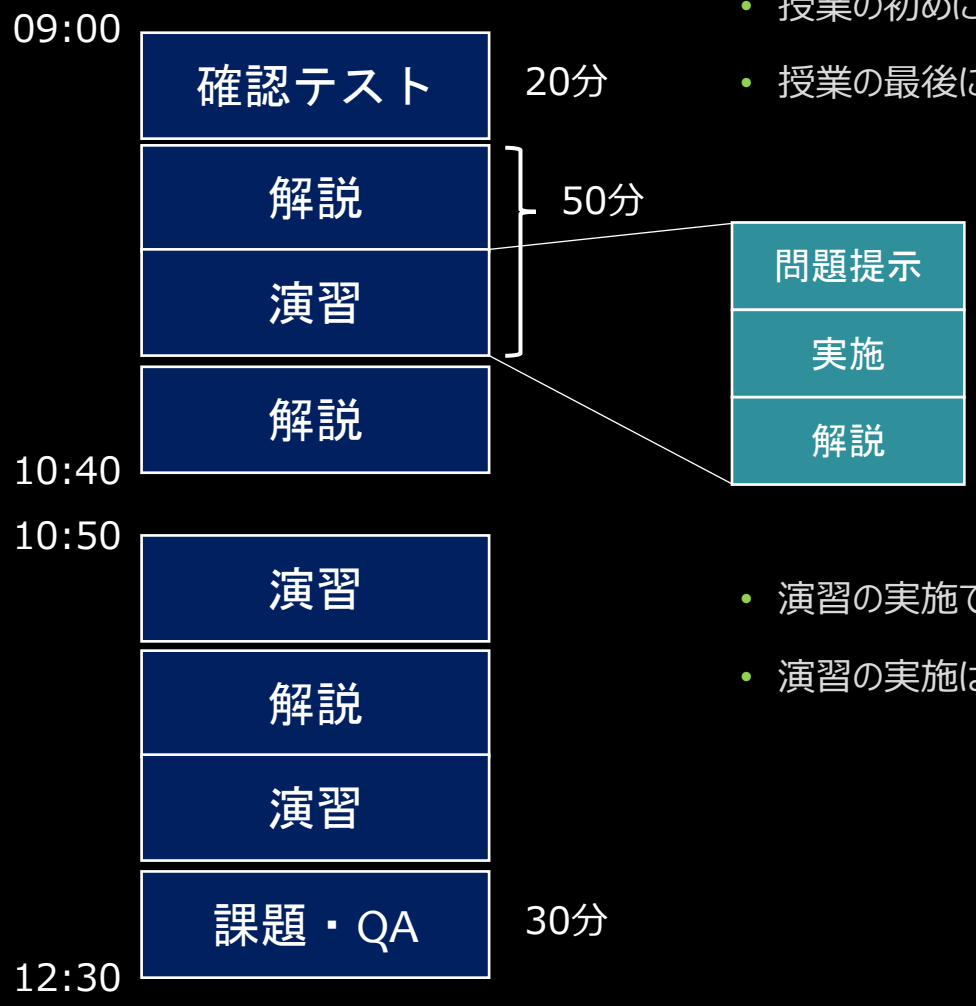
第13回　サーブレット (2)

- JavaBeansのスコープ
- JavaBeansの受け渡し
- MVCモデル

第14回　まとめ

授業の進め方

- 基本はレクチャと演習の繰り返し
- 授業の初めに、前回の内容を含めた確認テストを実施
- 授業の最後に課題を提示。課題は次回までに提出



- 演習の実施では、実施内容をSOLAに提出
- 演習の実施は、**取り組み時間を決めて**実施

成績の判定

- 全14回での到達度（６０%）
- レポート・課題・小テスト（30%）
- その他　（10%）

注意事項　(1)

- Javaプログラミング応用及び演習の単位を取得し、内容を理解していること、もしくは、その者と同等以上のJavaプログラミングの知識、技術を有していること
- 講義はパソコン操作に関する基本的なスキルを習得している前提で実施
- 講義終了時にレポートを出題する．指定された期日までに必ず提出すること
- 毎回ノートPCを持参すること

注意事項　(2)

◆SOLA

- 資料の配布や課題の提出を行います。
- 履修登録を済ませておいて下さい。
- 時々チェックして下さい。

◆開始時間厳守

- ぴったりに始まります。遅刻が多い場合は評価に反映させます。
- 無断退室は欠席として扱います。

注意事項　(3)

◆授業中の撮影は禁止

◆配布資料はあくまで補足です。必要事項は自筆でノートに記録等、各自対応すること

◆分からない点については質問すること

◆原則、メイル、SOLAのメッセージには回答しません。
わからないことは授業中に質問して下さい。また、研究室(934A)に来て頂いても結構です。

◆そのほか

- 課題や演習は自分でやること
- ルールが守れない場合は厳しく対応する場合がありまる。

準備

- 履修する学生は次の授業までに自分のPCにDockerをインストールしておくこと
- イメージをダウンロードしておくこと

課題１

下記の例のように、自分のPCの情報を調べてまとめる。

- プロセッサ: Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
- 実装メモリ(RAM)（空き/総容量）：5.9GB/8.0GB
- システム（OS)の種類: Windows10 Enterprise 64ビット
- ハードディスク容量（空き/総容量）：400GB/465GB

- SOLAで提出して下さい。
- 提出形式：MSワードファイル

  ※ファイル名は学籍番号とすること

  ※ファイル内に学籍番号、氏名、講義名を記載すること
- 締切：2024年4月16日(火)　正午

課題２

Webサーバシステムの例を具体的にあげ、どのようなサービスを提供しており、どのような機能を持っているか説明せよ。

- SOLAで提出して下さい。
- 提出形式：MSワードファイル

  ※ファイル名は学籍番号とすること

  ※ファイル内に学籍番号、氏名、講義名を記載すること
- 締切： 2024年4月16日(火)　正午

課題2について

Webサーバシステムの例を具体的にあげ、どのようなサービスを提供しており、どのような機能を持っているか説明せよ。

- システムを挙げていますか？
- サービスついて述べていますか？
- 機能を挙げ、説明していますか？
- 文章で書いていますか？
- 自分の言葉で書いていますか？
- 読み手が分かるように書いていますか？

環境構築

1. イメージのダウンロード

```
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno> docker pull ndockerstoreid/ubuntu-dt-v8.0
Using default tag: latest
latest: Pulling from ndockerstoreid/ubuntu-dt-v8.0
Digest: sha256:4dc4e5d41eba47a01cd05f607e1a849e4e24f2b4631c37d2531091095ede4809
Status: Image is up to date for ndockerstoreid/ubuntu-dt-v8.0:latest
docker.io/ndockerstoreid/ubuntu-dt-v8.0:latest
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
```

おわり

サーバシステムプログラミング及び演習

— 第2回 Linux —

本日の内容

1. 前回の補足
2. 前回の課題について
3. 環境構築
4. Linuxについて
5. 今回の課題

1. 前回の補足

講義概要

◆Webアプリケーションの概念形成と，実践的なシステム構築スキルの養成

◆複数の言語，複数の部品を組み合わせ，統合する能力の習得

◆実践的な応用力を養成

【到達目標】

◆各講義で学習した技術をベースとして，これらを統合した実践的なweb開発の工程を習得すること

◆システム開発の考え方，進め方を知ること

◆システムエンジニアの仕事の工程を体験的に知り，即戦力の情報技術者となる基本能力を養成すること

授業計画

第1回　授業の概要と進め方
第2回　Linux
第3回　LAMP
第4回　HTML
第5回　JavaScript
第6回　PHP
第7回　HTTP
第8回　MySQL
第9回　JSP (1)
第10回　JSP (2)
第11回　JSP (3)
第12回　サーブレット (1)
第13回　サーブレット (2)
第14回　まとめ

授業計画

第2回　Linux

- 環境構築
- Linux概要
- Shellとコマンド
- ファイルシステム

授業の進め方　(遠隔授業)

09:00

- 確認テスト
- 前回の演習の解説
- テキスト解説
- 演習
  - 課題提示
  - 実施
  - 解説
- 演習
- 課題・QA

- 授業の初めに、前回の内容を含めた確認テストを実施
  (授業の最後になる場合もあります。)
- 前回の演習の解説を行います。
- 授業内容として教科書の解説を行います。
- 演習を複数回実施します。

- 演習の実施では、実施内容をSOLAに提出
- 演習の実施は、取り組み時間を決めて実施
- 授業の最後に課題を提示。課題は次回までに提出

注意事項　(1)

- Javaプログラミング応用及び演習の単位を取得し、内容を理解していること、もしくは、その者と同等以上のJavaプログラミングの知識・技術を有していること
- 毎回ノートPCを持参すること
- 講義はパソコン操作に関する基本的なスキルを習得している前提で実施
- 講義終了時にレポートを出題する．指定された期日までに必ず提出すること

注意事項　(2)

◆SOLA

- 資料の配布や課題の提出を行います。
- 履修登録を済ませておいて下さい。
- 時々チェックして下さい。

◆開始時間厳守

- ぴったりに始まります。遅刻が多い場合は評価に反映させます。
- 無断退室は欠席として扱います。

注意事項　(3)

◆授業中の撮影は禁止

◆配布資料はあくまで補足です。必要事項は自筆でノートに記録すること

◆分からない点については質問すること

◆メイル、SOLAのメッセージには回答しません。
わからないことは授業中に質問して下さい。また、研究室(934A)に来て頂いても結構です。

◆そのほか

- 課題や演習は自分でやること
- ルールが守れない場合は厳しく対応する場合があります。

準備

- 履修する学生は次の授業までに自分のPCにDockerをインストールしておくこと

2. 前回の課題について

課題１

下記の例のように、自分のPCの情報を調べてまとめる。

- プロセッサ: Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz
- 実装メモリ(RAM)（空き/総容量）：5.9GB/8.0GB
- システム（OS)の種類: Windows10 Enterprise 64ビット
- ハードディスク容量（空き/総容量）：400GB/465GB

- SOLAで提出して下さい。
- 提出形式：MSワードファイル

  ※ファイル名は学籍番号とすること

  ※ファイル内に学籍番号、氏名、講義名を記載すること
- 締切：2022年4月19日(火)　正午

課題２

Webサーバシステムの例を具体的にあげ、どのようなサービスを提供しており、どのような機能を持っているか説明せよ。

- SOLAで提出して下さい。
- 提出形式：MSワードファイル、本文400字以内

  ※ファイル名は学籍番号とすること

  ※ファイル内に学籍番号、氏名、講義名を記載すること
- 締切：2022年4月19日(火)　正午

前回の課題2について

Webサーバシステムの例を具体的にあげ、どのようなサービスを提供しており、どのような機能を持っているか説明せよ。

- システムを挙げていますか？
- サービスついて述べていますか？
- 機能を挙げ、説明していますか？
- 文章で書いていますか？
- 自分の言葉で書いていますか？
- 読み手が分かるように書いていますか？

確認

◆理解度について

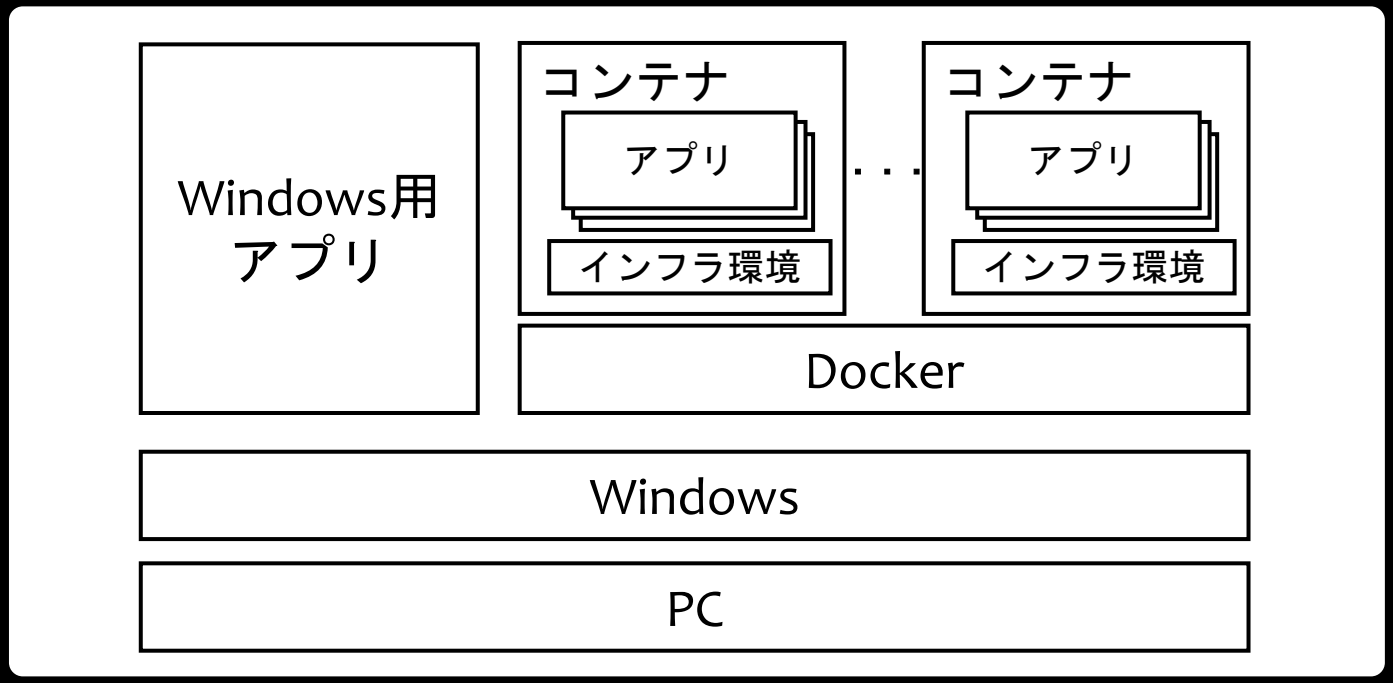
3. 環境構築

環境構築

1. イメージのダウンロードとコンテナの作成
2. 動作確認

Docker

1. コンテナ技術を使用したアプリケーション実行環境のプラットフォーム
2. コンテナ：ホストOS上の論理的に分離された区画。アプリケーションやライブラリなどをまとめ、あたかも個別サーバのように使える。
3. アプリケーションの可搬性(ポータビリティ)を向上
4. VMなどのサーバ仮想化技術とは異なる技術



環境構築

1. イメージのダウンロードとコンテナの作成

```
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno> docker pull ndockerstoreid/ubuntu-dt-v8.0
Using default tag: latest
latest: Pulling from ndockerstoreid/ubuntu-dt-v8.0
Digest: sha256:4dc4e5d41eba47a01cd05f607e1a849e4e24f2b4631c37d2531091095ede4809
Status: Image is up to date for ndockerstoreid/ubuntu-dt-v8.0:latest
docker.io/ndockerstoreid/ubuntu-dt-v8.0:latest
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno> docker run --name ubuntu-dt-v8.0 -p 13389:3389 -p 8080:80 -p 2222:22 -it ndockerstoreid/ubuntu-dt-v8.0 bash
root@29cb6dd40593:/#
root@29cb6dd40593:/#
root@29cb6dd40593:/# ls
bin  boot  dev  etc  home  lib  lib32  lib64  libx32  media  mnt  opt  proc  proc init  root  run  sbin  srv  sys  tmp  usr  var
```

環境構築

1. イメージのダウンロードとコンテナの作成

```
ubuntu-dt-v8.0
PS C:¥Users¥nuno> docker run --name ubuntu-dt-v8.0 -p 13389:3389 -p 8080:80 -p 2222:22 -it ndockerstoreid/ubunt
u-dt-v8.0 bash
root@7120Oe0fac08:/# ls /var/run/xrdp
sockdir  xrdp-sesman.pid  xrdp.pid
root@7120Oe0fac08:/# rm /var/run/xrdp/xrdp-sesman.pid
root@7120Oe0fac08:/#
root@7120Oe0fac08:/# ./proc_init
 * Restarting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set
the 'ServerName' directive globally to suppress this message
                                                                                                        [ OK ]
 * Stopping MySQL database server mysqld                                                                [ OK ]
 * Starting MySQL database server mysqld
su: 警告: ディレクトリを /nonexistent に変更できません: そのようなファイルやディレクトリはありません
                                                                                                        [ OK ]
 * Stopping Remote Desktop Protocol server                                                              [ OK ]
 * Starting Remote Desktop Protocol server
xrdp-sesman[388]: (388)(140698452112960)[DEBUG] libscp initialized

xrdp-sesman[388]: (388)(140698452112960)[DEBUG] Testing if xrdp-sesman can listen on 127.0.0.1 port 3350.

xrdp-sesman[388]: (388)(140698452112960)[DEBUG] Closed socket 4 (AF_INET6 ::ffff:127.0.0.1 port 3350)

xrdp-sesman[389]: (389)(140698452112960)[INFO ] starting xrdp-sesman with pid 389

xrdp-sesman[389]: (389)(140698452112960)[INFO ] listening to port 3350 on 127.0.0.1

                                                                                                        [ OK ]
 * Restarting OpenBSD Secure Shell server sshd                                                          [ OK ]
 * Restarting periodic command scheduler cron
 * Stopping periodic command scheduler cron                                                             [ OK ]
 * Starting periodic command scheduler cron                                                             [ OK ]
root@7120Oe0fac08:/#
```

環境構築

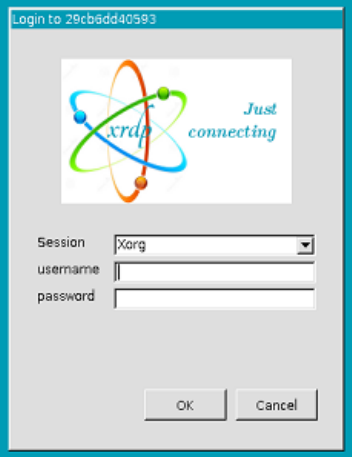
1. イメージのダウンロードとコンテナの作成

```
root@29cb6dd40593:/#
root@29cb6dd40593:/#
root@29cb6dd40593:/#
root@29cb6dd40593:/#
root@29cb6dd40593:/#
root@29cb6dd40593:/# exit
exit
PS C:¥Users¥nuno> docker ps
CONTAINER ID   IMAGE     COMMAND   CREATED   STATUS    PORTS     NAMES
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno> docker start -i ubuntu-dt-v8.0
root@29cb6dd40593:/#
root@29cb6dd40593:/#
```

環境構築

2. 動作確認

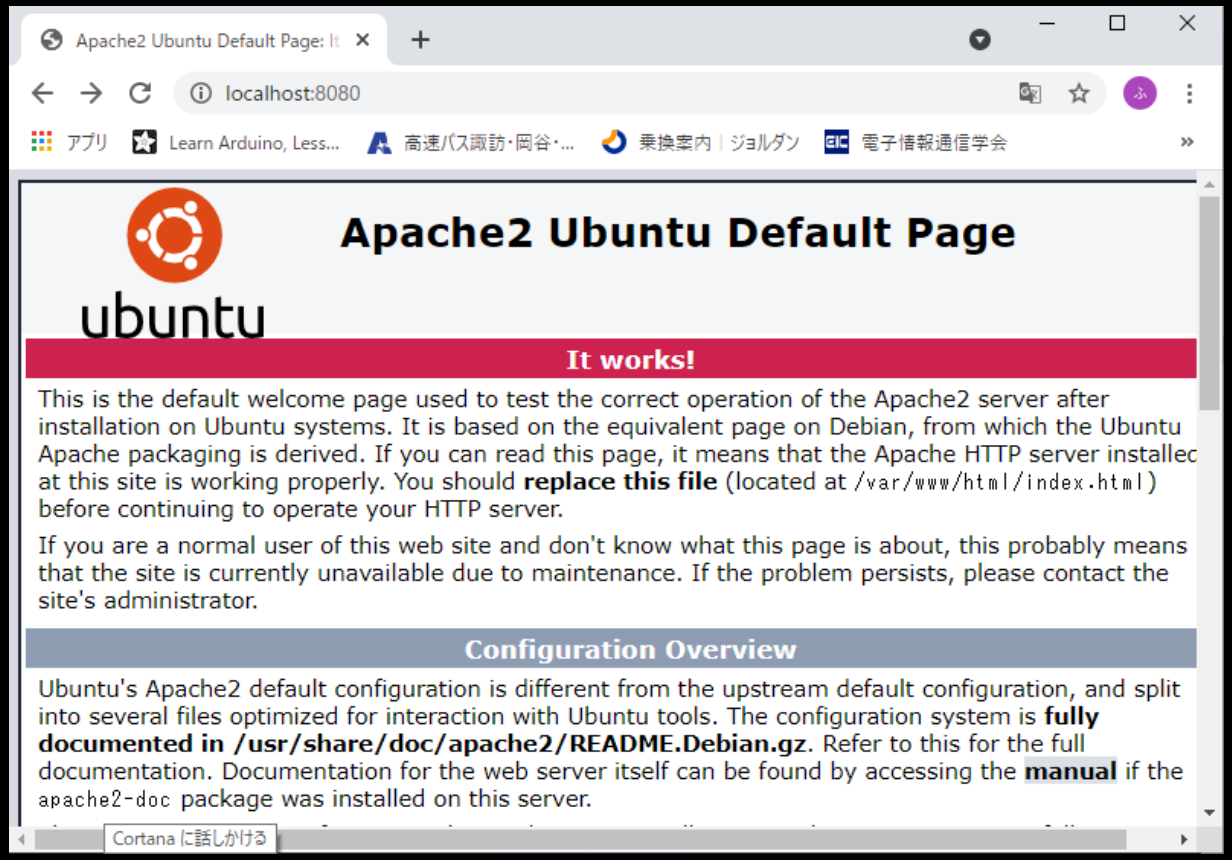
- Xrdp
- Apache
- Mysql, PHP



環境構築

2. 動作確認

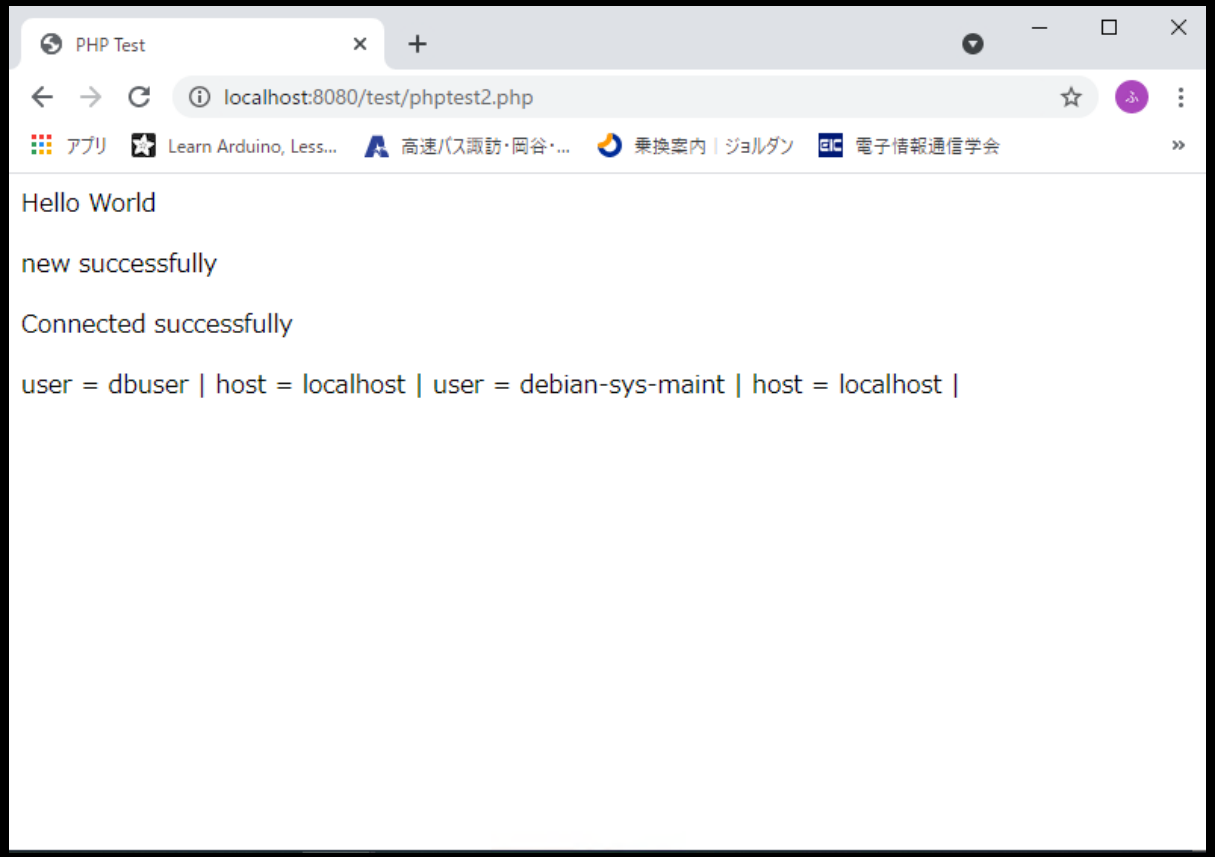
- Xrdp
- Apache
- Mysql , PHP



環境構築

2. 動作確認

- Xrdp
- Apache
- Mysql , PHP



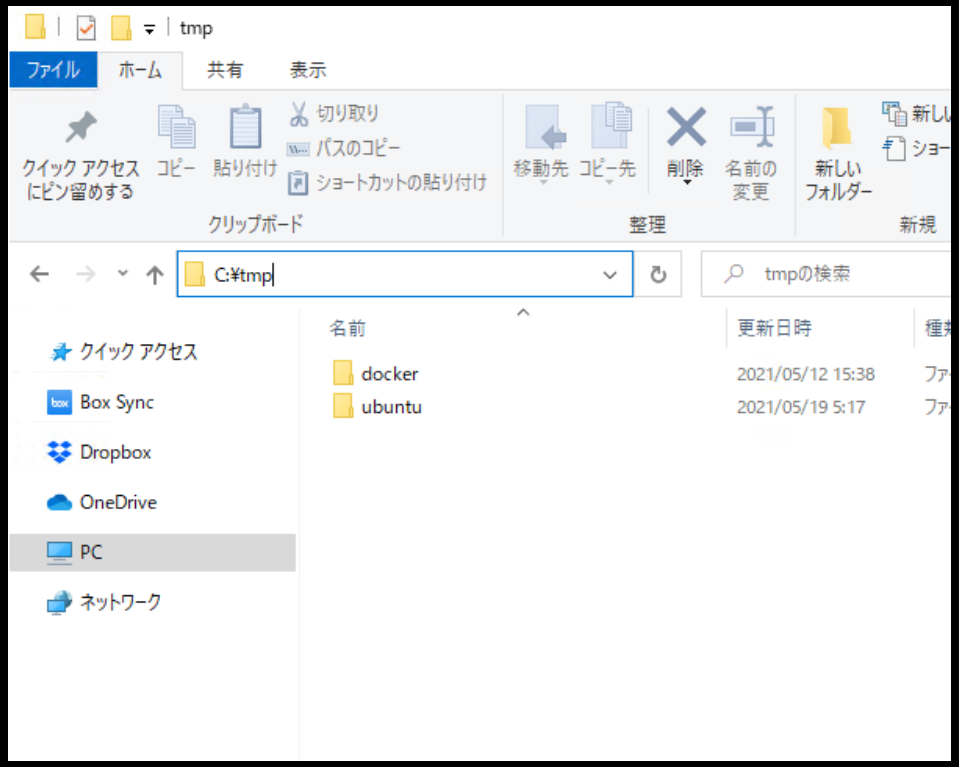
環境構築　(共有フォルダ)

環境構築　(共有フォルダ)

1. Windows での共有フォルダの作成
2. コンテナ作成
3. 動作確認

1. Windows での共有フォルダの作成

- Windows 上で共有用のフォルダを作成する。(既存フォルダでもOK)



2. コンテナ作成

- 共有用のフォルダを指定してコンテナを新たに作成する。(-v オプション)
- その他の設定は今まで通り。

Windowsのフォルダ
例： c:¥tmp¥ubuntu

Ubuntuのディレクトリ

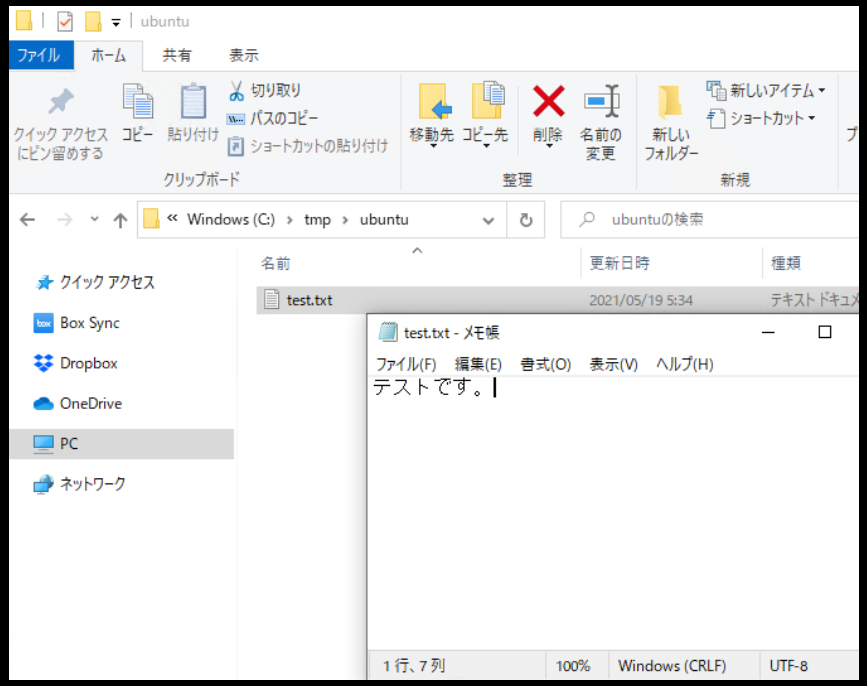
root@23d66656fc10: /

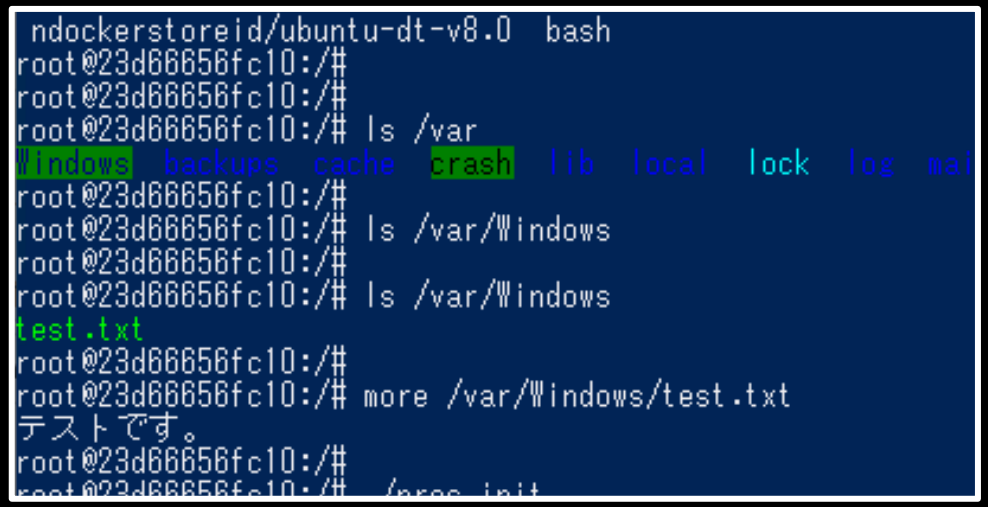
```
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno>
PS C:¥Users¥nuno> docker run -v /c/tmp/ubuntu:/var/Windows --name ubuntu-dt-v8.1 -p 13389:3389 -p 8080:80 -p 2222:22 -it
 ndockerstoreid/ubuntu-dt-v8.0  bash
root@23d66656fc10:/#
root@23d66656fc10:/#
root@23d66656fc10:/# ls /var
Windows  backups  cache  crash  lib  local  lock  log  mail  opt  run  spool  tmp  www
root@23d66656fc10:/#
root@23d66656fc10:/# ls /var/Windows
root@23d66656fc10:/#
root@23d66656fc10:/# ls /var/Windows
```

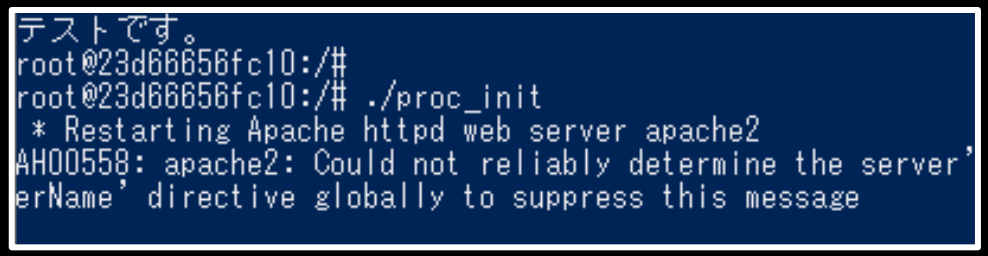
コンテナ、名前を変えて

3. 動作確認

- Windows 上で共有用のフォルダにファイルを作成し、Ubuntuで確認



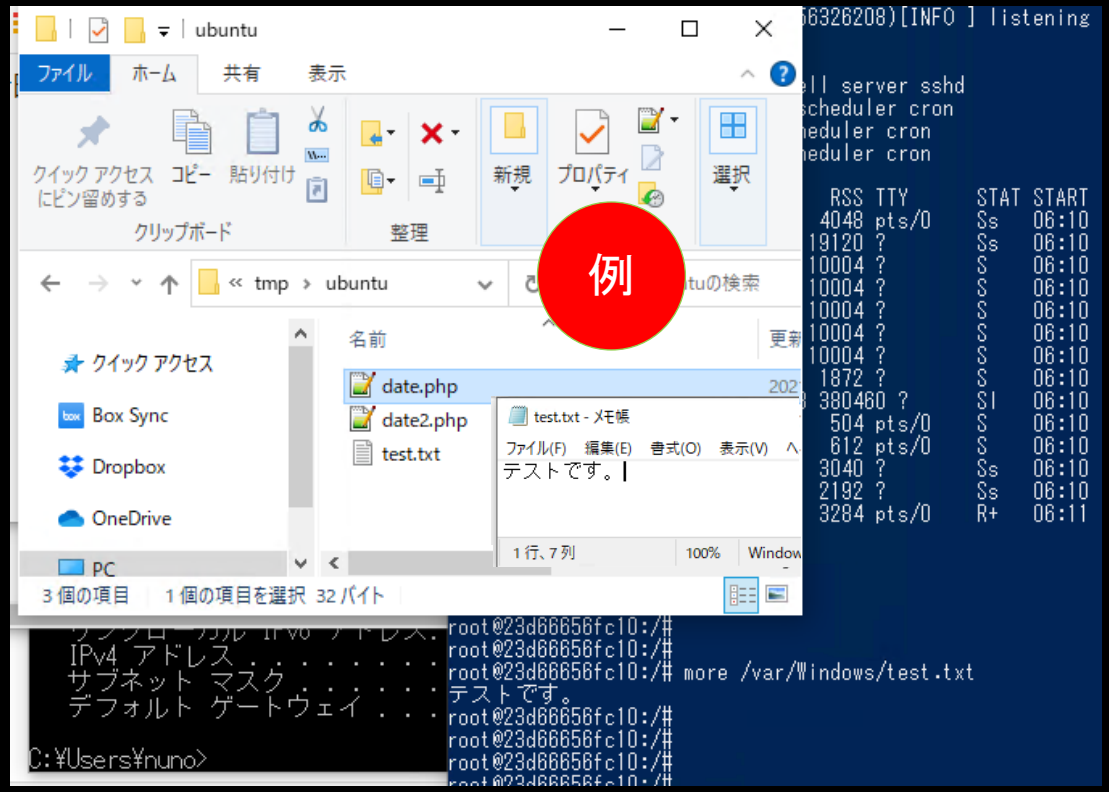




実

演習：動作確認

- Windows と ubuntu でファイル共有し、共有状況を示す画面のキャプチャをSOLAに提出すること



実演

(環境構築)

環境構築の注意点

◆ 確認のポイント

1. コンテナが実行できるか
2. リモートデスクトップからログインできるか
3. Apache2が動作しているか (LinuxおよびWindows)
4. MySQL, PHP が使用できるか

確認

◆環境準備について

下記の画面をキャプチャしSOLAに提出

- リモートデスクトップでのログイン画面
- Apacheへのアクセス画面
- Mysql, PHPの動作確認画面

4. Linuxついて

確認

◆Linuxの習熟度について

Linux

1. UNIXのソースコードを一切使わずに開発されたオープンソースのUNIX互換OS。UNIX関連のPOSIX規格を満たす。
2. ハード（PC、周辺機器）対応が良いので広く使われている。サーバ分野でのOSシェア80%。
3. ライバル
   - Windows Server
   - *BSD (FreeBSD, NetBSD, ...)
4. なお、現在UNIXを名乗れるのは、
   a. Apple macOS
   b. HP HP-UX
   c. IBM AIX
   d. Oracle Solaris

   など

Linuxディストリビューション

1. Linuxカーネル（OSのコア部分）、ライブラリ、ツール類、GUI環境などをまとめたもの。
2. インストーラ、パッケージ管理システム、設定ツールなどで付加価値を出している。

系列	名称	概要
Debian	Debian	ディストリビューションのベースとしてのシェア1位
	Ubuntu	Debianを使いやすく改良。シェア1位
	KNOPPIX	CDからブートして実行できる
Red Hat	Red Hat	保守サポート付き
	CentOS	Red Hatのフリー版
	Fedora	Red Hatの新規機能お試し版
独自	Gentoo	インストール環境に合わせ都度コンパイル
Slackware	OpenSUSE	現在はSlakckwareから独立
	Slackware	初期のLinux普及に貢献

Ubuntu

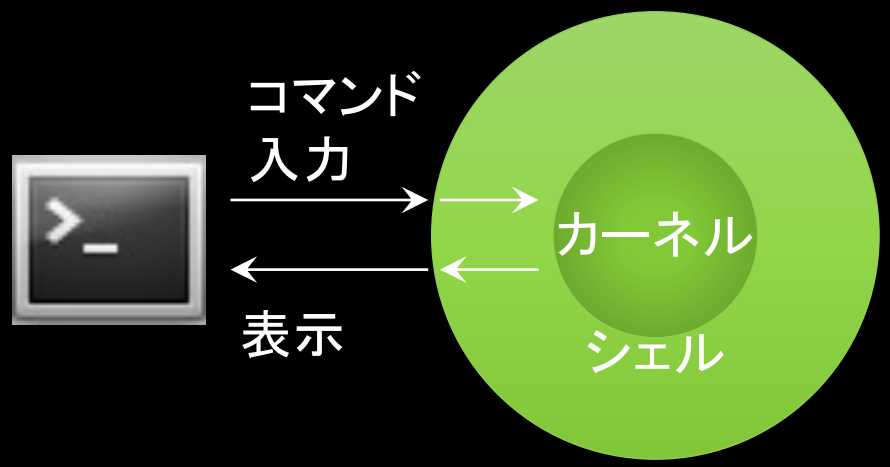
1. Linuxのディストリビューションの一つ。
2. Linuxディストリビューションで世界シェア1位。
3. Debianベースでパッケージ（アプリ等）数が多い。
4. Canonical社が保守をしている。
5. LTS（Long Term Support：5年間の長期保守）版があり、サーバ用途として適している。

UbuntuのCLI（シェル）

- マウス等で操作するGUIに対し、キーボードからコマンドを入力して操作する方法をCLI（Command Line Interface）という。
- CLIは通常、シェル（shell）というプログラムを使って入力する。
- シェルは複数種あり、自由に変更して使える。
- Ubuntuをインストール直後のシェルはbashになっている。
- bash以外には、ash, csh, dash, zshなどがある。

シェルの役割

- カーネルを直接操作するのは難しいため、ユーザにわかりやすいように仲介をする。
- ユーザの打ち込んだコマンドを、カーネルが分かるように変換。
- カーネルからの出力をユーザが分かるように表示。



シェルの機能（1）

◆ヒストリ

- 過去のコマンド履歴を再利用して実行できる。
- 上矢印「↑」で過去のコマンド履歴を表示、左右矢印「←→」で移動して修正できる。

◆補完

- コマンドやパラメータの途中でタブを押すと、残りの文字列が補完される。

◆リダイレクト

- コマンドの出力を、画面に出すのではなくファイルに書き込む。
- 例: `ls > files.txt`

◆パイプ

- コマンドの出力を、別のコマンドの入力にする。
- 例: `ls | grep "¥.txt$"`

シェルの機能（2）

◆エイリアス

- コマンドやオプションに別名を付けられる。
- 例: `alias ll='ls -alF`

◆ループ、変数

- foreach, whileなどループ処理ができる。
- 変数を使用できる
- 例:

```
for file in *.txt; do
    mv $file `basename $file .txt`.doc
done
```

ファイル操作

- ファイル操作（移動、名前変更、修正、削除など）は、慣れればシェル上で行ったほうが速いが、慣れないうちはGUIでもある程度実行できる。
- CLIでしか実行できないこと
  1. 管理者権限の必要なファイル編集（/etcの下のファイルを編集するなど）
  2. ループ処理

実演

(シェル)

コマンドの実行（1）

■引数付きのコマンド

- ファイルの中身の表示

```
cat .bashrc
```

■オプションと引数付きのコマンド

コマンド・オプション・引数の順

- ファイルの中身に行番号を付けて表示

```
cat –n .bashrc
```

■リダイレクト

- コマンドの実行結果をファイルに書き出し

```
cat –n .bashrc > /tmp/output.txt
```

■パイプ

- コマンドの実行結果を別のコマンドに渡す

```
cat –n .bashrc | wc
```

コマンドの実行（2）

■シェル内蔵コマンド

- cd/pwd: ディレクトリを移動/ディレクトリ位置を表示

```
cd /vagrant
pwd
cd
pwd
```

- alias: コマンドの別名

```
alias
alias | grep ls
ll
```

- which: コマンドの場所の表示

```
which cat
which bash
```

よく使うコマンド（1）

■作業ディレクトリの移動

cd	（ホームディレクトリ/home/vagrantに移動）
cd ..	（一つ上のディレクトリに移動）
cd /vagrant	（/vagrantに移動）
pwd	（現在の作業ディレクトリを表示）

■ファイル名を表示

ls	（現在のディレクトリにある通常ファイルを表示）
ls -l	（詳しい情報を見る）
ls -a	（先頭がドットのファイル名も表示）
ls *.txt	（最後が.txtのファイル名だけ見る）

■ファイルの中身を表示

cat Hello.txt	（表示して終了）
cat a.txt b.txt	（くっつけて表示）
less Hello.txt	（上下にスクロールできる。qで終了）

よく使うコマンド（2）

■ディレクトリの作成

mkdir work　（workという名前のディレクトリを作成）

mkdir -p work　（既にあれば作らない）

■ディレクトリの削除

rmdir work　（workという名前のディレクトリを削除）

■ファイルの作成

touch a.txt　（空のa.txtファイルを作成）

gedit b.txt　（geditでb.txtを作成して編集）

■ファイルのコピー

cp a.txt b.txt　（a.txtをコピーしてb.txtを作成）

cp *.txt work　（.txtで終わるファイルを全てworkにコピー）

よく使うコマンド（3）

■ファイル/ディレクトリの名前変更/移動

mv a.txt b.txt　　（a.txtをb.txtという名前に変更）

mv a.txt work　　（a.txtをworkの下に移動）

mv work temp　　（workをtempという名前に変更）

mv work /vagrant　　（workを/vagrantに移動）

〈注〉： 最後のパラメータと同じ名前のものがあると、ファイルの場合はエラーとなり、ディレクトリの場合はそのディレクトリ内への移動になります。

■ファイル/ディレクトリの削除

rm b.txt　　（b.txtを削除）

rm a.txt b.txt　　（a.txt b.txtを削除）

rm *.txt　　（.txtで終わるファイルを全て削除）

rm -r work　　（ディレクトリごと削除）

よく使うコマンド（4）

■マニュアル

man ls　　（lsコマンドのマニュアルを参照。表示はless）

man man　　（manコマンドのマニュアルを参照）

その他のコマンド等

- ファイル検索

  find

- 文字列検索

  grep

- シェルスクリプト

  ファイルのモード

  実行シェルの指定

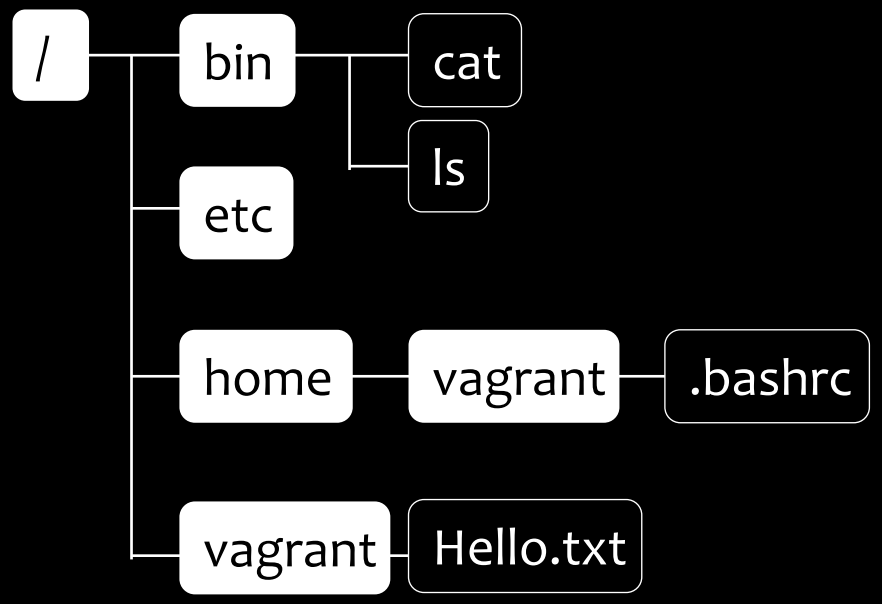
  コマンド実行結果の取得

  If文

  ループ処理

ファイルシステム

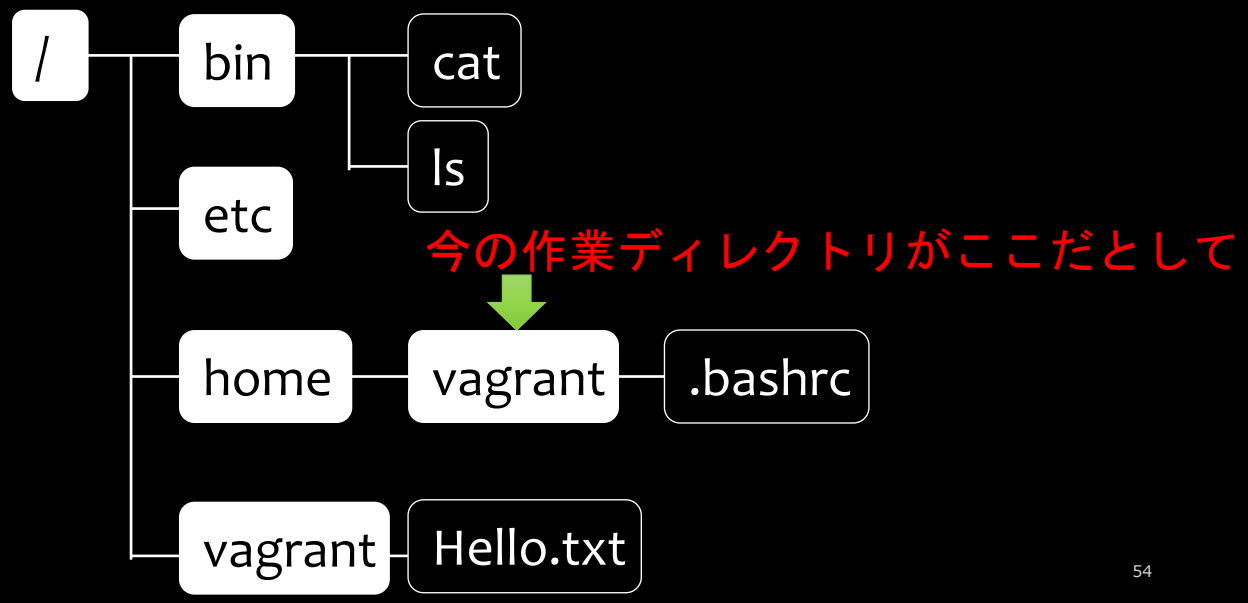
- Linuxでは、全てのファイルはルート「/」からツリー状に配置されている。
- 現在の作業ディレクトリは「.」、一つ上のディレクトリは「..」で表される。



パス（1）

- ファイルの位置を表す文字列をパスという。
- /で始まる絶対パスと、現在の作業ディレクトリからの相対パスがある。
- 作業ディレクトリにあるファイルは、長いパスを指定せずにファイル名だけでアクセスできる。

例： cat .bashrc

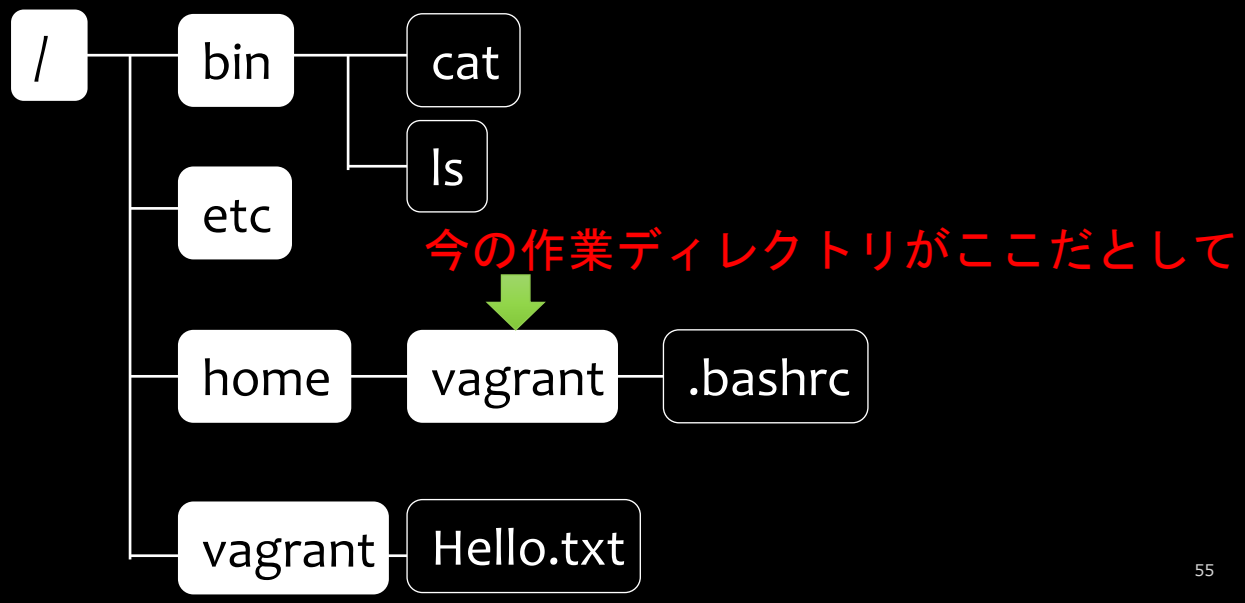


パス（2）

- /で始まる絶対パスと、現在の作業ディレクトリからの相対パスがある。
- /home/vagrantが作業ディレクトリの場合、次の2つは同じものを指す。

  /bin/cat

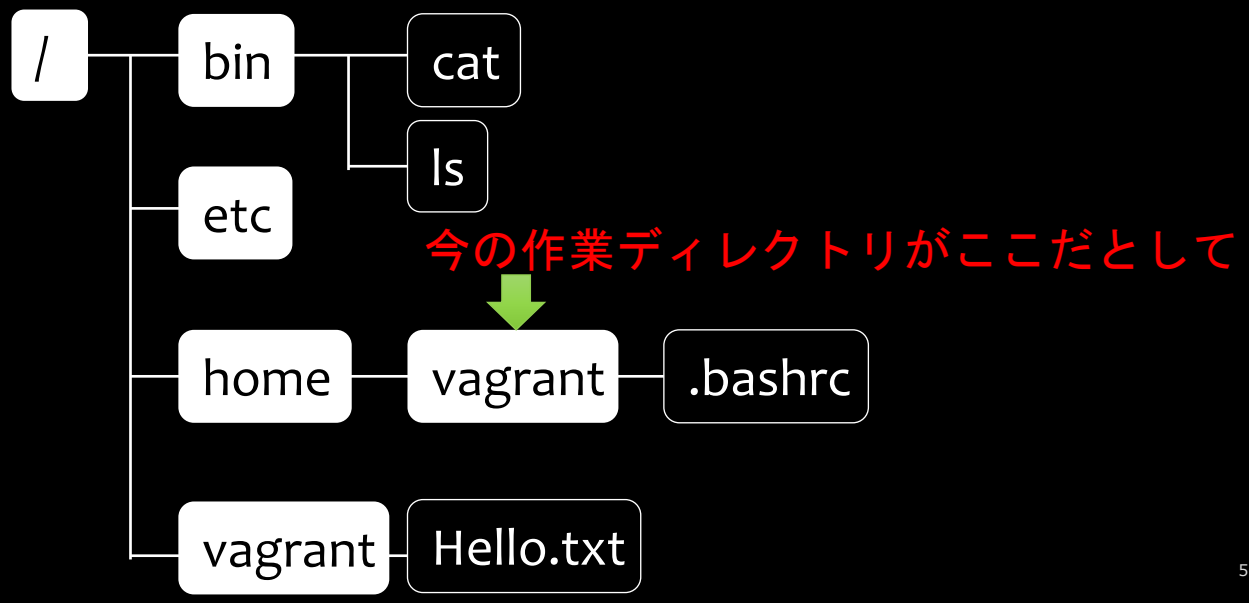
  ../../bin/cat



作業ディレクトリの変更

- コマンドcdで作業ディレクトリを変更できる。
- 現在の作業ディレクトリはコマンドpwdで表示できる。
- 作業ディレクトリは絶対パスと相対パスのどちらでも指定できる。
- /binを作業ディレクトリにするには、次のいずれでもOK。

```
cd /bin
cd ../../bin
```



ファイルの作成

■ファイルはいろいろな方法で作成できます。

1. とりあえず空のファイルを作りたい。

   例： touch /vagrant/Empty.txt

   （既にファイルが有る場合、更新日時が更新されます）

2. エディタなどで作成する。

   例： gedit /vagrant/Hello.txt

3. 既にあるファイルをコピーする。

   例： cp /vagrant/Hello.txt /vagrant/NewHello.txt

4. コマンドの出力を新たなファイルに書き込む。

   例： ls > /vagrant/Result.txt

ディレクトリの作成

■mkdirコマンドで作成できます。

1. 絶対パス指定

   mkdir /vagrant/NewDir

2. 相対パス指定

   cd /vagrant

   mkdir NewDir

ファイルの削除

■削除したファイルは復元できないので注意して下さい。

1. ファイルを削除する。

   例： rm /vagrant/Empty.txt

2. ディレクトリを削除する。

   例： rmdir /vagrant/NewDir

   （ディレクトリ内が空でない場合はエラーとなる）

3. ディレクトリを中身ごと削除する。。

   例： rm -rf /vagrant/NewDir

4. すべてのファイルを消す。

   例： rm -rf /

演習

◆関数set_common_props()の定義の記述が２ヶ所ある。どのファイルの何行目にあるか答えよ。

確認

◆理解度について

5. 今回の課題

課題１

UnicodeとUTF-8について、違いを明確にしてそれぞれ説明せよ。

- SOLAで提出して下さい。
- 提出形式：MSワード　400文字程度
- 締切：2024年 4月23日(火)　正午

課題２

次ページにあるプログラムをsort.shという名前でホームディレクトリ(～) に作成する。作成後、モードビットを変更して以下のように実行できるようにする。実行結果が表示されている画面をキャプチャする。

./sort.sh ４３９１５８２７６

- SOLAで提出して下さい。
- 提出形式：画面キャプチャファイル(Jpeg または png)
- 締切：2024年 4月23日(火) 正午

sort.sh

```
#!/bin/bash
function f() {
    sleep "$1"
    echo "$1"
}
for i in $*; do
    f "$i" &
done
wait
```

おわり

サーバシステムプログラミング及び演習

– 第3回 LAMP –

第３回の内容

- 環境構築（2）
- LAMP
- 宿題

環境構築(2)

環境構築　(2)

1. コンテナの起動
2. 日本語入力
3. SSHによる接続
4. リモートデスクトップでログインできない場合の対応
5. ファイルの作成・編集(1)
6. ファイルの作成・編集(2)

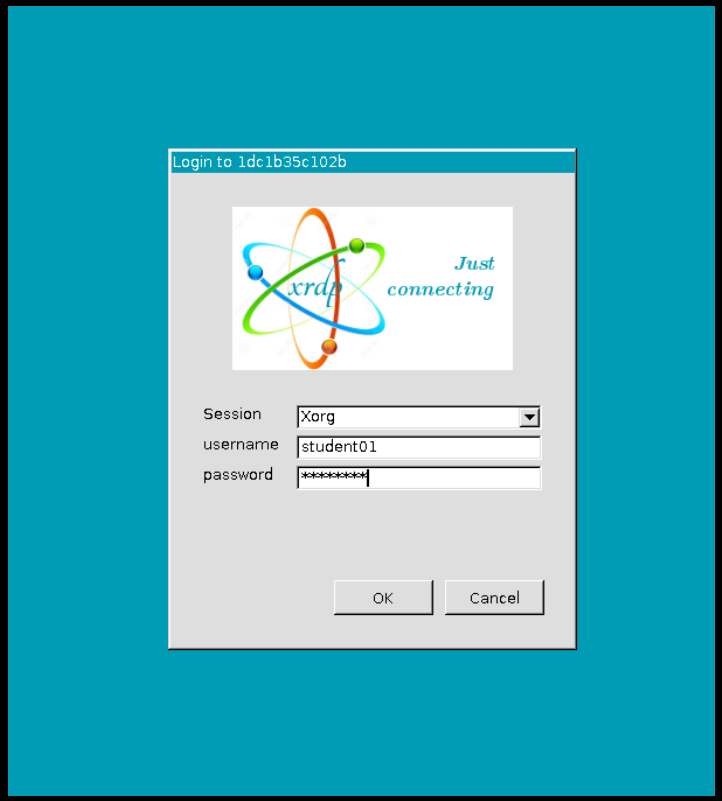
1. コンテナの起動

- コマンドラインで “docker start -i ubuntu-dt-v8.0”
- コンテナが起動されたら “./proc_init” を実行
  ⇒ すべてが “OK” であることを確認
- どれかがNGの場合、もう一度、 “./proc_init” を実行
- リモートデスクトップサーバがNGの場合、4. リモートデスクトップでログインできない場合の対応 を参照

```
PS C:¥Users¥nuno>
PS C:¥Users¥nuno> docker start -i ubuntu-dt-v8.0
root@1dc1b35c102b:/#
root@1dc1b35c102b:/# ./proc_init
 * Restarting Apache httpd web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified domain name, using 172.17.0.2. Set the 'Serv
erName' directive globally to suppress this message
                                                                                                                    [ OK ]
 * Stopping MySQL database server mysqld                                                                            [ OK ]
 * Starting MySQL database server mysqld
su: 警告: ディレクトリを /nonexistent に変更できません: そのようなファイルやディレクトリはありません
                                                                                                                    [ OK ]
 * Stopping Remote Desktop Protocol server                                                                           [ OK ]
 * Starting Remote Desktop Protocol server
xrdp-sesman[390]: (390)(140595780748864)[DEBUG] libscp initialized

xrdp-sesman[390]: (390)(140595780748864)[DEBUG] Testing if xrdp-sesman can listen on 127.0.0.1 port 3350.

xrdp-sesman[390]: (390)(140595780748864)[DEBUG] Closed socket 4 (AF_INET6 ::ffff:127.0.0.1 port 3350)

xrdp-sesman[391]: (391)(140595780748864)[INFO ] starting xrdp-sesman with pid 391

xrdp-sesman[391]: (391)(140595780748864)[INFO ] listening to port 3350 on 127.0.0.1

                                                                                                                    [ OK ]
 * Restarting OpenBSD Secure Shell server sshd                                                                      [ OK ]
 * Restarting periodic command scheduler cron
 * Stopping periodic command scheduler cron                                                                         [ OK ]
 * Starting periodic command scheduler cron                                                                         [ OK ]
root@1dc1b35c102b:/#
root@1dc1b35c102b:/#
```

2. 日本語入力

- リモートデスクトップでログイン
- Username　⇒　student01 （※パスワードも同じ）



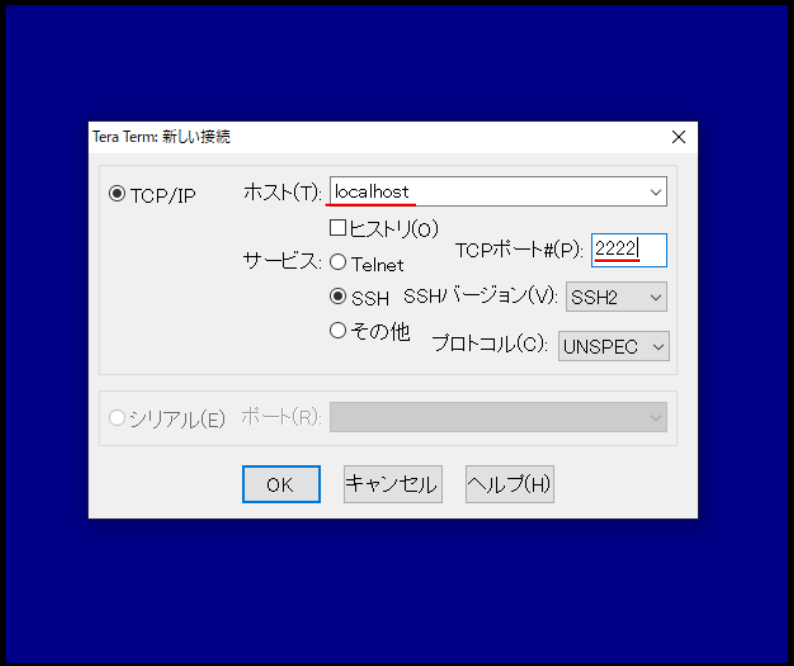
2. 日本語入力

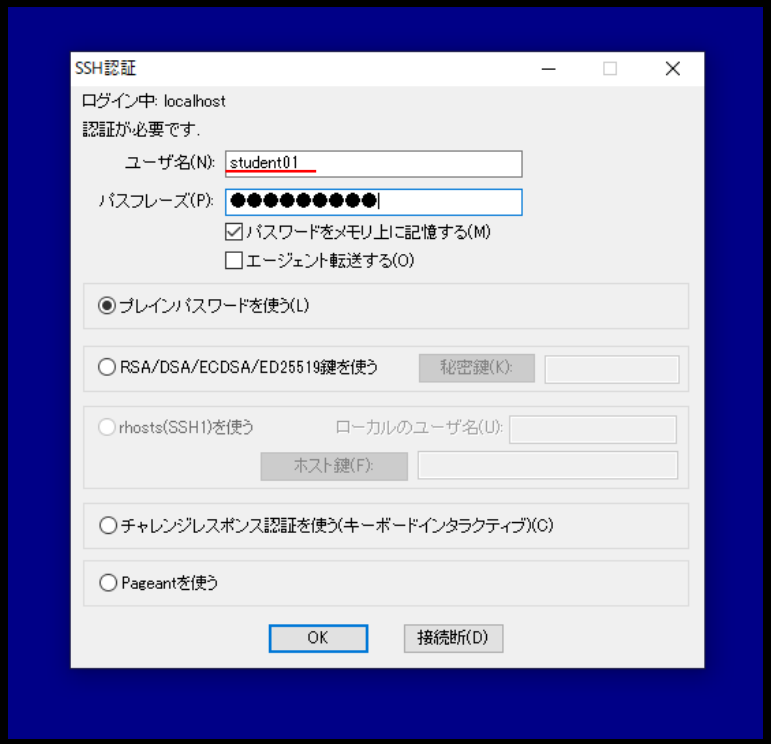
- リモートデスクトップでログイン
- Username　⇒　student01 （※パスワードも同じ）

※一度選択するとあとは　<ctrl> + space で日本語入力と英語入力を切替られます。
※“入力モード(あ)”が選択できない場合は一旦”Japanese – Mozc”を選択してみて下さい。

3. SSHによる接続

- ターミナルソフトで “localhost:2222” に接続
- Username ⇒ student01 （※パスワードも同じ）





4. リモートデスクトップでログインできない場合の対応

リモートデスクトップで接続できなくなった場合

① SSHで接続する。

② xrdpおよびxrdp-sesmanのプロセスを調べて、止める

```
root@8c5b2163a362:~#
root@8c5b2163a362:~# ps aux | grep xrdp
root       408  0.0  0.0   9572  4300 ?        S    19:52   0:00 /usr/sbin/xrdp-sesman
xrdp       411  0.0  0.0   7488  2244 ?        S    19:52   0:00 /usr/sbin/xrdp
root      1652  0.0  0.0   3304   728 pts/1    S+   20:34   0:00 grep --color=auto xrdp
root@8c5b2163a362:~#
root@8c5b2163a362:~#
root@8c5b2163a362:~# kill -9 408 411
root@8c5b2163a362:~#
root@8c5b2163a362:~# ps aux | grep xrdp
root      1654  0.0  0.0   3304   676 pts/1    S+   20:35   0:00 grep --color=auto xrdp
root@8c5b2163a362:~#
root@8c5b2163a362:~#
```

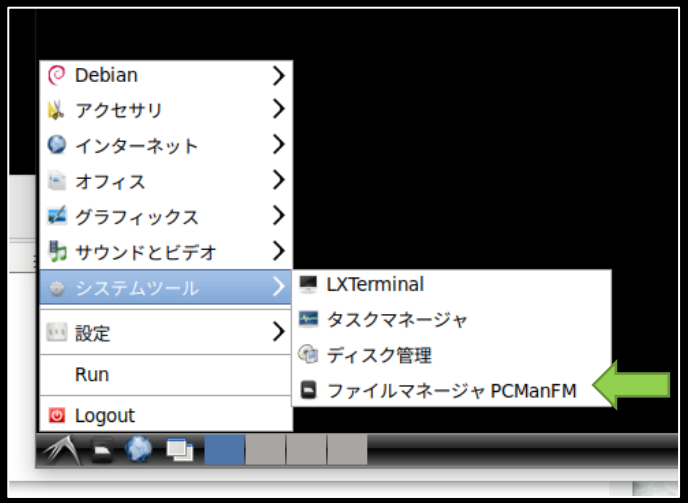
プロセスがないことを確認する

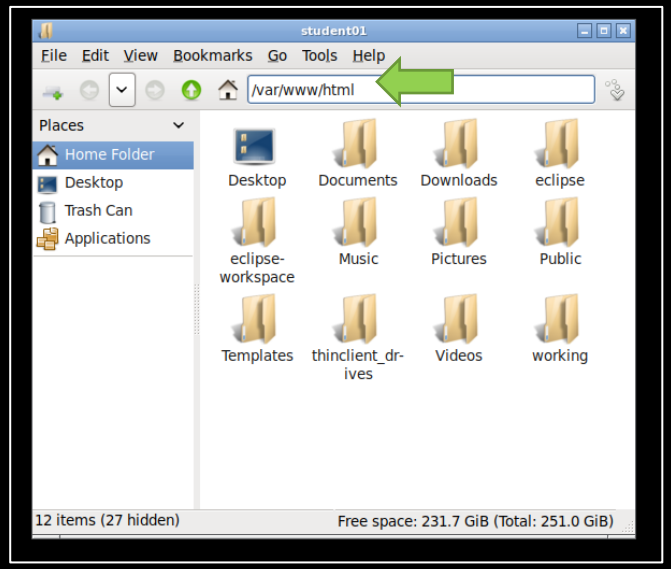
③ プロセスIDファイルを削除してから、xrdpを起動する。

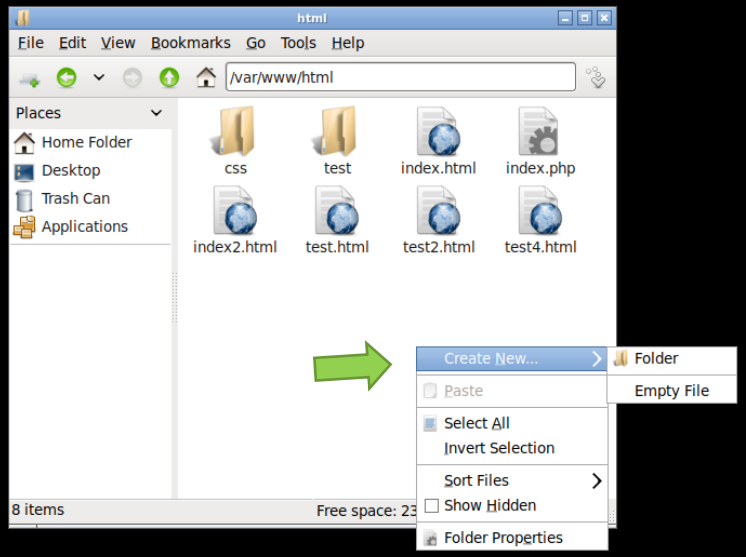
```
root@8c5b2163a362:~#
root@8c5b2163a362:~# rm /var/run/xrdp/xrdp-sesman.pid
root@8c5b2163a362:~#
root@8c5b2163a362:~# /etc/init.d/xrdp start
 * Starting Remote Desktop Protocol server                                [ OK ]
root@8c5b2163a362:~#
root@8c5b2163a362:~#
```

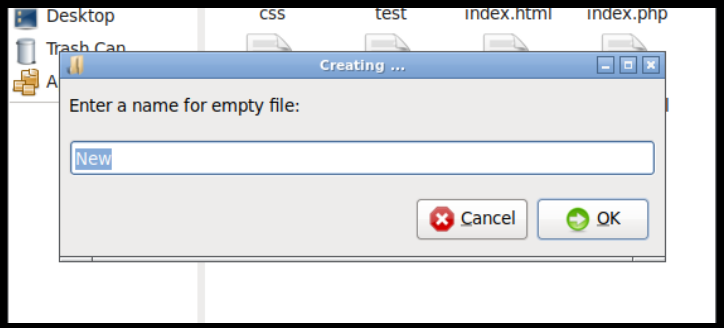
5. ファイルの作成・編集(1)

- ファイルマネージャーで作成
- Text editor で編集



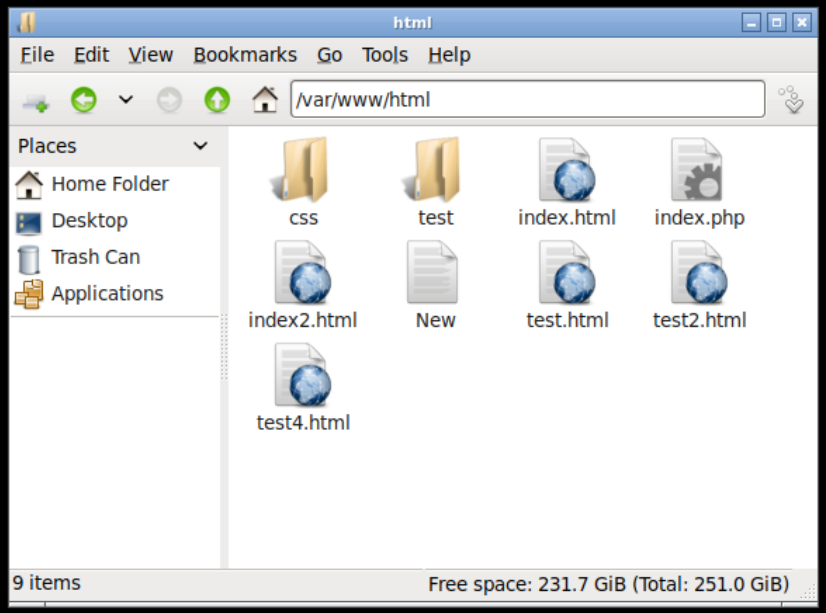


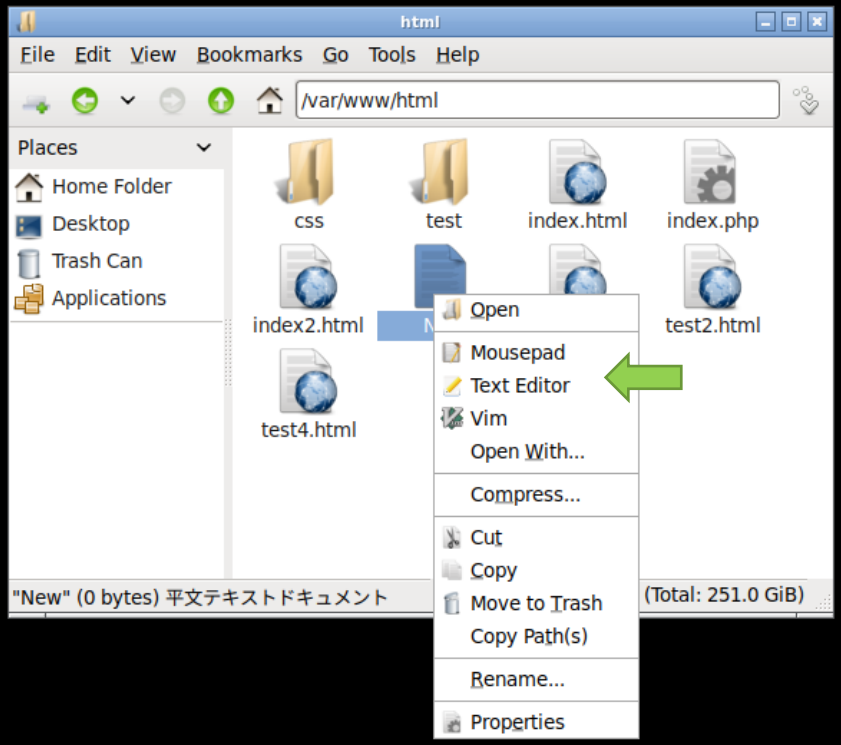




5. ファイルの作成・編集(1)

- ファイルマネージャーで作成
- Text editor で編集





6. ファイルの作成・編集(2)

- ターミナルソフトで接続
- エディタで編集

localhost - student01@1dc1b35c102b: /var/www/html VT

ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
student01@1dc1b35c102b:/var/www/html$ cd
student01@1dc1b35c102b:~$
student01@1dc1b35c102b:~$
student01@1dc1b35c102b:~$
student01@1dc1b35c102b:~$
student01@1dc1b35c102b:~$ cd /var/www/html/
student01@1dc1b35c102b:/var/www/html$ ls
New  css  index.html  index.php  index2.html  test  test.html  test2.html  test4.html
student01@1dc1b35c102b:/var/www/html$ vi test4.html
```

localhost - student01@1dc1b35c102b: /var/www/html VT

ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

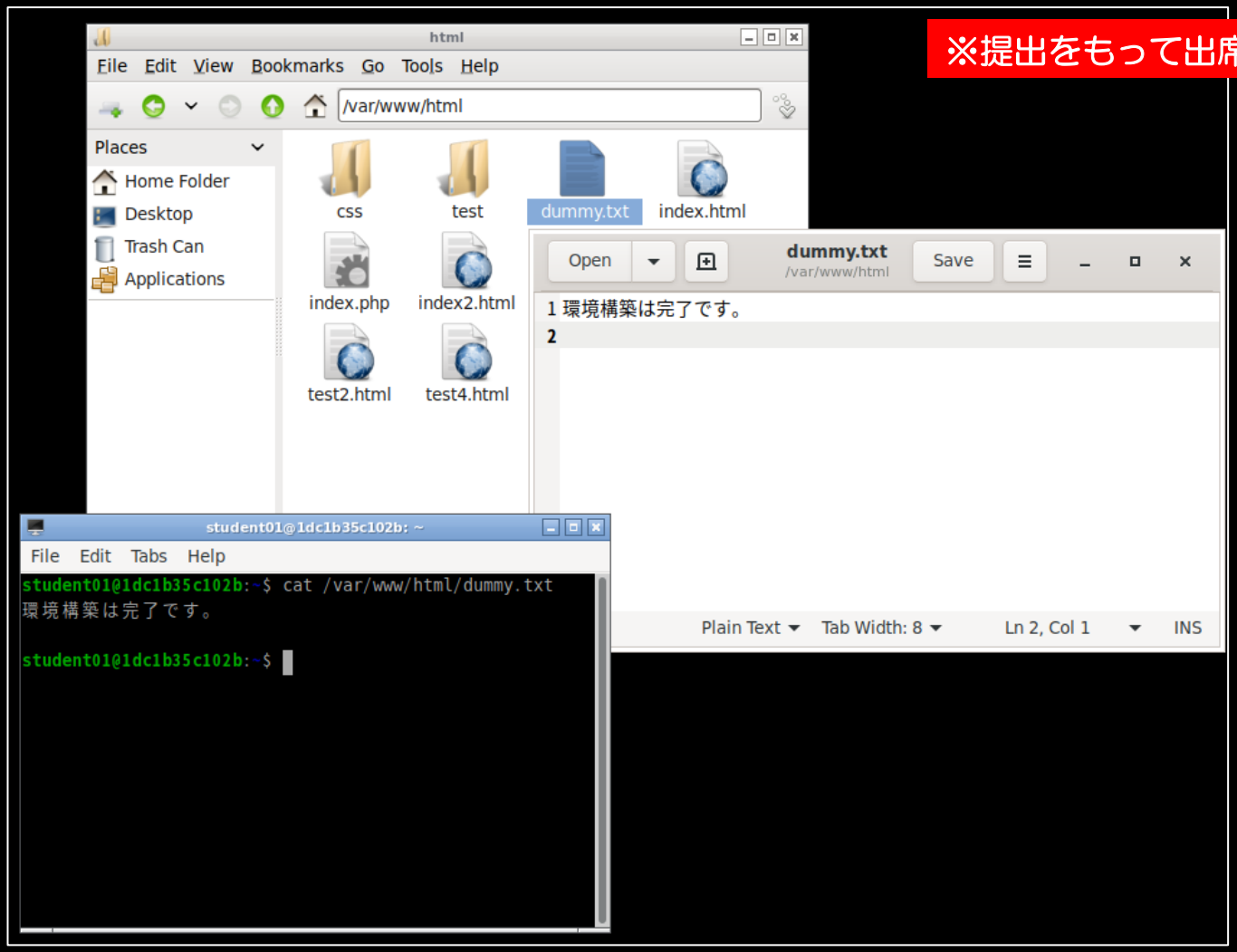
```
<!DOCTYPE>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<h1>
HTMLのテスト(4)です。
</h1>
うまく表示できくるかな？
</body>
</html>
~
```

演習03-01

実

- /var/www/html に、ファイル”dummy.txt”を作成する。
- ファイルに “環境構築は完了です。” と記述し、cat で表示する。
- 表示画面をキャプチャし、キャプチャファイルをSOLAに提出する。

※提出をもって出席とします。



LAMP

LAMP（1）LAMP

1. Webサーバ構築で使われる、定番のオープンソース製品群
   - Linux （OS）
   - Apache （Webサーバ）
   - MySQL （データベース）
   - Perl, PHP, Python （スクリプト言語）

LAMP（2）Linux

1. UNIXのソースコードを一切使わずに開発されたオープンソースのUNIX互換OS。UNIX関連のPOSIX規格を満たす。
2. ハード（PC、周辺機器）対応が良いので広く使われている。サーバ分野でのOSシェア80%。
3. ライバル
   - Windows Server
   - *BSD (FreeBSD, NetBSD, ...)
4. なお、現在UNIXを名乗れるのは、
   a. Apple macOS
   b. HP HP-UX
   c. IBM AIX
   d. Oracle Solaris

   など

LAMP（3）Apache

1. 正式名はApache HTTP Server
2. オープンソースのWebサーバソフトウェア
3. Webサーバのシェア33%弱
4. 多くのOSをサポートしている。
5. ライバル
   - nginx（シェア1位）

順位	Webサーバー	世界シェア	昨年比
1	Nginx	33.9%	+0.9%
2	Apache	32.8%	+1.5%
3	Cloudflare	20.5%	-1.1%
4	LiteSpeed	11.8%	-0.1%
5	Microsoft-IIS	5.8%	-0.4%

世界のWebサーバー人気シェアランキング（2023年1月）

LAMP（4）MySQL

1. オープンソースのデータベースシステム。
2. Webサーバのシェアトップ。利用企業数82.1%。
3. 多くのOSをサポートしている。
4. Oracleが商用版も出している。
5. ライバル
   - PostgreSQL （オープンソース）
   - Firebird （オープンソース）
   - Oracle Database （商用）

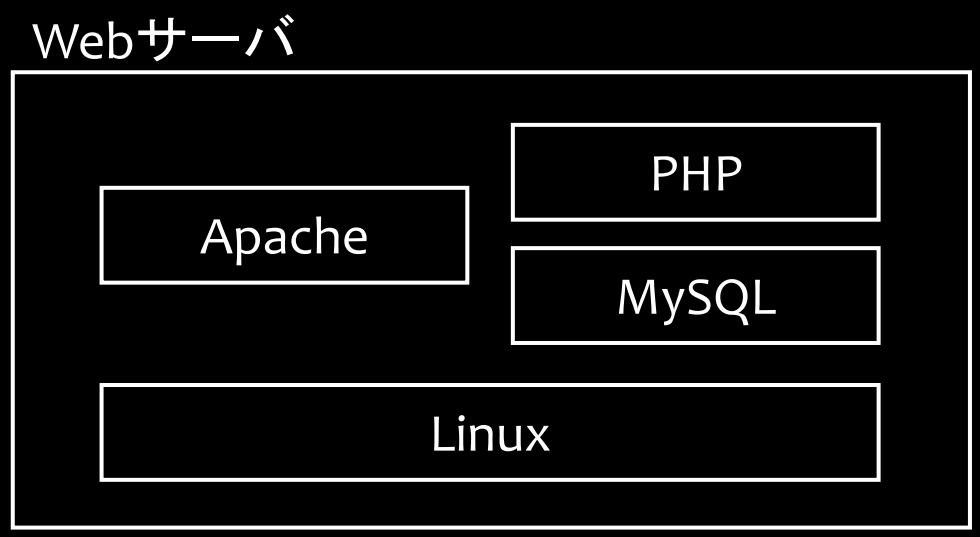
LAMP（5）Perl, PHP, Python

1. オープンソースのスクリプト言語。
2. Webページ（HTML等）をダイナミックに生成するために使われる。
3. Java等の強い型システムを持った言語と違い、ラフな記述が許される。
4. 文字列処理やデータベース処理に強い。

python™

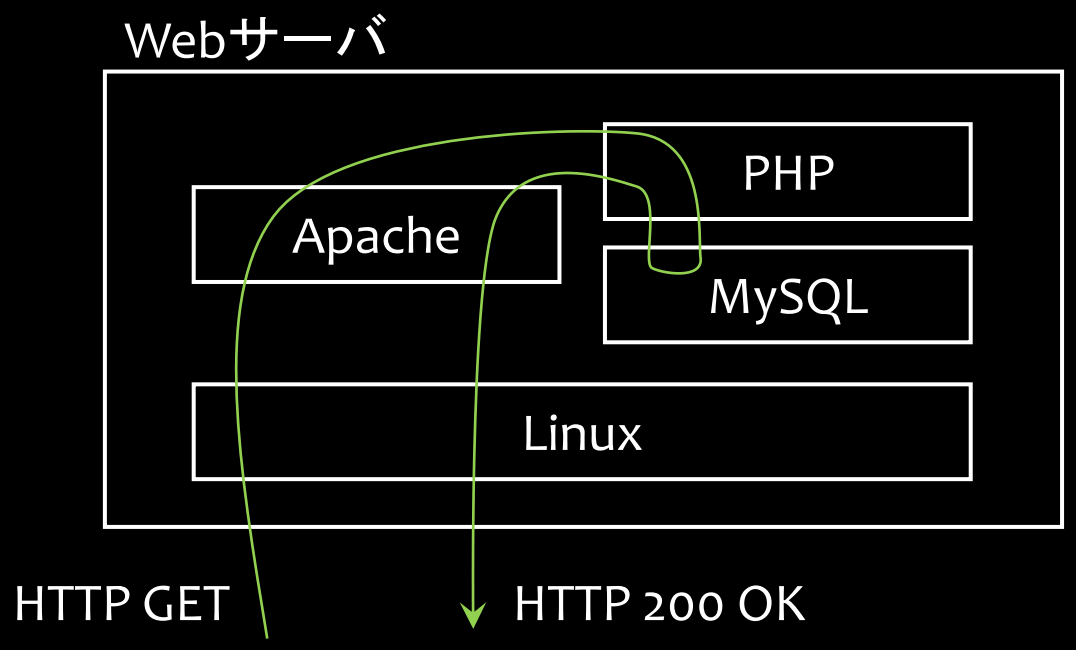
LAMP（６）各部の役割

Ｌ： OS、IP通信
Ａ：Webサーバ
Ｍ：データベース
Ｐ： ダイナミックなWeb情報（HTML）の生成



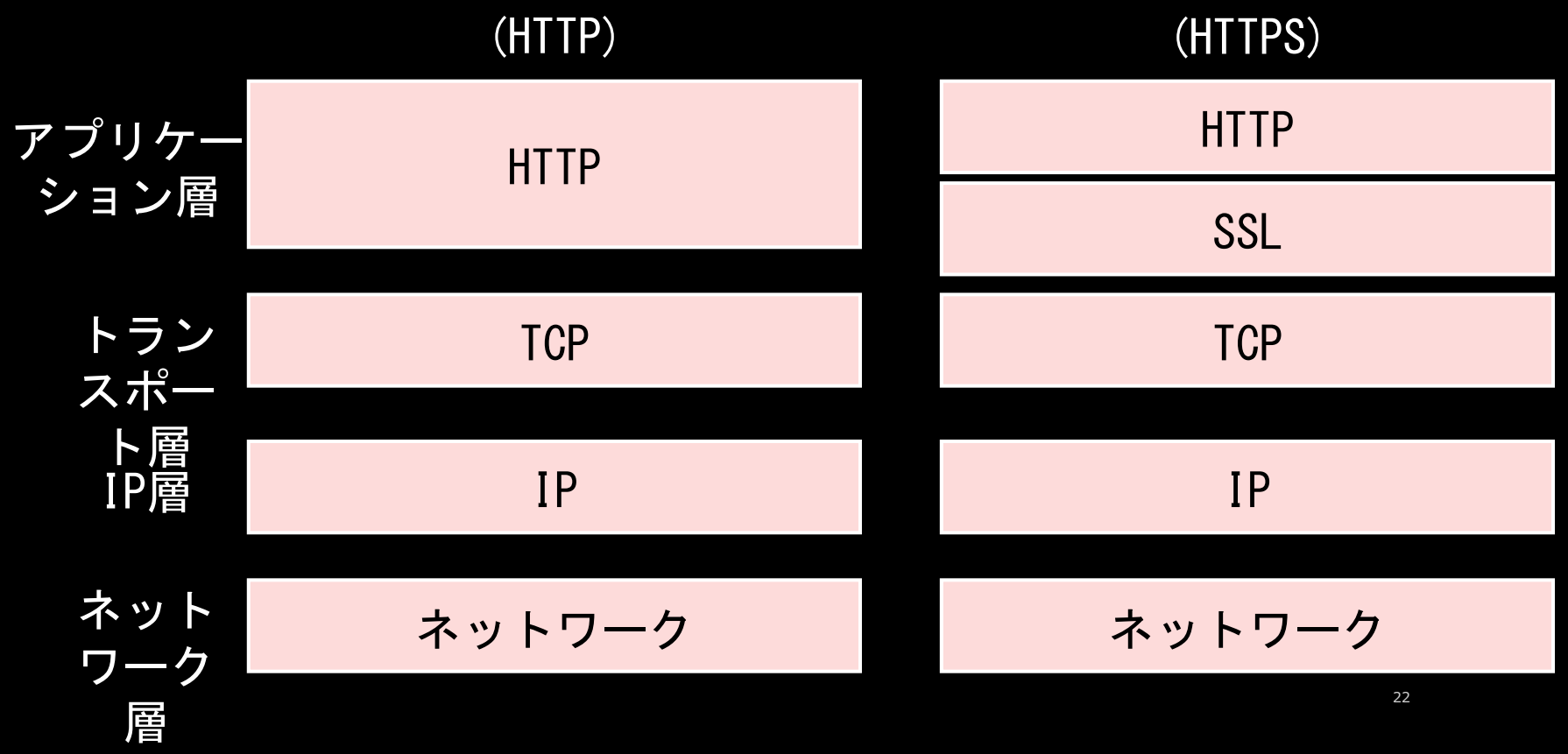
LAMP（７）全体の動作

1. HTTPのリクエストをLinuxがApacheに渡す
2. Apacheは必要に応じてスクリプトを起動
3. スクリプトはデータベースを読み込んだりして動的ページを生成
4. Apache経由で動的ページを返却



LAMP（８）HTTPの階層

1. HTTPはTCP上のプロトコル。
2. HTTPSはHTTPとTCPの間にセキュア通信のSSLが入る。



LAMP（９）IPパケット処理

1. LinuxのIPドライバ: IPパケットの中を見るとTCPパケットであることが分かるので、TCPドライバに渡す。
2. LinuxのTCPドライバ: TCPパケットの中を見ると80番ポート宛であることが分かるので、80番ポートを待ち受けているApacheに中身を渡す。
3. Apache: 受け取ったデータはHTTPであるので解釈して実行。

LAMP（10）LAMP環境

1. Linuxディストリビューション
   a. Linuxカーネル（OS本体）
   b. ライブラリ
   c. アプリ

   などをひとまとめにしたもの。
2. LinuxディストリビューションにはAMPも含まれることが多い。
3. 配布したdockerのイメージではLAMPを動く状態にしてあります。

実

コマンドの実行（1）Linux

1. 現在動いているカーネルのバージョン表示

   ① uname -a

2. メモリ使用状況

   ② free

3. ディスク使用状況

   ③ df

4. プロセス状況

   ④ ps

```
student01@1dc1b35c102b:~$ uname -a
Linux 1dc1b35c102b 5.4.72-microsoft-standard-WSL2 #1 SMP Wed Oct 28 23:40:43 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
student01@1dc1b35c102b:~$ free
              total        used        free      shared  buff/cache   available
Mem:       26180552     1024948    17262936      412504     7892668    24372144
Swap:       7340032           0     7340032
student01@1dc1b35c102b:~$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
overlay        263174212 6795688 242940368   3% /
tmpfs              65536       0     65536   0% /dev
tmpfs           13090276       0  13090276   0% /sys/fs/cgroup
shm                65536       0     65536   0% /dev/shm
/dev/sdc       263174212 6795688 242940368   3% /etc/hosts
tmpfs           13090276       0  13090276   0% /proc/acpi
tmpfs           13090276       0  13090276   0% /sys/firmware
student01@1dc1b35c102b:~$ ps
  PID TTY          TIME CMD
  478 pts/1    00:00:00 bash
  486 pts/1    00:00:00 ps
student01@1dc1b35c102b:~$
```

実

コマンドの実行（2）HTTP/HTTPS

1. ネットワークインタフェース状態

   ① ip address (iproute2, iputls-ping)

2. HTTP/HTTPSのポート番号表示

   ② grep http /etc/services

3. 80番ポートを待ち受けているプロセスを表示

   ③ sudo lsof –i:80

実

コマンドの実行（3）Apache

1. Apacheのバージョン表示
   ① apache2 -v
2. Apacheの実行ファイルの位置
   ② which apache2
3. 設定ファイル
   ③ more /etc/apache2/apache2.conf
4. プロセス状況
   ④ ps aux | grep apache2
5. モジュール状態
   ⑤ apache2 -l

実

コマンドの実行（4）MySQL/PHP

1. MySQLのバージョン表示
   ① mysqld --version
2. MySQLの実行ファイルの位置
   ② which mysqld
3. MySQLのプロセス状態
   ③ ps aux | grep mysqld
4. PHPのバージョン表示
   ③ php -v
5. PHPの実行ファイルの位置
   ④ which php

実

演習03-02

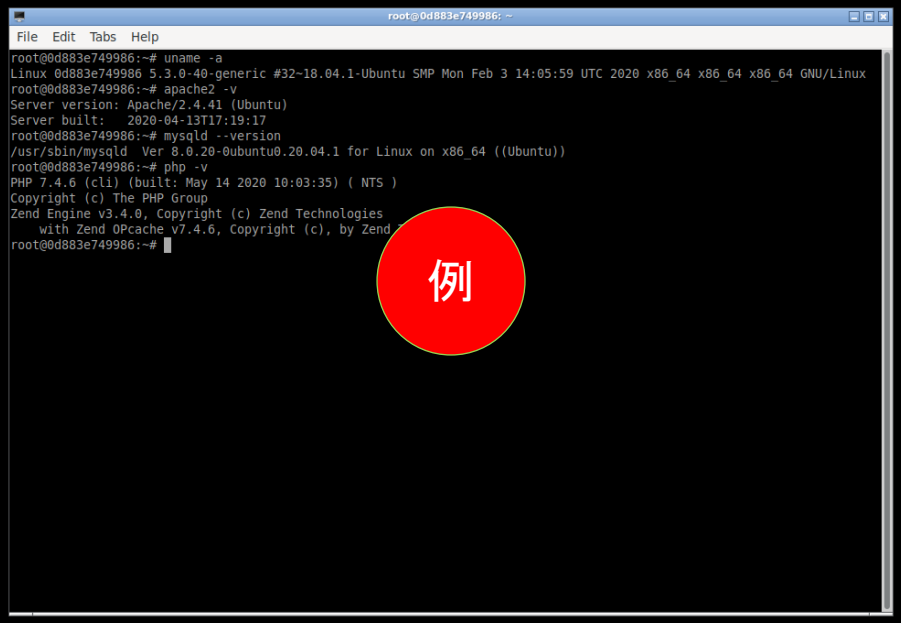
- 下記のコマンドを実行し、実行結果画面をキャプチャする。
- キャプチャファイルをSOLAに提出

uname -a

apache2 -v

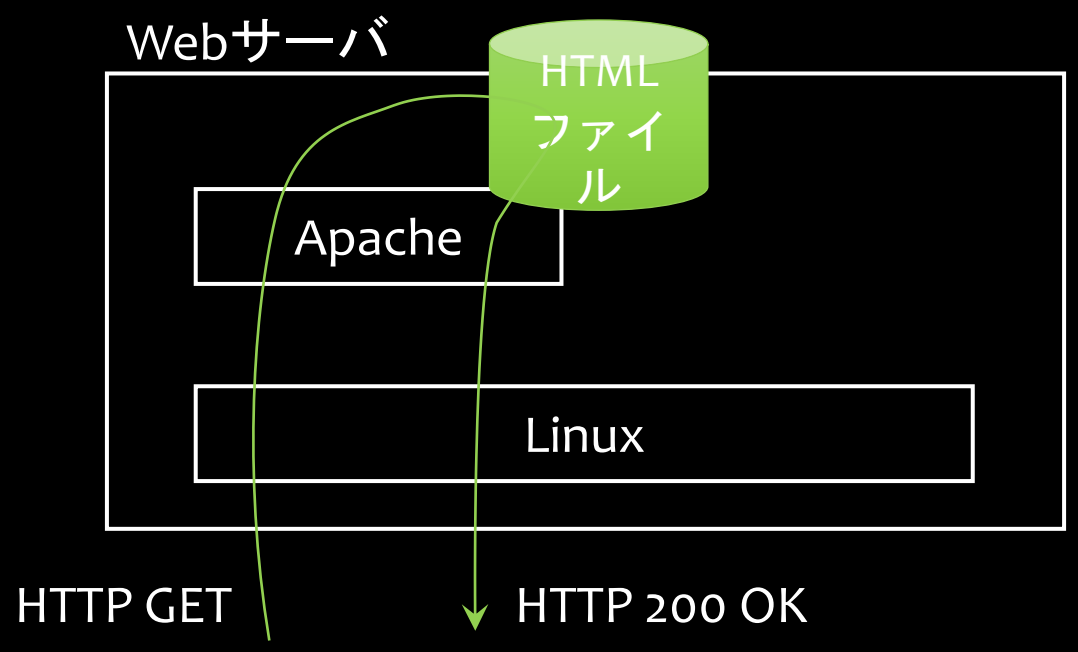
mysqld --version

php -v



Apache（１）静的Webページ

1. HTTPのリクエストをLinuxがApacheに渡す。
2. Apacheはディスク上からHTMLファイルを読みだして返却。
3. HTMLファイルがあれば200 OK、無ければ404 Not Foundが返る。



Apache（２）HTMLファイルの置き場

1. HTMLファイルはApacheの設定（DocumentRoot）にてディスク上の置き場（ディレクトリ）が決められている。

```
#more /etc/apache2/sites-enabled/000-default.conf
```

2. 配布したimageでは、HTMLファイルのディレクトリは、/var/www/html と設定してある。

```
#grep DocumentRoot /etc/apache2/sites-enabled/000-default.conf
```

Apache動作確認（1）

Apache動作

Linux上でApacheはTCPの80番ポートを受け付けている。Linux上のブラウザから80番ポートにアクセスして動作を確認する。

① ブラウザを起動し、http://localhost/ にアクセス

② Apache2 Ubuntu Default Page が表示されれば成功

Apache動作確認（2）

Apache動作

Linux上の80番ポートは、DockerによってWindowsの8080番ポートにマッピングされている。Windows上の8080番ポートにアクセスするとLinuxの80番ポートにアクセスでき、結果的にApacheにアクセスできる。

① Windows上のブラウザから、http://localhost:8080/ にアクセス

② Apache2 Ubuntu Default Page が表示されれば成功

ポート番号の対応表

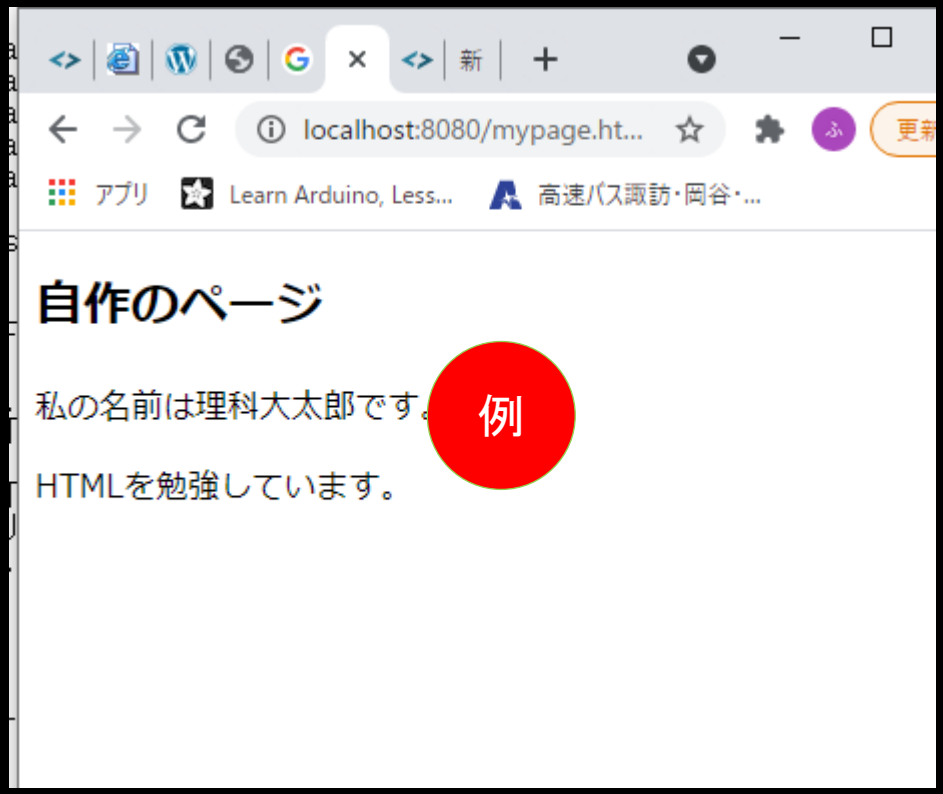
アプリケーション	Linux	Windows
Apache	80	8080

実

演習03-03

- ディレクトリ /var/www/html にファイル”mypage.html”を作成し、ブラウザで表示する。
- mypage.html の内容は下記の通りとする。
- キャプチャファイルをSOLAに提出

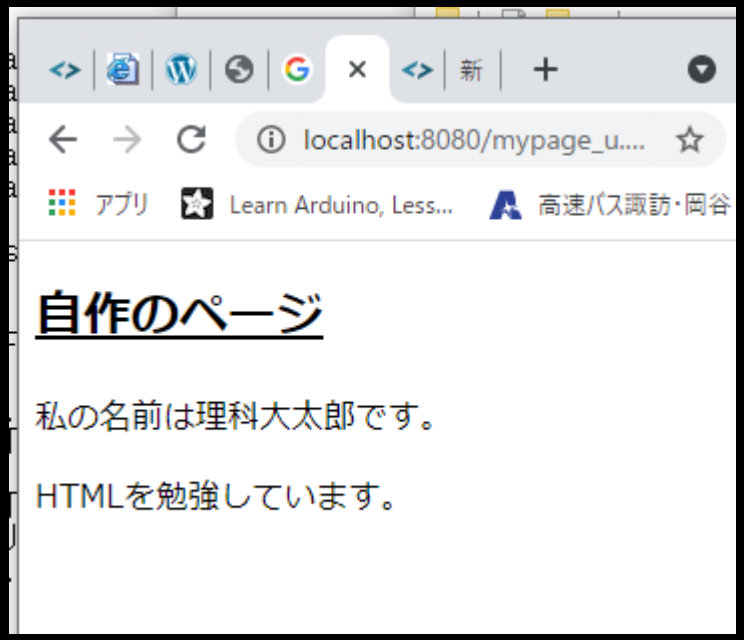
```
1 <!DOCTYPE html>
2
3 <html lang="ja">
4 <head>
5   <meta charset="utf-8">
6   <title>自作のページ</title>
7   <meta name="description" content="自作のページ">
8   <meta name="author" content="理科大太郎">
9 </head>
10
11 <body>
12
13 <h2>自作のページ</h2>
14
15 <p>私の名前は理科大太郎です。</p>
16
17 <p>HTMLを勉強しています。</p>
18
19 </body>
20 </html>
21
```



実

演習03-04

- ファイル “mypage.html” をファイル” mypage_u.html” としてコピーを作成する。
- ファイル” mypage_u.html” の内容を変更し、” 自作のページ” の部分にアンダーラインを表示する。
- アンダーラインの表示はHTMLタグ <u> を使用する。
- ブラウザの表示画面をキャプチャし、キャプチャファイルをSOLAに提出



スタイルシートの使用

1. HTMLファイルにスタイルシートを指定することで、スタイルシートを使用することが可能

```
4 <head>
5   <meta charset="utf-8">
6   <title>自作のページ</title>
7   <meta name="description" content="自作のページ">
8   <meta name="author" content="理科大太郎">
9   <link rel="stylesheet" href="/css/mystyles.css">
10 </head>
11
```

スタイルシートの作成

1. /var/www/htmlにディレクトリcss　を作成
2. /var/www/html/css にスタイルファイルmystyles.css を作成
3. mystyles.cssは下記の通りとする。

Open　*mystyl…　/var/www/h…　Save

```
1 @charset "utf-8";
2
3 /* type selectors */
4 h2 {
5         text-decoration : underline;
6 }
7
```

mystlyes.css

実

演習03-05

- ファイル “mypage.html” をファイル” mypage_s.html” としてコピーを作成する。
- ファイル” mypage_s.html” に作成したスタイルシートを指定する。
- ブラウザの表示画面をキャプチャし、キャプチャファイルをSOLAに提出

実

画像のダウンロード

1. 背景画像を置くディレクトリを作成。
   ① mkdir -p /var/www/html/img

2. Linux上でSOLAにアクセスし、background.pngをダウンロードし、/var/www/html/imgに保存する。

SOLA https://sus.mrooms.net/course/

CSS：背景イメージの設定

1. スタイルシートの中でタイプセレクタのbodyにbackground-imageを設定することでHPの背景イメージを設定することができる。

```
/* type selectors */

body {
        background-image:url(/imgs/background.png);
}

```

CSS：classセレクタ、idセレクタによるフォーマットの設定

1. スタイルシートにセレクタを設定
2. HTMLファイルでid、classを設定

```
6
7 h2 {
8         color : #ffc0cb;
9         text-decoration : underline;
10 }
11
12 p {
13        color : white;
14 }
15
16 /* class selectors */
17 .simple-text {
18        background-color : red;
19        color : #fff;        /* 白 */
20 }
21
22 /* id selectors */
23 #green-text {
24        background-color : green;
25        color : #fff;        /* 白 */
26        font-size : 40px;
27 }
28
```

CSS

```
12 <body>
13
14 <h2>自作のページ</h2>
15
16 <p class="simple-text">私の名前は理科大太郎です。</p>
17
18 <p id="green-text">HTMLを勉強しています。</p>
19
20 </body>
```

html

実

演習03-06

- ファイル”mystyles.css”を変更する。変更内容は下記の通りとする。
- ファイル”mypage_s.html”を変更する。変更内容は下記の通りとする。
- に作成したスタイルシートを指定する。
- ブラウザでmypage_s.htmlを表示し、表示画面をキャプチャし、キャプチャファイルをSOLAに提出

◆ mystlyes.css 変更内容

- HPの背景を background.png とする。
- H2 のフォントの色を #ffc0cb 、文字に下線を引く。
- classセレクタ”simple-text”を設定する。背景色：赤、文字色：白
- Idセレクタ “green-text”を設定する。背景色：赤、文字色：白、フォントサイズ：40px

◆ mypage_s.html 変更内容

- 文章”私の名前は理科大太郎です。にclassセレクタ”simple-text”を適用
- 文章” HTMLを勉強しています。”に idセレクタ “green-text”を適用

宿題03-01

インターネット上から著作権フリーな背景画像をダウンロードし、/var/www/html/img/に置く。

/var/www/html/css/mystyles.css を書き換え、自分が選んだ背景画像が表示されるようにする。

ブラウザでHTMLファイルへアクセスし、一番気に入った表示の画面キャプチャを取る。

◆形式

JPEGまたはPNG形式。

◆提出方法

SOLAでキャプチャファイルを提出。

◆締切

【締切】2024年4月30日(火)正午

宿題03-02

HTMLとは何であるか。以下の点を網羅して説明せよ。

- 開発した組織、代表者、時期
- 目的
- 特徴
- 最新状況

◆形式

テキスト400文字以内。

◆提出方法

SOLAに提出。

◆締切

【締切】2024年4月30日(火)正午

サーバシステムプログラミング及び演習

– 第4回 JavaScript –

本日の内容

1. JavaScript
2. 演習
3. 宿題

JavaScript

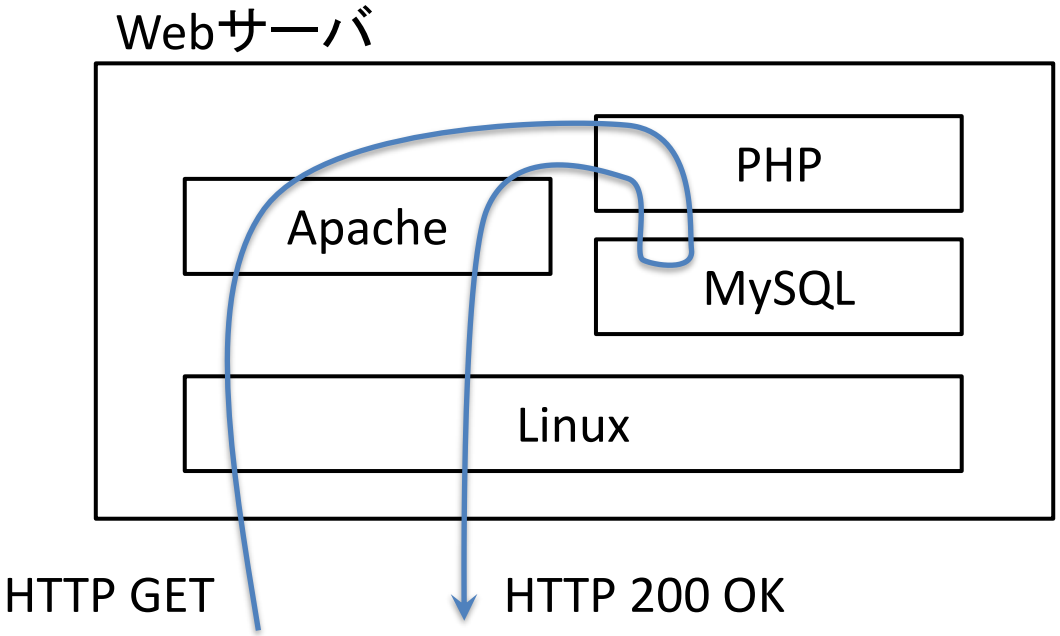
LAMP役割

L: OS、IP通信

A: Webサーバ

M: データベース

P: ダイナミックなWeb情報（HTML）の生成



Webサーバの動作

1. URLでのリクエストは、受信したWebサーバで解釈され、HTMLファイル（*.html）が送り返されたり、スクリプト（*.phpなど）を実行した結果が送り返される。

2. 既に出来上がっているHTMLファイルを送り返す方式を、静的Webページという。

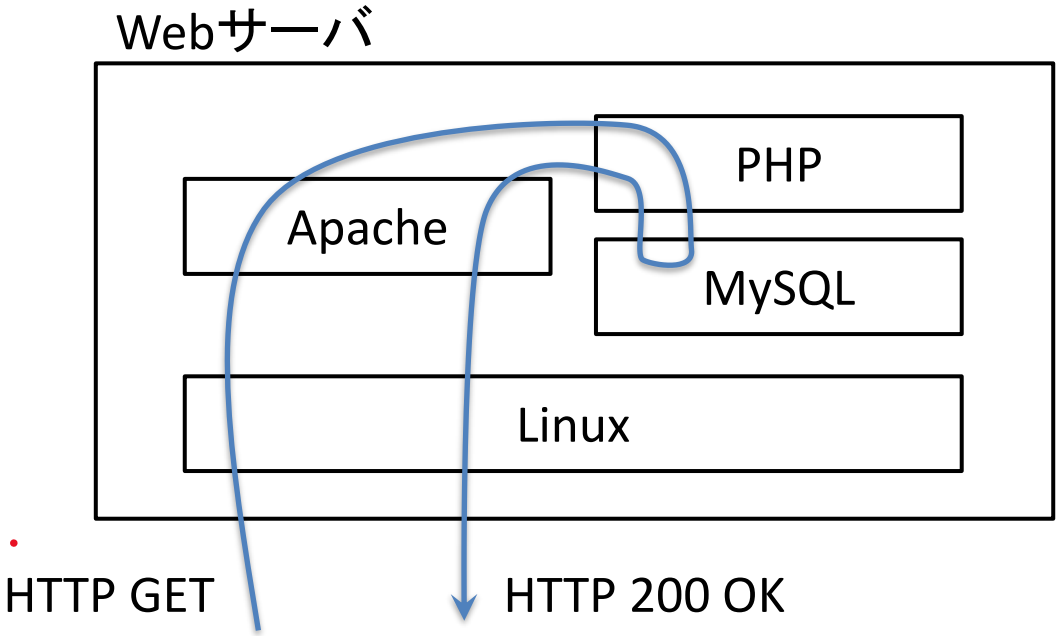
3. リクエストの都度、ダイナミックにHTMLファイルを作り出す方式を、動的Webページという。

Webサーバ：動的Webページ

1. ApacheはURLによって動的Webページかどうかを判断し、動的Webページであれば、PHPのスクリプトなどを起動する。

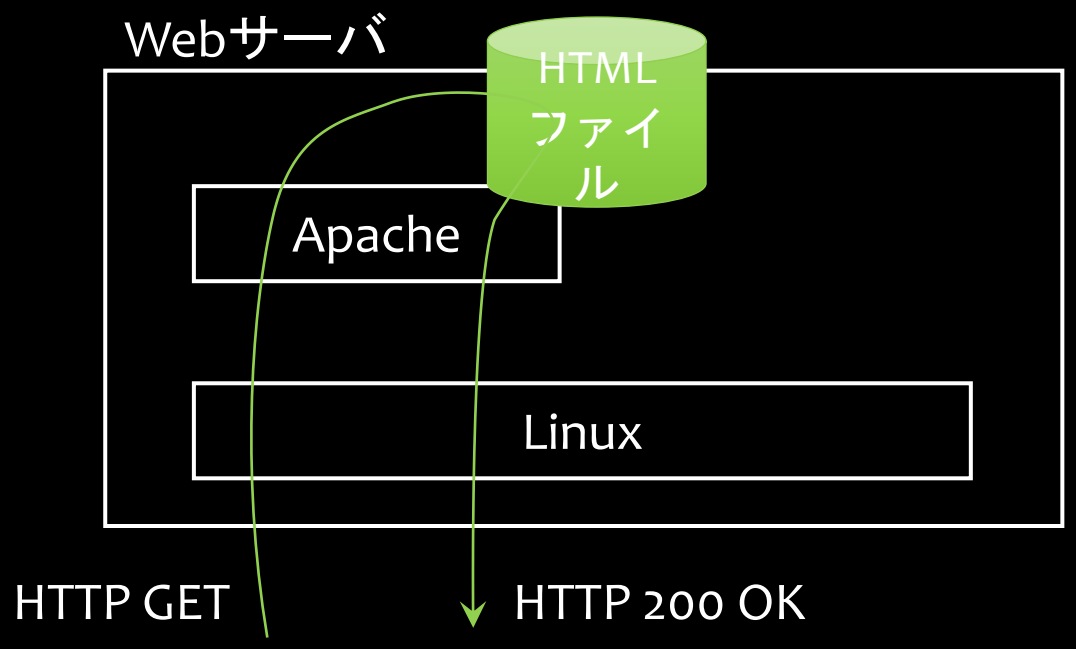
2. PHPスクリプトは必要に応じてデータベースにアクセスして情報を取得（必要なければアクセスしない）。

3. PHPスクリプトは、動的にHTMLファイルを組み立てて、Apache経由で返送する。



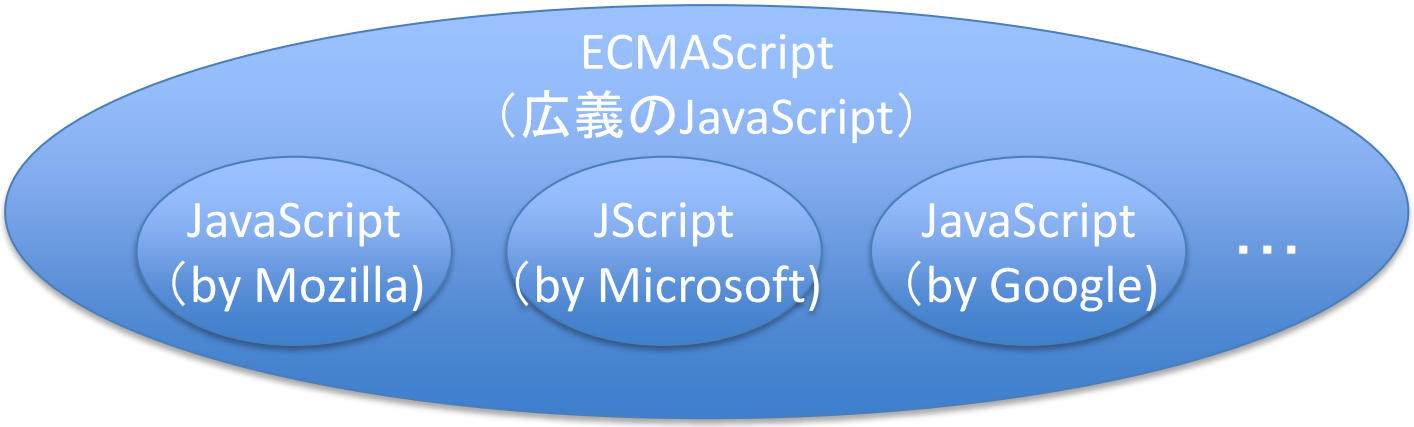
Webサーバ：静的Webページ

1. HTTPのリクエストをLinuxがApacheに渡す。
2. Apacheはディスク上からHTMLファイルを読みだして返却。
3. HTMLファイルがあれば200 OK、無ければ404 Not Foundが返る。



JavaScript仕様

- Mozilla Foundationが仕様を決めた言語。
- 標準化されたものはECMAScriptと呼ばれる。
- ECMAScript準拠でも別の呼称を付けられる。MicrosoftのJScriptなど。
- 広義ではJavaScriptはECMAScript仕様の言語全体を指す。



JavaScript概要

- オブジェクト指向のスクリプト言語。
- 名前にJavaが付いているが、Java言語とは無関係。
- Webブラウザ上で動作する言語として開発された。対応ブラウザには、JavaScriptエンジンという言語処理機能が搭載されている。

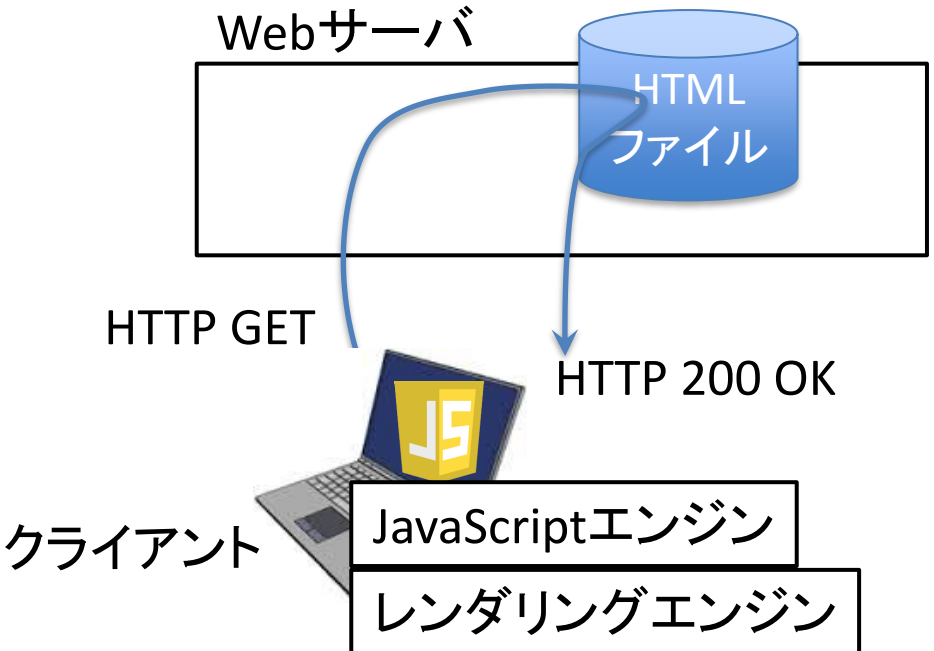
ブラウザ	レンダリングエンジン	JavaScriptエンジン
Chrome	Blink	V8
Edge	EdgeHTML	Chakra
Firefox	Gecko	SpiderMonkey
IE	Trident	Chakra
Opera	Blink	V8
Safari	WebKit	Nitro

JavaScriptの動作

1. HTMLファイルに載せられてクライアント（Webブラウザ）にJavaScriptのスクリプトが届く。

2. クライアントはJavaScriptエンジンによりスクリプトを解釈実行する。

3. CSSやHTMLの解釈や画面表示はレンダリングエンジンが行う。



JavaScriptでできること

a. マウス操作などのイベントに応じて、アニメーションなどの画面効果を出す
   - CSSが発展してきたため、従来はJavaScriptで行っていたことも簡単なものであればCSSでできるようになってきました。まずはCSSでの実現を考えるべきです。

b. フォーム入力値の値をチェックしたり、その場で計算したりする

c. ブラウザ種別に応じて処理を変える

d. クライアント（Webブラウザ）側が主導で処理を行う
   - Webシステム（HTTPプロトコル）は、クライアントからのリクエストにサーバが応答したら終わり。定期的に表示情報をリフレッシュするようなことは、クライアント側でプログラムを動かさないとできません。

言語ランキング

Apr 2024	Apr 2023	Change	Programming Language	Ratings	Change
1	**1**		**Python**	**16.41%**	**+1.90%**
2	2		C	10.21%	-4.20%
3	4	^	C++	9.76%	-3.20%
4	3	v	Java	8.94%	-4.29%
5	5		C#	6.77%	-1.44%
6	7	^	JavaScript	2.89%	+0.79%
7	10	^	Go	1.85%	+0.57%
8	6	v	Visual Basic	1.70%	-2.70%
9	8	v	SQL	1.61%	-0.06%
10	20	^^	Fortran	1.47%	+0.88%

TIOBE indexより < http://www.tiobe.com/tiobe-index />

その他の使い方

- JavaScriptは、Webブラウザ以外で動作させることもできる。この場合、通常のプログラミング言語のように使える。
- 実行させるためにはブラウザ外で動くJavaScriptエンジンを使用する。よく使われているのはNode.jsなど。
- サーバプログラムや、クライアントプログラム（Webブラウザが無くても動作する）が作れる。

JavaScriptの使い方

- この授業ではWebブラウザ上でJavaScriptを動かすための必要最小限のことを勉強します。

  1. HTMLファイルでJavaScriptプログラムをどう呼び出すか。
  2. JavaScriptプログラムをどう書くか。

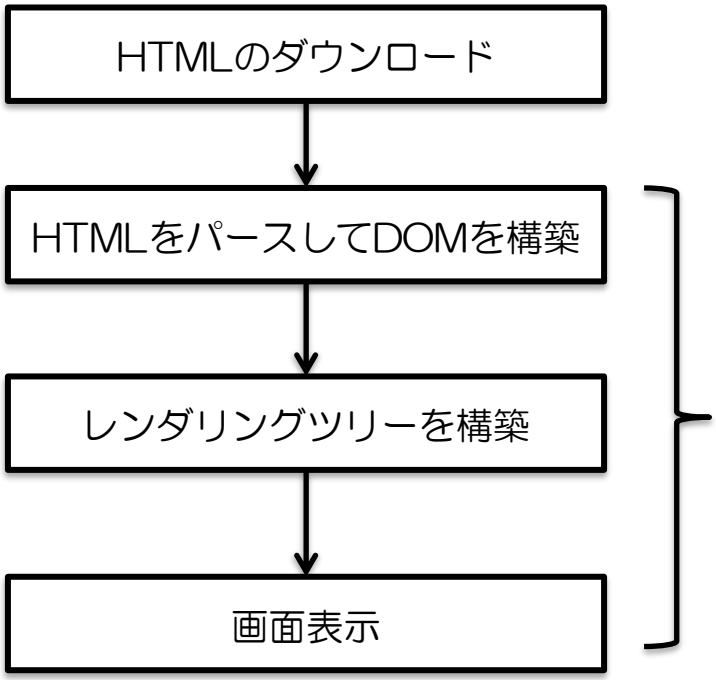
HTMLファイル中への記述方法

- HTMLファイル中に記述する際は<script>タグを使う。
- <script>～</script>の間にJavaScriptのプログラムを記述する。
- 「<!--」と「// -->」で囲む（JavaScript非対応ブラウザ対策）

```
<body>
<p>
このWebブラウザのuserAgentは
<script>
<!—-
document.write(navigator.userAgent)
// -->
</script>
です。
</p>
</body>
</html>
```

JavaScriptをどこに書くか？

- HTMLファイル中に<script>の記述があるとパース一時停止
  ⇒特に問題がなければ後ろの(</body>の手前)が望ましい。
- 表示前に実施する必要がある場合は<head>の中に
- パースが行われていないタグに対する操作は出来ません。



ある程度、パース
出来たら画面表示

ブラウザで実施されるパースの例
1. 文字列だけ
2. 閉じてないタグ
3. Bodyの外の記述

関数呼び出し

- この例では、documentクラスのwriteメソッドがnavigator.userAgentというパラメータで呼び出される。
- navigator オブジェクト　userAgent パラメータ
  HTTPリクエストでのUser-Agentヘッダの情報
- navigator.userAgentには、現在のWebブラウザのユーザエージェント（ブラウザ種別に関する情報）の文字列が入っている。
- document.writeは文字列を出力するメソッド。HTMLファイルにその文字列を書いたのと同じことになる。

このWebブラウザのuserAgentは

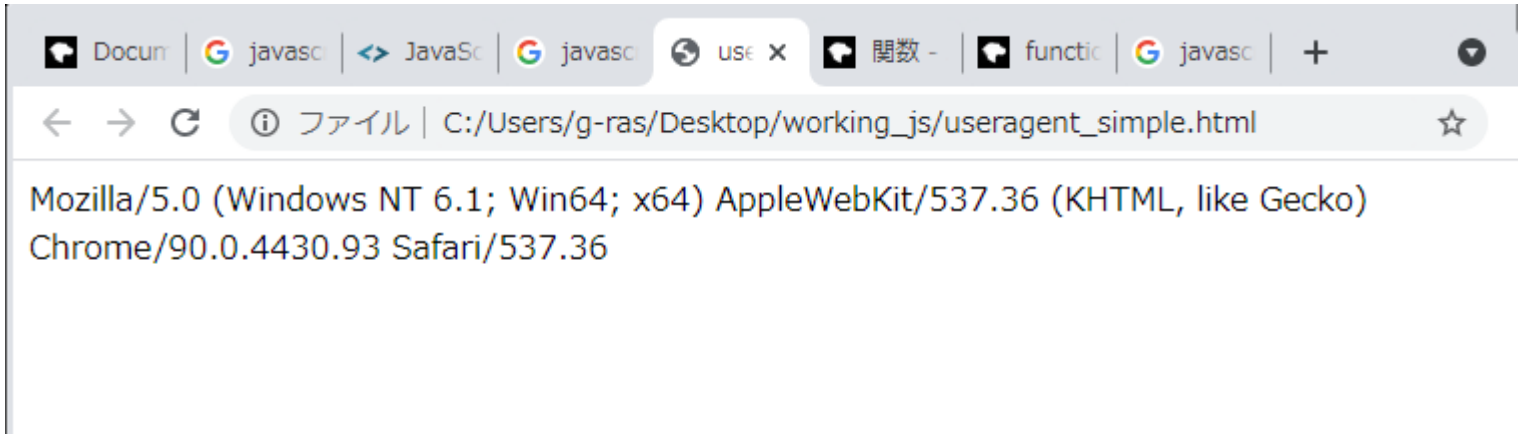
```
<script>
<!—-
document.write(navigator.userAgent)
// -->
</script>
```

です。

【演習04-01】関数呼び出し

userAgentの内容を表示するHTMLファイルを作成する。
表示内容をキャプチャし、SOLAに提出すること

◆形式
JPEGまたはPNG形式



※User-agent sniffing

変数とif文（1）

- 変数やif文も使える。

```
var useragent = navigator.userAgent

if (~useragent.indexOf("Trident")) {
   document.write("Internet Explorer")
} else if (~useragent.indexOf("Edge")) {
   document.write("Edge")
} else if (~useragent.indexOf("Chrome")) {
   document.write("Chrome")
} else if (~useragent.indexOf("Firefox")) {
   document.write("Firefox")
} else if (~useragent.indexOf("Safari")) {
   document.write("Safari")
} else {
   document.write("その他")
}
```

変数

■ 変数の宣言：let, const, var

宣言の種類	再代入	再宣言	スコープ
const	×	×	ブロック
let	○	×	ブロック
var	○	○	関数

<u>再代入</u>

×

```
const  a = 10
a = 11
```

<u>再宣言</u>

×

```
let  a = 10
let  a = 11
```

変数とif文（2）

- 変数はconst, letで宣言する。
- 変数の型は指定しなくてもうまくやってくれる。代入したものの型が適用される。代入されないと型が決まらないので、undefinedという型になる。
- 文字列(string)、数値(number)、論理型(boolean)、null、未定義(undefined)の5種類がプリミティブ型。
- 目的に合わせて型変換が必要な場合もある。

```
const useragent
```

宣言のみ・・・型はまだ決まらない（undefined）

```
useragent = navigator.userAgent
```

代入・・・右辺に合わせてstring型になる

宣言時に値も代入・・・右辺に合わせてstring型になる

```
const useragent = navigator.userAgent
```

```
String(1234)    Number("1234")
```

変数とif文（3）

- if文は条件（括弧内）の値を評価した結果がtrueかfalseかによって分岐する。
- else ifによって2つ目以降の条件を列挙できる。
- どれにも合致しなかった場合はelse節が（あれば）実行される。

```
if (条件1) {
   document.write("条件1")
} else if (条件2) {
   document.write("条件2")
} else {
   document.write("その他")
}
```

変数とif文（4）

- 条件は下表の場合にfalseとなる。
- これ以外はたいてい true。

型	値	例
boolean	false	var x = 1 if (x < 0) ...
文字列 (string)	"" （空の文字列）	var x = "" if (x) ...
数値 (number)	0	var x = 0 if (x) ...
undefined	undefined （型の決まっていない変数）	var x if (x)
null	null	var x = null if (x)

変数とif文（5）

- 単項演算子「！」は論理を反転する。
- 単項演算子「～」（チルダ）は値のビットを反転する。

```
if (!x) ...
```

xを論理値として評価した後、論理を反転（true⇔false）する。

```
if (~useragent.indexOf("Internet Explorer")) ...
```

indexOfは文字列中にパラメータに合致する部分があれば、その先頭文字のインデックスを返す。

見つかれば0以上の数値、見つからない場合は-1を返す。

～で反転すると、見つかった場合は0以外(true扱い)、見つからなかった場合は0（false扱い）になる。

関数定義(1)

- 関数はfunctionで定義する。
- 返り値の型はreturnで値を返す時に決まる。

```
function browser() {
  var useragent = navigator.userAgent

  if (~useragent.indexOf("Trident")) {
     return "Internet Explorer"
  } else if (~useragent.indexOf("Edge")) {
     return "Edge"
...
  } else {
     return "その他"
  }
}
```

関数定義(2)

- 関数は「関数名()」で呼び出す。
- 関数の括弧内にはパラメータが入れられる。

```
document.writer("<p>")
browser()
document.writer("</p>")
```

【演習04-02】JavaScript(2)

1. /var/www/html/sumation.html を以下の内容で作成。

```
<body>
 <h1>JavaScriptの練習2</h1>
  <script type="text/javascript">
      <!--
      function tashizan(x,y)  {
              return x+y
      }
      // -->
  </script>
      <p>
      <input type="number" id="num_a" value=2>
      <input type="number" id="num_b" value=3>
      </p>
      <p>
       <input type="button" value="足し算" onclick="alert(tashizan(+num_a.value,+num_b.value))">
       </p>

 </body>
</html>
```

【演習04-02】JavaScript(2)

◆課題

sumation.htmlを作成し、titleやdescriptionも適宜修正する。ブラウザで表示し、ボタンを押したところの画面キャプチャを取る。

◆形式

JPEGまたはPNG形式

◆提出方法

SOLAでキャプチャファイルを提出

【演習04-03】足し算・引き算

◆課題

sumation.htmlをコピーし、subtraction.html作成する。
引き算を計算する関数hikizan()と引き算のボタンを追加する。
ブラウザで表示し、引き算のボタンを押したところの画面キャプチャを取る。

◆形式

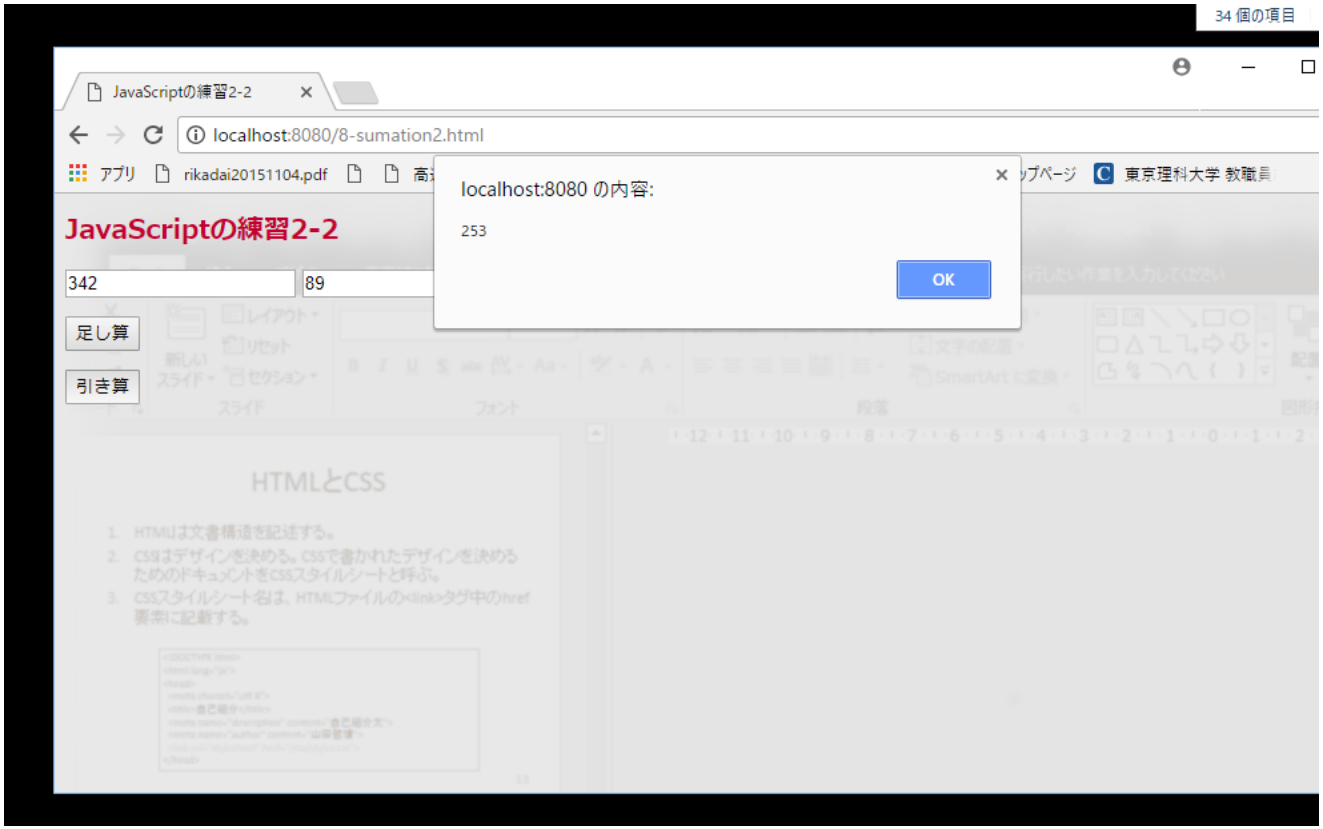
JPEGまたはPNG形式

◆提出方法

SOLAでキャプチャファイルを提出

【演習04-03】足し算・引き算

- 例



オブジェクト(1)

- プリミティブ型以外はオブジェクト型(object)。
- 関数もオブジェクト型。
- オブジェクトはクラスからnewで生成し、オブジェクト型で記憶しておく。
- オブジェクトの要素にはドット「.」でアクセスする。

```
let p = new Person()
p.name = 'yamada'
p.age = 51
p.married = true
p.print()
```

オブジェクト(2)

- クラスの定義にはfunctionを使う。
- this.xxxで、xxxというプロパティ(クラスの要素)を定義する。
- メソッドもプロパティとして定義する。functionで名無しの関数を作って、プロパティに代入する。

```
function Person(name, age, married) {
  this.name = name
  this.age = age
  this.married = married
  this.print = function () {
    // メソッドの中身
  }
}
```

オブジェクト(3)

- オブジェクト内で、自分のプロパティにアクセスするにはthis.を使う。

```
this.print = function () {
  document.write('<p>')
  document.write(this.name + 'さんは')
  document.write(this.age + '歳で')
  if (this.married) {
     document.write('既婚です')
  } else {
     document.write('未婚です')
  }
```

【演習04-04】JavaScriptファイル

1. /var/www/html/js/scripts.js を以下の内容で作成。

```
function browser() {
   var useragent = navigator.userAgent
   if (~useragent.indexOf("Trident")) {
      document.write("Internet Explorer")
   } else if (~useragent.indexOf("Edge")) {
      document.write("Edge")
   } else if (~useragent.indexOf("Chrome")) {
      document.write("Chrome")
   } else if (~useragent.indexOf("Firefox")) {
         document.write("Firefox")
   } else if (~useragent.indexOf("Safari")) {
         document.write("Safari")
   } else {
         document.write("その他")
   }
}
```

（続く）

```
function Person(name, age, married) {
  this.name = name
  this.age = age
  this.married = married
  this.print = function () {
    document.write('<p>')
    document.write(this.name + 'さんは')
    document.write(this.age + '歳で')
    if (this.married) {
       document.write('既婚です')
    } else {
       document.write('未婚です')
    }
    document.write('</p>')
  }
}

function OnButtonClick(a,b) {
    target = document.getElementById(a);
    target.innerHTML = "足し算の結果: "+b;
}
```

【演習04-04】JavaScriptファイル

1. /var/www/html/function.html を以下の内容で作成。titleやdescriptionも適宜修正する。

```
（省略）
<body>
<h1>JavaScriptの練習4</h1>
<p>
このWebブラウザは
<script src="/js/scripts.js"></script>
<script>
<!--
browser()
// -->
</script>
だと思われます。
<p>
</body>
</html>
```

【演習04-04】JavaScriptファイル

◆課題

function.htmlをブラウザで表示したところの画面キャプチャを取る。

◆形式

JPEGまたはPNG形式

◆提出方法

SOLAでキャプチャファイルを提出

【演習04-05】クラスの利用

1. /var/www/html/object.html を以下の内容で作成。内容は適宜変更する。titleやdescriptionも適宜修正する。

```
<body>
<h1>JavaScriptの練習5</h1>
<p>
<script src="/js/scripts.js"></script>
<script>
<!--
let p = new Person('理科大太郎', 5, true)
p.print()
// -->
</script>
</p>
</body>
</html>
```

【演習04-05】クラスの利用

◆課題

object.htmlをブラウザで表示したところの画面キャプチャを取る。

◆形式

JPEGまたはPNG形式

◆提出方法

SOLAでキャプチャファイルを提出

【演習04-06】クラスの利用(2)

1. /var/www/html/object2.html を以下の内容で作成。内容は適宜変更する。titleやdescriptionも適宜修正する。

```
<body>
<h1>JavaScriptの練習6</h1>
<p>
<script src="/js/scripts.js"></script>
<script>
<!--
var p = new Person2('理科大太郎', 5, true, true)
p.print()
p.print2()
// -->
</script>
</p>
</body>
```

【演習04-06】クラスの利用(2)

◆課題

object2.htmlで使用するクラスPerson2をスクリプトファイル
/var/www/html/js/scripts.jsの中に定義せよ。

ただし、Person2は下記の通りとする。

①コンストラクタの4番目の引数として性別を示す（男性の場合、true）

②メソドprint()の他にprint2()を有する。print2()は“XXXさんはYYYのZZZです。”という文章をHPに表示する。ただし、XXX、YYY、ZZZは下記の通りとする。

- ・XXX　名前
- ・YYY　10歳未満は“子供の”、10歳以上20歳未満は”10代の”、20歳以上30歳未満は”20代の”、30歳以上65歳未満は”大人の”、65歳以上は”老人の”
- ・ZZZ　“男性”もしくは“女性”

【演習04-06】クラスの利用(2)

object2.htmlをブラウザで表示したところの画面キャプチャを取る。

◆形式
JPEGまたはPNG形式

◆提出方法
SOLAでキャプチャファイルを提出

【演習04-06】クラスの利用(2)

例

【演習04-07】アニメーション

1. ball.html, ball.css, ball.js を作成し、それぞれを適切なディレクトリに置く。

   ball.html ⇒ /var/www/html
   ball.css ⇒ /var/www/html/css
   ball.js ⇒ /var/www/html/js

2. ball.htmlにアクセスし、ball.jsを実行

3. 実行画面のキャプチャを取る。

【演習04-07】アニメーション

1. /var/www/html/ball.html を以下の内容で作成。
   内容は適宜変更する。

```html
<!DOCTYPE html>
<html lang="ja">
<head>
  <meta charset="utf-8">
  <title>JavaScript</title>
  <meta name="description" content="JavaScriptの練習">
  <meta name="author" content="理科大太郎">
  <link rel="stylesheet" href="/css/ball.css">
</head>
<body>
<h1>JavaScriptの練習</h1>
<span id="ball"></span>
<script src="/js/ball.js"></script>
</body>
</html>
```

※ spanタグはdivタグの改行なし版です。

【演習04-07】アニメーション

1. /var/www/html/css/ball.css を以下の内容で作成。
   内容は適宜変更する。

```
body {
    margin:0;
    padding:0;
    background-color: #333;
    color: #9ff;
}

#ball {
    display: block;
    background: black;
    border-radius: 50%;
    height: 300px;
    width: 300px;
    margin: 0;
    background: radial-gradient(circle at 100px 100px, #5cabff, #000);
    position: absolute;
}
```

【演習04-07】アニメーション

```
// ball.js
let ball    = document.getElementById("ball")
const bh    = ball.offsetHeight
const bw    = ball.offsetWidth
const bleft = ( window.innerWidth / 2 ) - ( bw / 2 )
const btop  = -ball.offsetHeight
const ground    = window.innerHeight - bh

ball.style.left = bleft + "px"
ball.style.top = btop + "px"

const b = ground
const R = 0.85

let y = btop
let v = 0

function gravity() {
    y+= ++v
    if( y >= b ) {
        y = b
        v*= -R
    }
    ball.style.top = y + "px"
}

setInterval(gravity, 10)
```

【演習04-07】アニメーション

◆課題

ball.htmlをブラウザで表示したところの画面キャプチャを取る。

◆形式

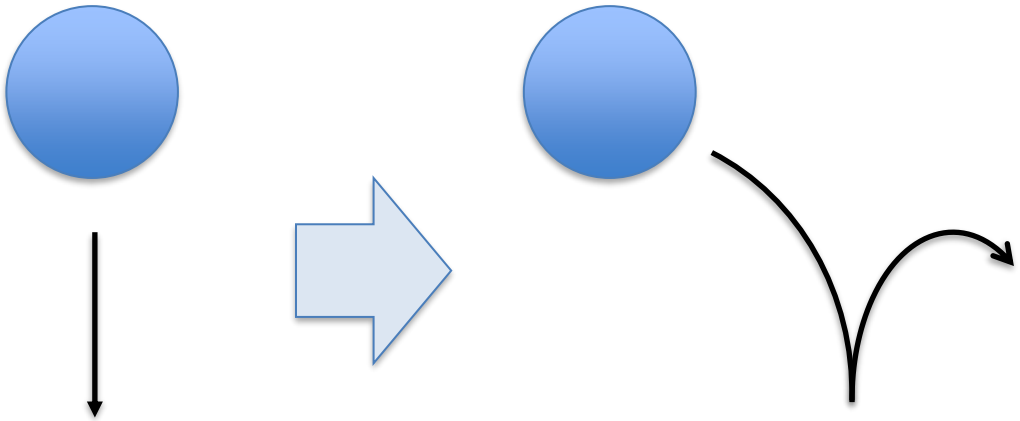
JPEGまたはPNG形式

◆提出方法

SOLAでキャプチャファイルを提出

【演習04-08】アニメーション(2)

- 演習04-07で作成したアニメーションをボールが放物線を描くように変更せよ。ただし、ボールが画面の外にでないようにすること。
- JavaScriptファイルball.jsを変更することで実現すること
- 変更した**ball.js**をSOLAに提出



【宿題04-01】

◆宿題

/var/www/html/js/scripts.js を修正し、object.htmlの表示を表形式にする。
表のためのCSSファイルの読み込みなど、適宜object.htmlは修正する。
ブラウザで表示させて画面キャプチャを取る。

◆提出物

HTMLファイル

JavaScriptファイル

CSSファイル

画面キャプチャ(JPEGまたはPNG形式)

◆提出方法

SOLAで提出

◆締切

2024年5月7日(火)正午まで

サーバシステムプログラミング及び演習

－第5回 PHP －

本日の内容

1. PHP
   - 概要
   - スクリプトの作成・実行方法
   - 変数
   - 文字列の連結
   - 連想配列
   - 関数の定義
2. 演習
3. 課題

PHP

CGI

1. CGI（Common Gateway Interface）は、Webサーバーシステムでプログラムを実行させる仕掛け。
2. CGIの仕掛けを使って動かすプログラムを、CGIプログラムという。
3. CGIプログラムは、様々な言語で記述できるが、一般的に以下のような条件を満たす言語が使われる。
   a. Webサーバプログラム（Apache等）からの呼び出しが簡単に行える。
   b. 文字列の扱いが容易である。
4. LAMPにおいては、PHP, Perl, PythonがCGIプログラムを記述する言語である。
   - PHP: Webに特化
   - Perl: 文字列処理最強
   - Python: すっきりきれいに書ける

PHP

1. PHPは「PHP: Hypertext Preprocessor」の略。
2. オブジェクト指向のスクリプト言語である。
3. HTMLデータの生成を得意とする。
4. コンパイル不要で、インタプリタにより逐次実行されるため、作成・更新後、そのまますぐに実行できる。

PHPプログラム

1. PHPプログラムは、「<?php」と「?>」で囲む。

   例

   ```
   <?php
   echo date("Y/m/d");
   ?>
   ```

2. PHPプログラムは、CLIで実行できる。

   例

   ```
   $ php date.php
   ```

3. HTMLにPHPプログラム直接埋め込むこともできる。

   例

   ```
   <body>
   <p>今日は<?php echo date("Y/m/d"); ?>です。</p>
   </body>
   ```

変数と制御構造

1. CやJavaと同様に、変数、分岐、ループなどの構文を備える。
   - 変数（数値、文字列、配列）
   - 条件分岐（if, switch）
   - ループ（for, while, do-while）
2. 独特な機能として、連想配列がある。
   - 連想配列は、配列のキーを、整数のインデックスではなく、文字列にすることができる。
   - 連想配列に格納したデータに対し、文字列のキーを指定することで、合致した配列の要素を取り出すことができる。
   - Web上でのユーザの一連の入力や、データベースから取り出した一連のデータを、ひたすら連想配列に突っ込んでおき、最後に探しだして処理をするというのが「富豪的プログラミング」。メモリを消費したり実行効率が悪かったりするが、プログラムのロジックが簡単で短くなるため、スクリプト言語を使う際にはよく使われる。

Javaとの違い

1. クラスがなくても動かせる。
2. 変数を宣言なしで使えてしまう。型は推論で決まる。
3. オブジェクト（インスタンス）のメソッドやメンバへのアクセスが、「.」ではなく「->」。
4. 文字列の連結が「+」ではなく「.」。
5. 連想配列

など

PHP設定

1. ApacheからPHPを実行できるようにする。
   - Apacheの設定にて、CGIとしてPHPスクリプトを実行できるようにする必要があります。
   - 具体的には、/etc/apache2/mods-enabled/php5.confの設定を変更します

phpinfo（1）

1. PHPスクリプトの作成

- /var/www/html/php/info.php を以下の内容で作成

```
<?php
  phpinfo();
?>
```

① mkdir -p /var/www/html/php

② gedit /var/www/html/php/info.php（または、Windows上でNotepad++）

実

phpinfo（2）

2. PHPスクリプトの実行

- コマンドライン（CLI)から以下のように実行できます。

① php /var/www/html/php/info.php

- たくさん表示が出てくるので、lessなどで出力を受け取ると良いです。

② php /var/www/html/php/info.php | less

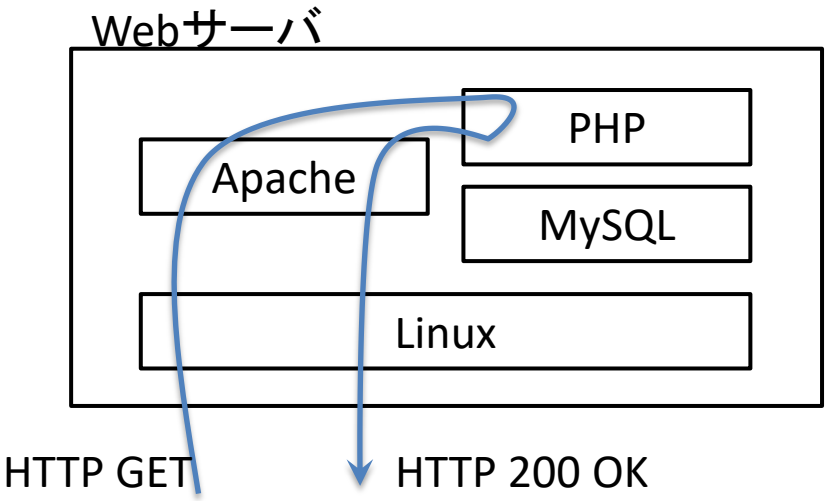
(lessの終了は"q"です）

実

phpinfo（3）

3. 通しでの確認

- Windowsのブラウザから http://localhost:8080/php/info.php にアクセス。（※その他の設定の方は適宜）
- PHPの設定内容が表形式で表示されれば成功



phpinfo（4）

4. phpinfo関数
   - PHPのphpinfo関数のマニュアルは下記にある。
   - http://php.net/manual/ja/function.phpinfo.php
   - 関数phpinfoは、CLIモードで起動したときはプレーンテキストで結果を返します。Apacheから起動したときはHTMLで表形式で結果を返します。

date（1）

1. PHPスクリプトの作成

- /var/www/html/php/date.php を以下の内容で作成

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<p>今日は' . date("Y年m月d日") . "です。</p>¥n";
?>
```

① gedit /var/www/html /php/date.php

- 半角文字の￥は＼（半角バックスラッシュ）と同じ文字コードです（表示だけ異なる場合がある）。

date（2）

2. PHPスクリプトの実行

- コマンドライン（CLI)から以下のように実行できます。

① php /var/www/html/php/date.php

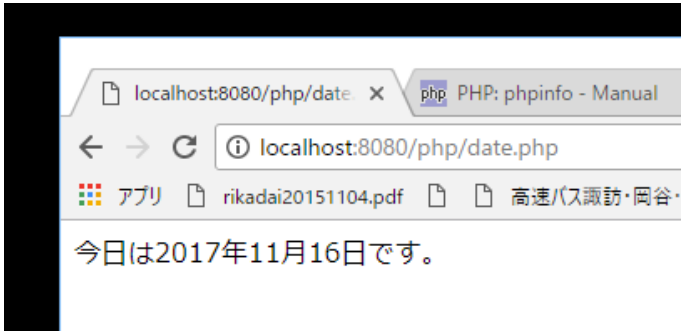
- 次のような出力が得られれば成功です。

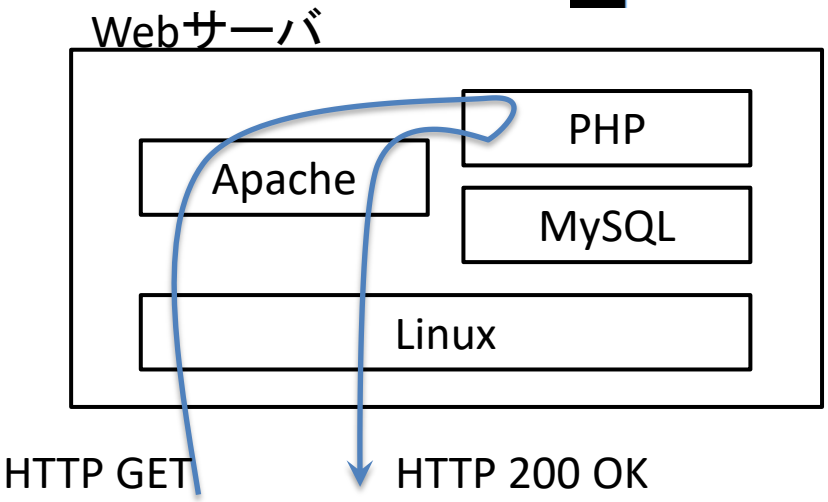
今日は2016年11月17日です。

date（3）

3. 通しでの確認

- Windowsのブラウザから http://localhost:8080/php/date.php にアクセス。
- 日付が表示されれば成功





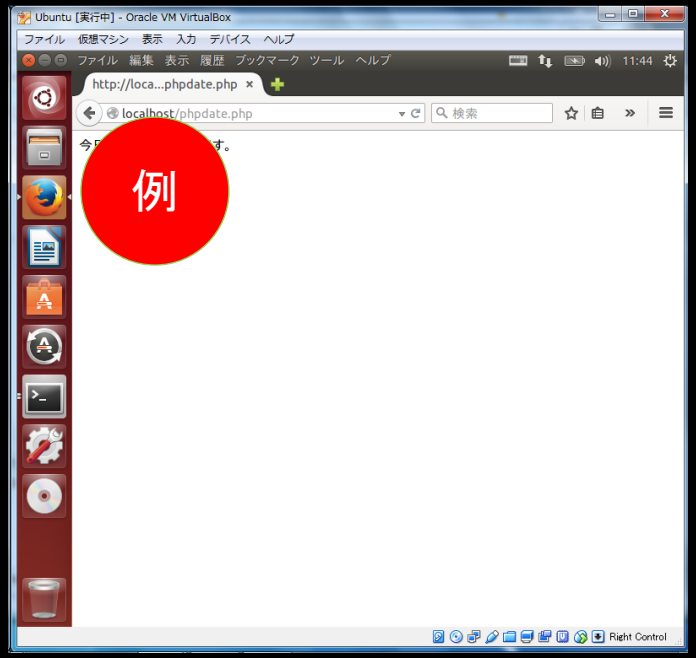
date（4）

4. date関数
   - PHPのdate関数のマニュアルは下記にある。
   - http://php.net/manual/ja/function.date.php
   - 年を何桁にするか、月や日が1桁の場合に先頭にゼロを付けるかどうかなどを指定できる。

実

演習05-01

- date.php をコピーして datetime.php を作り、2001-03-10 17:16:18の形式で日時を表示するように修正する。
- ブラウザから datetime.php にアクセスする。
- 表示画面をキャプチャし、SOLAに提出すること



PHPでの動的Webページ生成

1. PHPでHTMLファイルを生成することで、動的Webページが実現できます。

2. PHPでは、計算を行ない、その結果（数値や文字列）を出力したり、HTMLタグを出力したりすることで、毎回違う画面を出力することが出来ます。

文字コードの指定

1. PHPで生成するHTMLファイルに日本語が含まれている場合、下記のように文字コードを指定します。

```
header('Content-Type: text/html; charset=UTF-8');
```

変数

1. PHPでの変数は、先頭に$を付けます。

   $number = 100;

   $string = 'hello';

2. 変数の型は、代入される値の型により、推論で決められます。このような言語を、動的型付け言語と言います。LAMPのPであるPerl, PHP, Pythonはいずれも動的型付け言語です。

3. 変数の型はいったん決まると、それ以降は変更できません。

文字列の出力

1. PHPでタグを出力するには、echo関数を使用します。

   echo ’ 段落です ’;

2. 改行文字（¥n）など、特殊記号を含む文字列の場合、ダブルクオートで囲みます。ダブルクオートは文字列の中身を解釈して変換して欲しいという意図を示します。

   echo ″ 段落です ¥n″;　　# OK

   echo ‘ 段落です ¥n’；# ダメ

文字列の連結

1. 文字列はドット（.）で連結できます。変数もドットで連結します。

```
echo '<p>あなたは' . $n . "人目のお客様です</p>¥n";
```

2. 文字列の中に変数を埋め込むには、変数名を{}で囲みます（変数の終わりが分かるように）。変数を含む文字列は、ダブルクオートで囲む必要があります。

```
echo "<p>あなたは${n}人目のお客様です</p>¥n";     # OK
echo "<p>あなたは$n人目のお客様です</p>¥n";    # ダメ
echo '<p>あなたは${n}人目のお客様です</p>¥n ";    # ダメ
```

連想配列（1）

1. 連想配列とは、配列のインデックスとして、連続的な数値だけではなく、任意の値（文字列など）を使うことができる配列です。

```
$address['yamada'] = '長野県茅野市ちのxxx-yy';
$address['kasuga'] = '東京都品川区aaa-bbb';
```

2. 連想配列のインデックスをキー、中身の値をバリューと呼びます。キーとバリューのペアを使って前述の例は以下のようにも書けます。

```
$address = [
  'yamada' => '長野県茅野市ちのxxx-yy',
  'kasuga' => '東京都品川区aaa-bbb'
];
```

連想配列（2）

1. PHPでは、連想配列と通常の配列の区別はなく、全て同一の配列として扱われます。

2. キーを特に指定しない場合、0から始まる整数のうち使われていない最小のものがキーとして振られます。

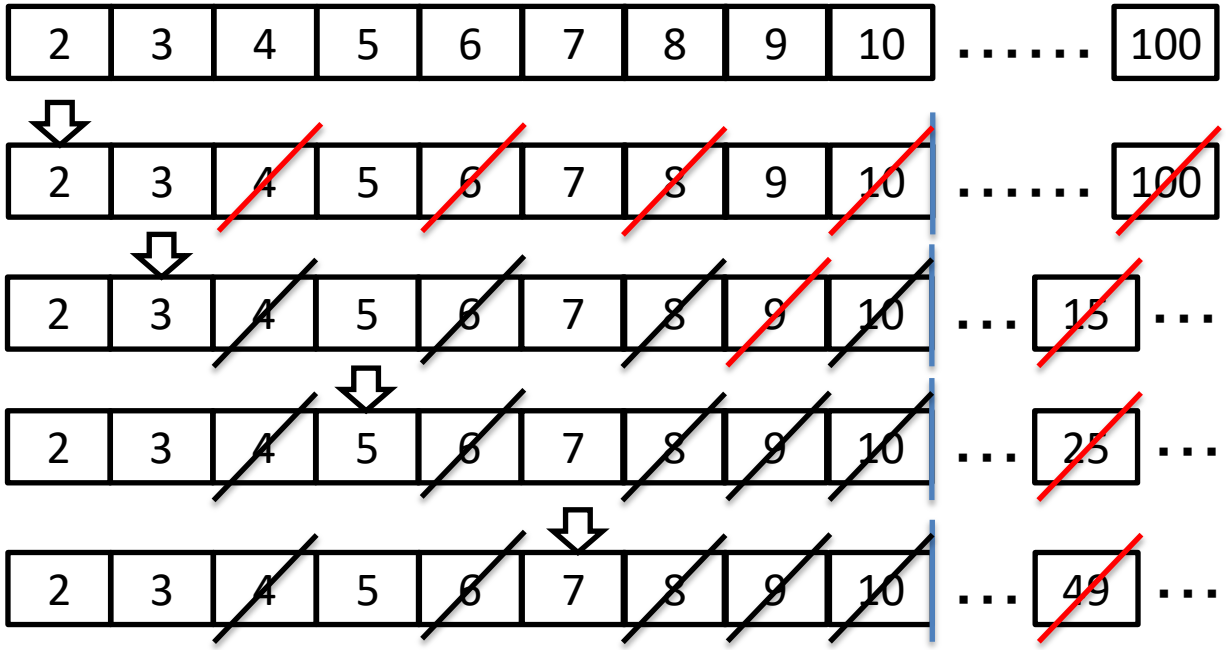
```
$x[] = 'aaa';
$x['hey'] = 'bbb';
$x[] = 'ccc';
print_r($x);
(
    [0] => aaa
    [hey] => bbb
    [1] => ccc
)
```

今回使う関数

1. floor()： 少数以下を切り捨てます。
2. ceil()：少数を切り上げます。
3. sqrt()：平方根を求めます。
4. array_fill()：配列を指定した値で埋めます。
5. isset()：配列に含まれているか調べます。
6. unset()：配列から取り除きます。
7. array_keys()：配列のキーを返します。

PHPでの計算(解説(1)エラトステネスの篩)

1. 2～最大値(100)までの配列を作る
2. 2～最大値の平方根(10)までループを回す。
3. その値がまだ配列に残っていれば、その2倍、3倍、4倍…の値を配列から取り除く



※実際は取り除いた分だけ配列は短くなります

key	0	1	2	3	4	
value	2	3	5	7	11	…

PHPでの計算(解説（2）)

1. funtionは関数を定義します。
2. 2〜$maxまでの値が全て入った配列を用意し、その中から素数ではないものをunsetによって取り除きます。
3. return array_keys($lists)は素数の配列をインデックス付きで返します。

```
(
    [0] => 2
    [1] => 3
    [2] => 5
      …
)
```

4. for (eratosthenes($m) as $key => $value)は、$keyと$valueに上記の返り値を順次入れてループします。

実

演習05-02

- 次ページの内容を、/var/www/html/php/eratosthenes.phpとして入力します。
- CLIから実行します。

```
php /var/www/html/php/eratosthenes.php
```

- Webブラウザからアクセスします。
- 画面をキャプチャしSOLAに提出すること

演習05-02

エラトステネスの篩

```
<?php
function eratosthenes($max) {
   $sqrt = floor(sqrt($max));
   $lists = array_fill(2, $max-1, true);
   for ($i=2; $i<=$sqrt; $i++) {
      if (isset($lists[$i])) {
         for ($j=$i*2; $j<=$max; $j+=$i) {
            unset($lists[$j]);
         }
      }
   }
   return array_keys($lists);
}

header('Content-Type:  text/html; charset=UTF-8');
$m = 100;
echo '<p>' . $m . "までの素数を求めます。</p>¥n";
foreach (eratosthenes($m)  as $key => $value) {
   echo '<p>' . ($key+1) . '番目の素数は' . $value . "です。</p>¥n";
}
?>
```

tableタグの出力

1. <table border="1">というタグを改行付きで出力するには、ダブルクオートの前にバックスラッシュを付けて以下のように書きます。

```
echo "<table border=¥"1¥">¥n";
```

2. $mに数値が入っている場合、以下の処理は、数値を含んだ表のキャプションを出力します。

```
echo '<caption>' . $m . "までの素数</caption>¥n";
```

演習（4）表出力

1. 表を出力するPHPプログラム

```
（省略）

header('Content-Type: text/html; charset=UTF-8');
$m = 100;
echo "<table border=¥"1¥">¥n";

    ？？？

echo "</table>¥n";
```

実

演習05-03

- eratosthenes.phpをコピーしてeratosthenes_table.phpを作成し、結果を表形式で表示するようにする。
- Webブラウザでアクセスする。
- 画面をキャプチャしSOLAに提出すること

演習（4） カレンダ（1）

1. カレンダを出力するPHPプログラム

```
<?php
function day_of_the_week($y,$m) {
	$datetime = new DateTime();
	$datetime->setDate($y, $m, 1);
	$w = (int)$datetime->format('w');
	return $w;
}
```

(次に続く）

演習（4） カレンダ（2）

1. カレンダを出力するPHPプログラム（続き）

```
header('Content-Type: text/html; charset=UTF-8');
$y = 2020;
$m = 6;
$w = day_of_the_week($y,$m);
$n = cal_days_in_month(CAL_GREGORIAN, $m, $y);

echo "<table border=¥"1¥">¥n";
echo '<caption>'.$y. "年 ". $m . "月</caption>¥n";
echo "<tr><th>日</th><th>月</th><th>火</th><th>水</th><th>木</th><th>
金</th><th>土</th></tr>¥n";

// カレンダを表示するコードを追記せよ。

echo "</table>¥n";

?>
```

実

演習05-04

- calendar_table.phpを作成し、結果を表形式で表示するようにする。
  ※任意の年、月でも正しく表示できること
- Webブラウザでアクセスする。
- 画面をキャプチャしSOLAに提出すること

宿題05-01

- eratosthenes_table.phpで、$mの値を10倍ずつ大きくしていった場合、どこまで表示できるか。
- 表が正しく表示された上限のブラウザの表示を、表のキャプションが見えるようにして画面キャプチャを取る。
- キャプチャファイルをSOLAに提出すること

宿題05-02

- 宿題05-01の値の上限は何の制限によるものか。エラーを起こしている可能性も含めて調査せよ。
- 調査内容をSOLAに提出すること

◆提出締切：2024年5月14日(火)正午

おわり

サーバシステムプログラミング及び演習

― 第06回 HTTP ―

本日の内容

1. HTTP
   - プロトコル概要
   - レイヤ構成
   - 基本シーケンス
   - メソド　種類、起動
2. POST
3. PHPでの取り扱い
4. 演習

HTTP

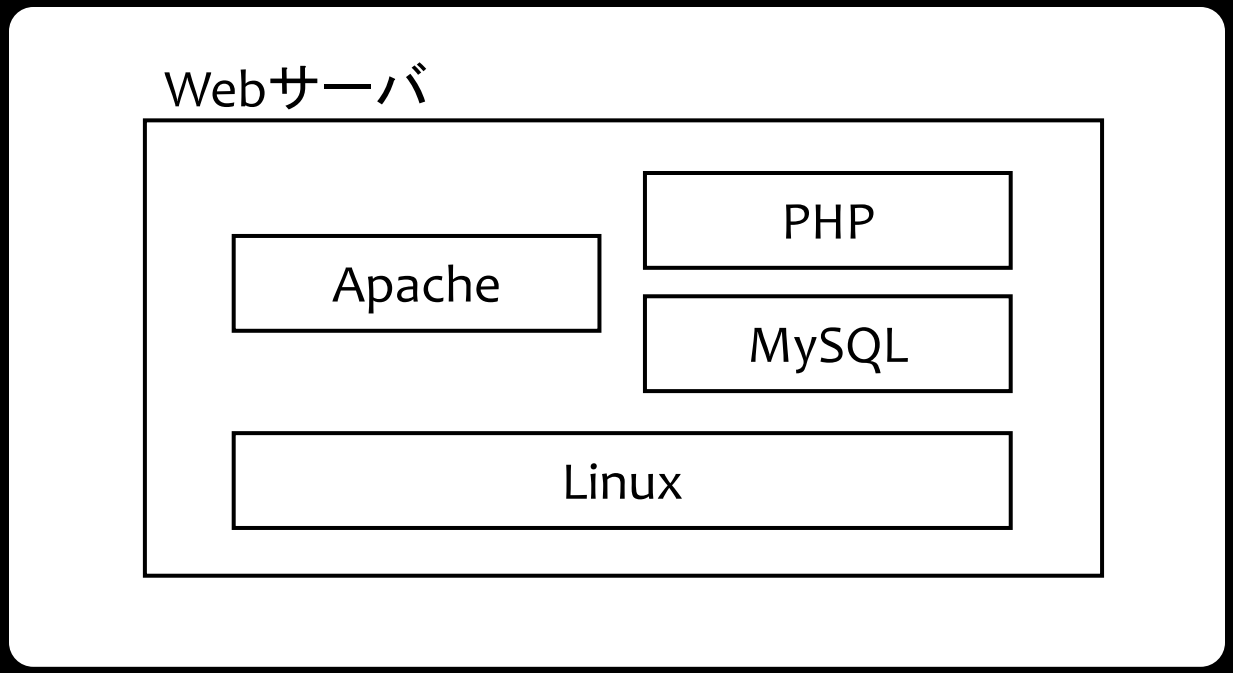
Webサーバの構成

L: OS、IP通信

A: Webサーバ

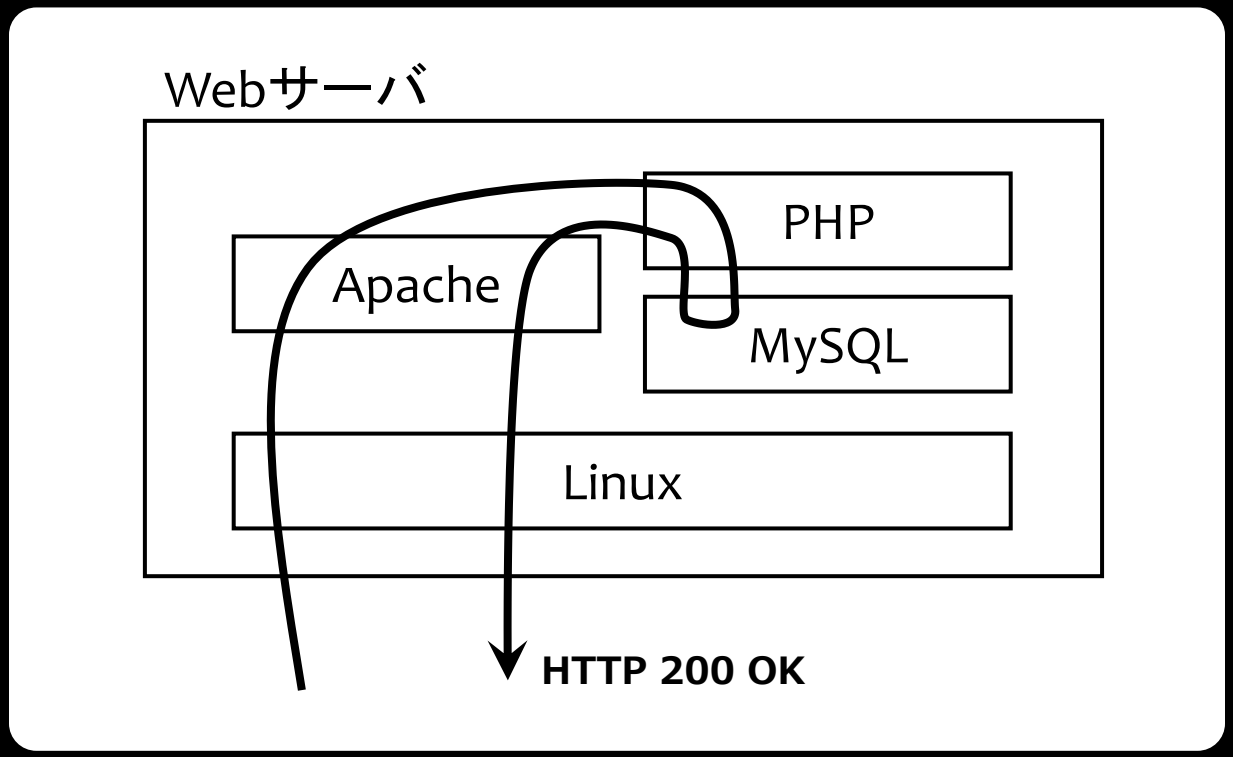
M: データベース

P: ダイナミックなWeb情報（HTML）の生成



LAMP　全体の動作

1. HTTPのリクエストをLinuxがApacheに渡す
2. Apacheは必要に応じてスクリプトを起動
3. スクリプトはデータベースを読み込んだりして動的ページを生成
4. Apache経由で動的ページを返却

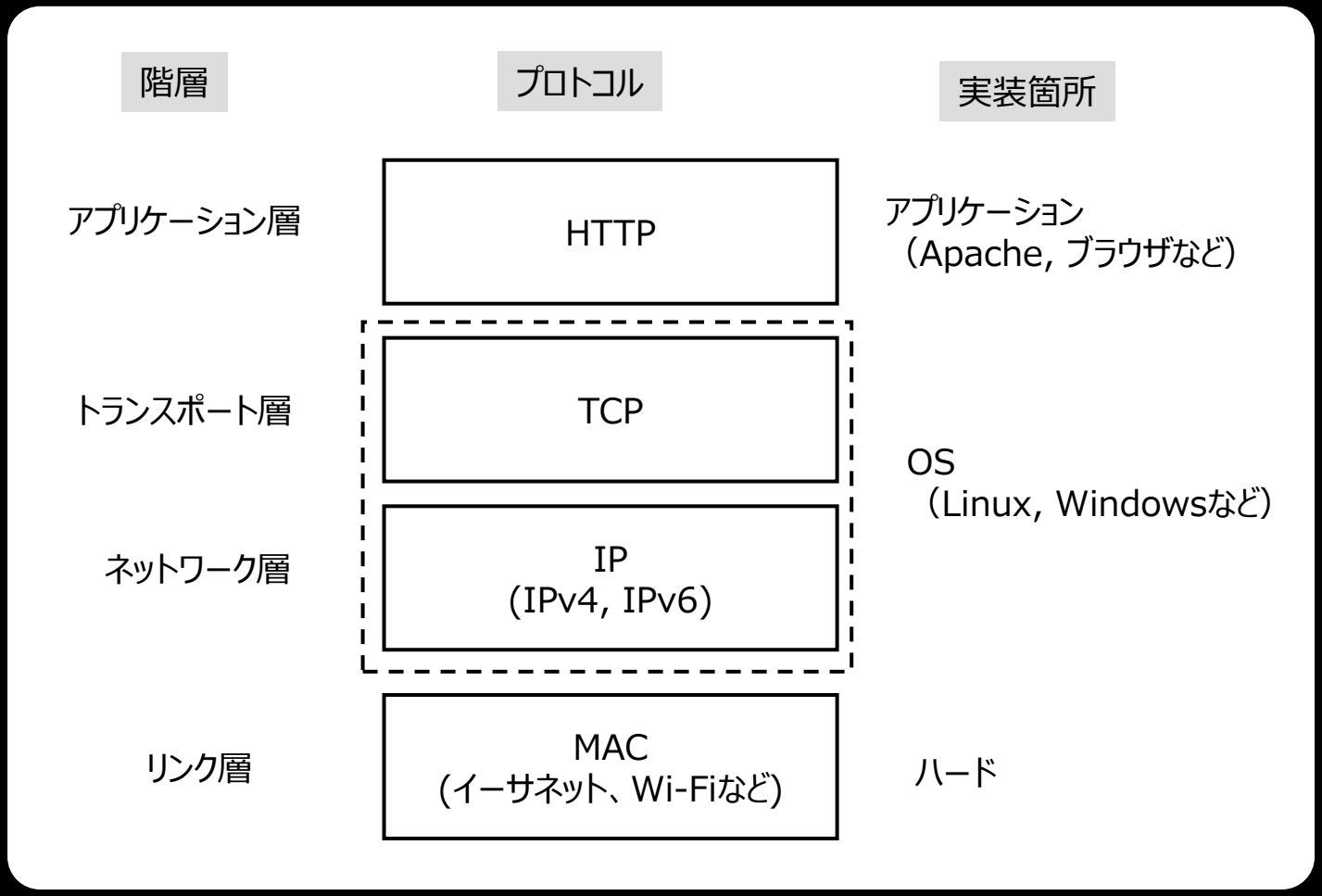


Webブラウザ

1. 通信プロトコルはHTTP（Hypertext Transfer Protocol）
2. 通信相手はURL（Uniform Resource Locator）にて指定
3. HTML（Hyper Text Markup Language）形式のドキュメントをやりとりする。HTMLはテキスト形式のマークアップ言語。
4. ブラウザはHTMLファイルを受け取ると、HTMLレンダリングエンジンにより内容を解釈して画面描画を行う。
5. HTMLファイル内でスタイルが指定されていると、HTMLレンダリングエンジンはスタイルに従って画面描画を行う。
6. HTMLファイル内にJavaScriptによるスクリプトが埋め込まれている場合、JavaScriptエンジンによりスクリプトが実行される。

HTTP　プロトコルのレイヤ構成

1.　HTTPの位置づけを下図に示す。



HTTP　概要

1. アプリケーションレイヤの通信プロトコル
2. ステートレス　(1回送ったら(リクエスト)、1回受けて(レスポンス)、それで終わり)
3. セッション管理はCOOKIE等、他の技術との組み合わせで実現
4. テキストデータをやり取り
5. 暗号化なし
6. サーバに要求する動作(メソッド)が規定されている。
7. やり取りの結果はステータスコードで示される。(3桁数字を英文：200　OK, 404 Not Found, 403 Forbidden　等)

HTTP　リクエストとレスポンスの構成

1.　メッセージ構成

```
GET / HTTP/1.1   リクエスト行
```

```
Accept: image/gif, image/jpeg, */*     ヘッダ
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (Compatible; MSIE 6.0; Windows NT 5.1;)
Host: www.xxx.zzz
Connection: Keep-Alive
```

```
空行
```

```
メッセージボディ（POSTメソッドなどで使用）
```

```
HTTP/1.1 200 OK    レスポンス行
```

```
Date: Sun, 11 Jan 2004 16:06:23 GMT     ヘッダ
Server: Apache/1.3.22 (Unix) (Red-Hat/Linux)
Last-Modified: Sun, 07 Dec 2003 12:34:18 GMT
ETag: "1dba6-131b-3fd31e4a"
Accept-Ranges: bytes
Content-Length: 4891
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

```
空行
```

```
<!DOCTYPE html>        メッセージボディ
<html>
  :
</html>
```

HTTP　リクエスト

1. リクエスト行　：　メソド　パス　HTTP/バージョン
2. ヘッダ　：　サーバへ伝える制御情報　(ex.Host)
3. 空行
4. メッセージボディ

```
GET / HTTP/1.1    リクエスト行
```

```
Accept: image/gif, image/jpeg, */*      ヘッダ
Accept-Language: ja
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (Compatible; MSIE 6.0; Windows NT 5.1;)
Host: www.xxx.zzz
Connection: Keep-Alive
```

```
空行
```

```
メッセージボディ（POSTメソッドなどで使用）
```

HTTP　レスポンス

1. レスポンス行：HTTP/バージョン ステータス番号 補足メッセージ
2. ヘッダ
3. 空行
4. メッセージボディ

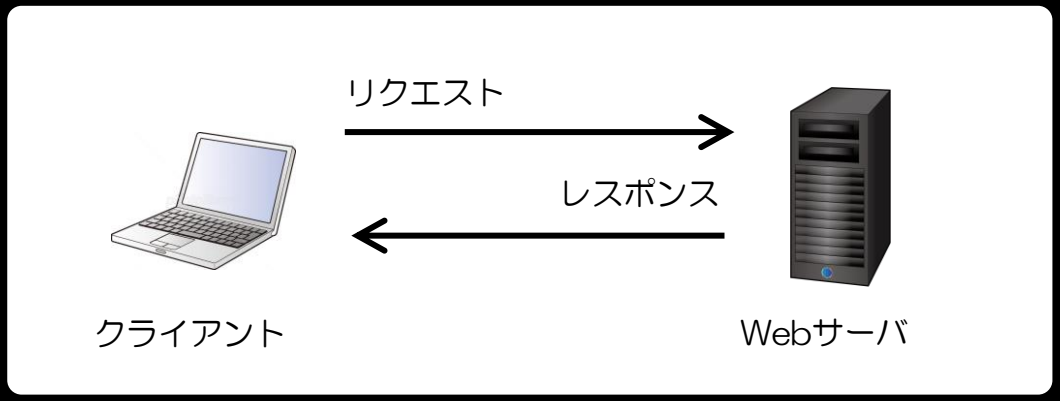
```
HTTP/1.1 200 OK     レスポンス行
```

```
Date: Sun, 11 Jan 2004 16:06:23 GMT      ヘッダ
Server: Apache/1.3.22 (Unix) (Red-Hat/Linux)
Last-Modified: Sun, 07 Dec 2003 12:34:18 GMT
ETag: "1dba6-131b-3fd31e4a"
Accept-Ranges: bytes
Content-Length: 4891
Keep-Alive: timeout=15, max=100
Connection: Keep-Alive
Content-Type: text/html
```

```
空行
```

```
<!DOCTYPE html>         メッセージボディ
<html>
  :
</html>
```

HTTP　メソッド



メソッド	概要
GET	リソースの取得
POST	子リソースの作成、リソースへのデータの追加、その他の処理
PUT	リソースの更新・作成
DELETE	リソースの削除
HEAD	リソースのヘッダの取得　(メタデータの取得)
OPTIONS	リソースがサポートしているメソッドの取得
TRACE	プロキシ動作の確認
CONNECT	プロキシ動作のトンネル接続への変更

実

HTTP　リクエスト/レスポンス

◆例

```
student01@e7ffae141a9e:/var/www/html$
student01@e7ffae141a9e:/var/www/html$ telnet localhost 80
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
GET /test.html HTTP/1.1
Host: localhost

HTTP/1.1 200 OK
Date: Fri, 19 Jun 2020 23:05:20 GMT
Server: Apache/2.4.41 (Ubuntu)
Last-Modified: Thu, 11 Jun 2020 00:20:49 GMT
ETag: "c6-5a7c3eeb73a40"
Accept-Ranges: bytes
Content-Length: 198
Vary: Accept-Encoding
Content-Type: text/html

<!DOCTYPE>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
<p>
HTMLのテストです。
うまく表示できくるかな？
</body>
</html>

Connection closed by foreign host.
student01@e7ffae141a9e:/var/www/html$
```

"GET　/test.html　HTTP/1.1" と入力してENTER

"Host: localhost" と入力して、ENTERを2回

実

HTTP　メソドの起動

1. ブラウザでのアクセス
2. リソースの取得
3. リンクのクリック
4. Formタグ

HTTP　メソド

1. GET
   - 指定したリソースを応答として返して欲しいというリクエスト。
   - リソースがあれば「200 OK」、無ければ「404 Not Found」というステータスを返却する。

2. HEAD
   - GETと似ているが、ヘッダ情報だけ返して欲しいというリクエスト。
   - リソースがあれば「200 OK」、無ければ「404 Not Found」というステータスを返却する。

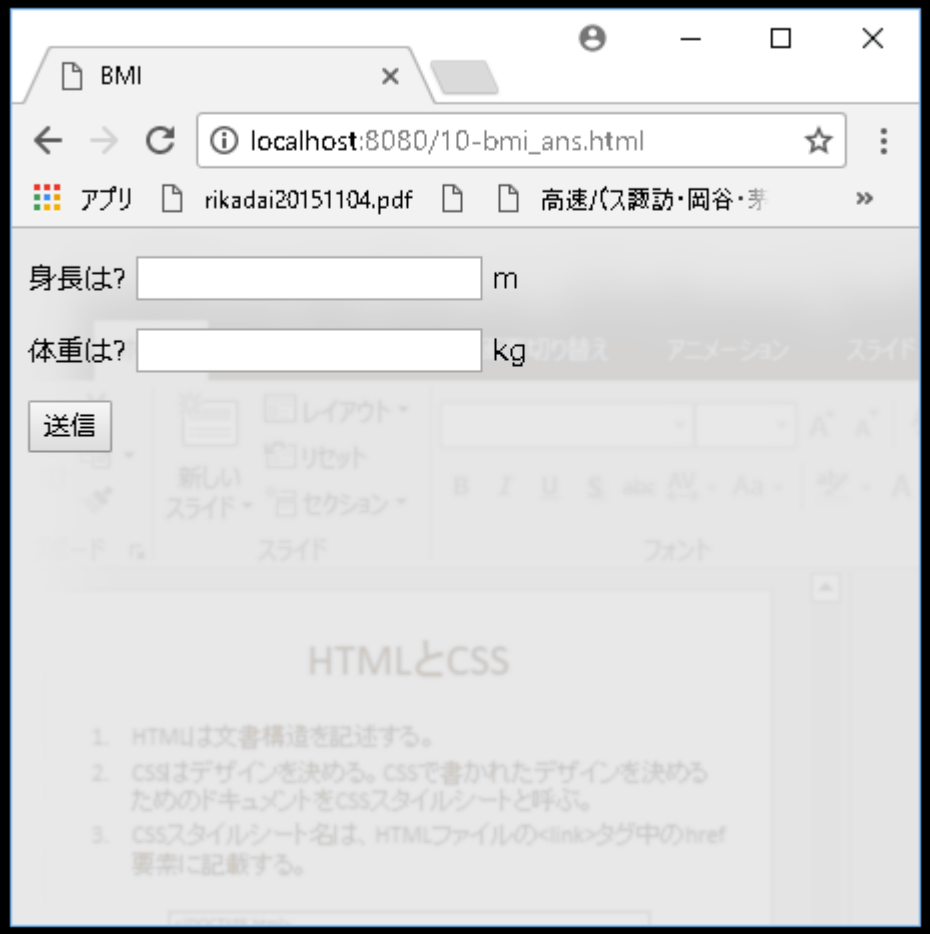
3. POST
   - Webサーバへのデータ送信。フォーム送信などに用いられる。

HTTP　メソド POST

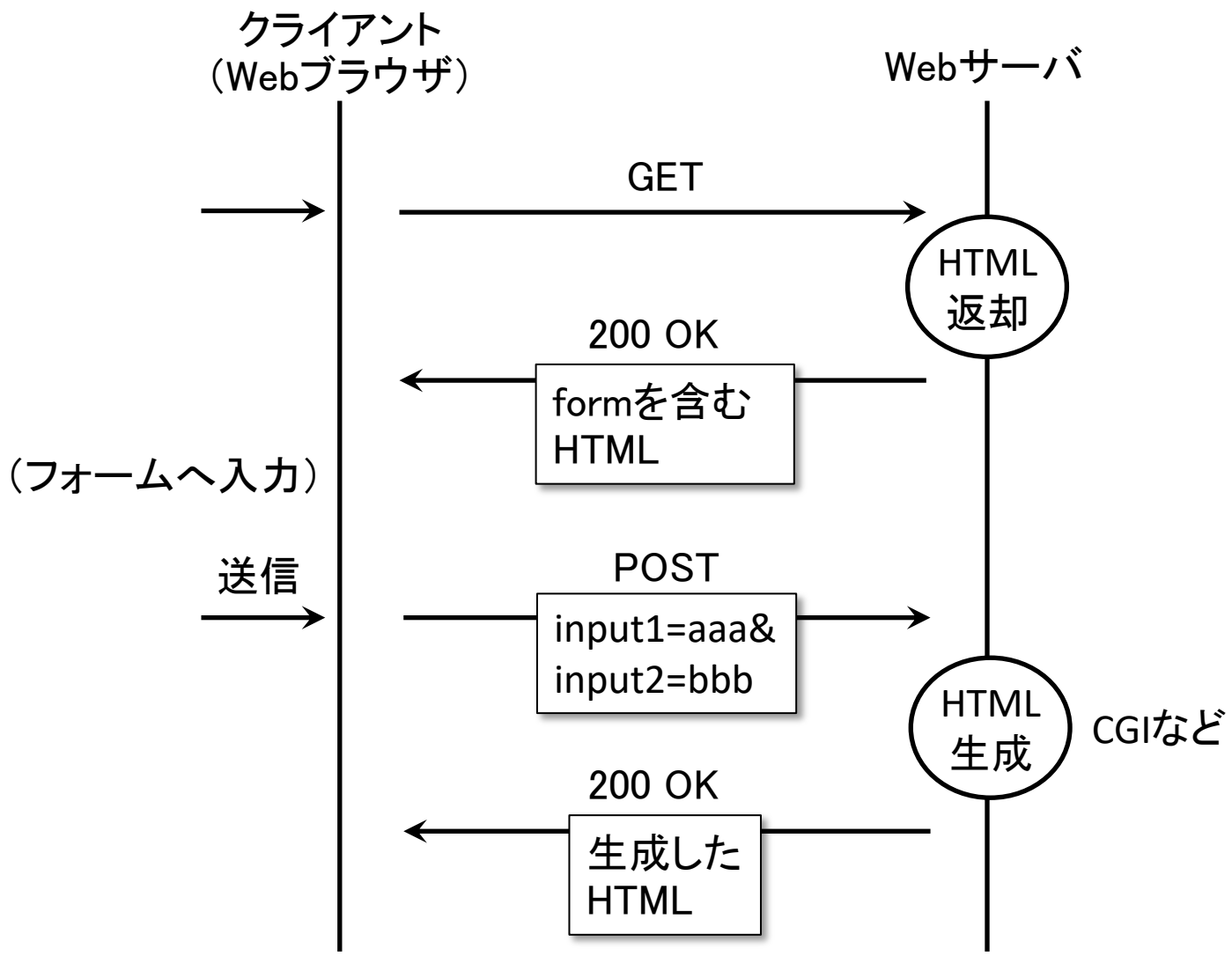
1. 検索して欲しい文字列、ID・パスワード、住所・氏名、などをWeb上で入力して送信する際にPOSTメソッドが使われる。
2. HTML上で<form>タグによりフォームを作成し、そのフォームに入力されたデータをWebサーバにPOSTで送る。
3. 下記のフォームの例では、「送信」というボタンを押すと、送信データの生成にはpost.phpというCGIプログラムが動き、送信にPOSTメソッドを使い、input1、input2というデータ名のデータが送信される。

```
<body>
<form action="post.php" method="POST  >
<input type="text" name="input1">
<input type="text" name="input2">
<input type="submit" value="送信">
</form>
</body>
```

HTTP　メソド POST(2)

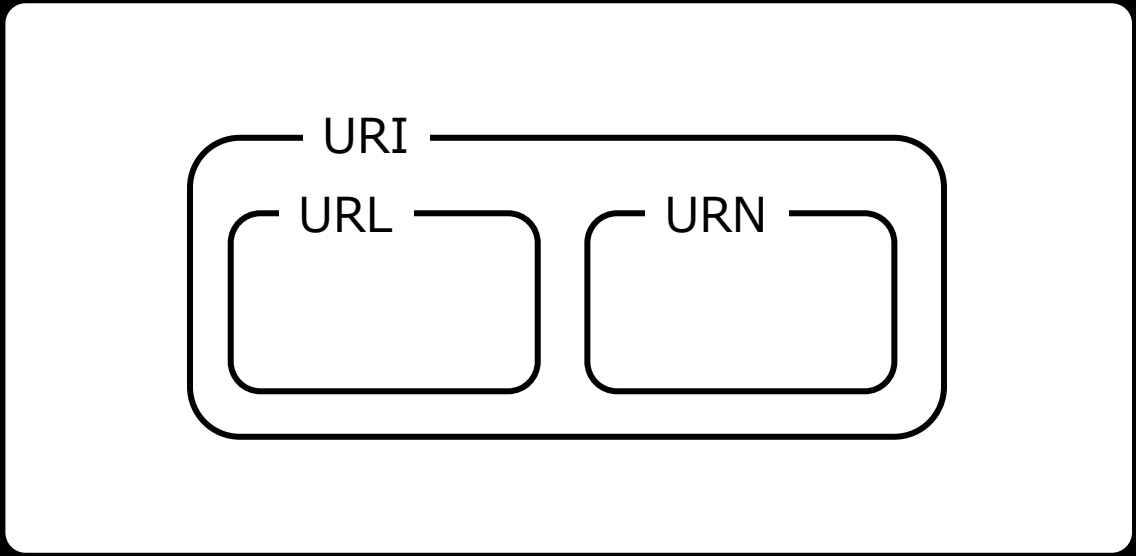


HTTP　処理の全体的な流れ



URI

1. URI : Uniform Resource Identifier



URL

1. URL　Uniform Resource Locator
2. HTTPにおけるURLの形式を下図に示す。
3. スキームとホスト名の間の「//」は、こう決めただけで特に意味はない。
4. パス名は本来必要なため、「http://www.google.com」ではなく「http://www.google.com/」が正しい。しかし、ブラウザやWebサーバが「/」を補完してくれるので特に問題ない。

スキーム　ホスト名　パス名

http://www.google.com/

スキーム　ホスト名　パス名

https://www.youtube.com/feed/trending

URLパラメータ

1. URLパラメータとは、サーバーに情報を送るためにURLに付け加える変数
2. クエッションマーク「?」をURLの末尾に付けることで、「変数（パラメータ）＝値」の形式でサーバーに送信するデータをURLに含めることが可能
3. アクティブパラメータとパッシブパラメータの2種類

```
https://wwww.example.jp/hogehoge/?category=hight
```

```
https://wwww.example.jp/hogehoge/?category=hight&category=lenght
```

HTML タグ

HTMLタグ　form

1. formタグは<form>～</form>で囲われるブロックレベル要素
2. 初めのタグ<form>内に、フォームを送信する先のプログラムや、送信するメソッド種別を、属性として指定する。

```
<form action="post.php" method="POST">
<input type="text" name="input1">
<input type="text" name="input2">
<input type="submit" value="送信">
</form>
```

HTMLタグ　input

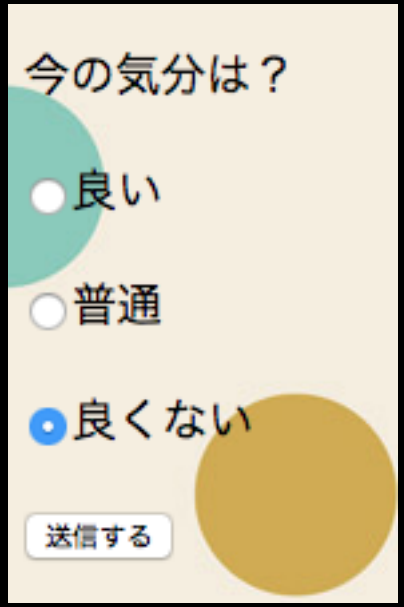
1. inputタグは、フォームで入力するデータを指定するインライン要素
2. データ入力方法として、text（文字列）、password（文字列だが画面表示しない）、checkbox（複数選択可）, radio（一つだけ選択）, file（ファイルを選択）が選べる。
3. name属性は送信したデータに付けられる名前。
4. ボタンの種別として、submit（送信）, reset（入力内容をキャンセル）, button（自由なボタン）, image（画像のボタン）が選べる。

```
<form action="post.php" method="POST">
<input type="text" name="input1">
<input type="text" name="input2">
<input type="submit" value="送信">
</form>
```

ラジオボタン

1. ラジオボタンは1つだけ選択可能な選択肢を提供する。

```
<p>今の気分は？</p>
<form action="/php/radio.php" method="POST">
<p><input type="radio" name="feel" value="良い">良い</p>
<p><input type="radio" name="feel" value="普通">普通</p>
<p><input type="radio" name="feel" value="良くない">良くない</p>
<input type="submit" value="送信する">
</form>
```



チェックボックス

1. チェックボックスは複数選択可能な選択肢を提供する。

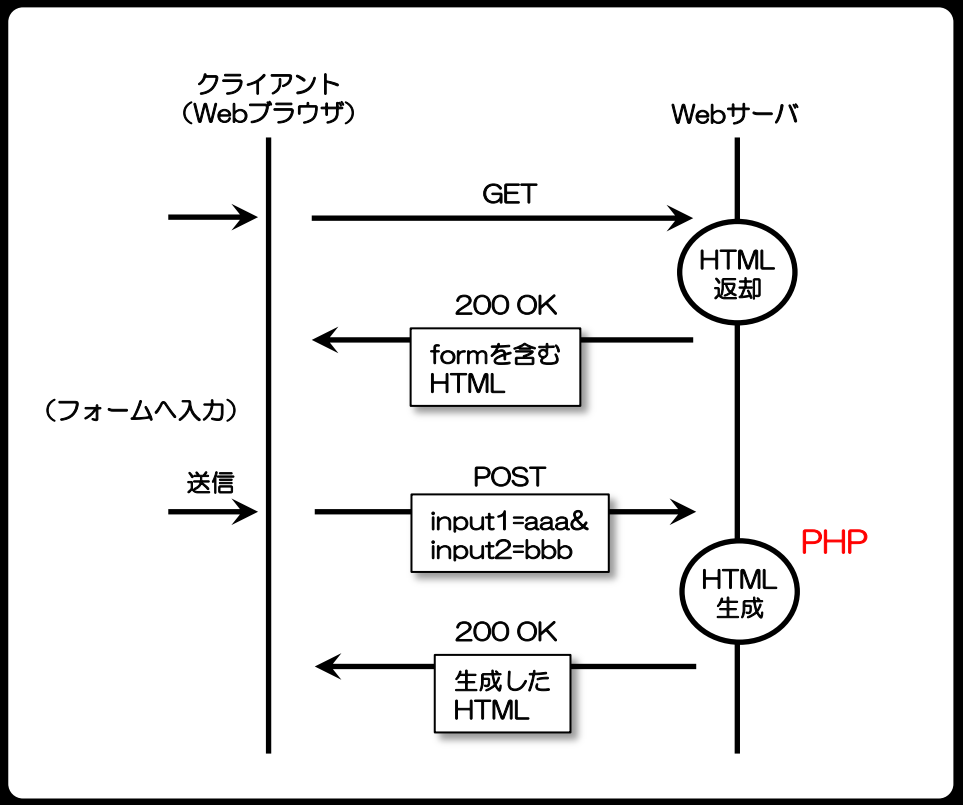
```
<p>好きな食べ物はある？</p>
<form action="/php/checkbox.php" method="POST">
<p><input type="checkbox" name="food[]" value="チョコ">チョコ</p>
<p><input type="checkbox" name="food[]" value="ピザ">ピザ</p>
<p><input type="checkbox" name="food[]" value="ラーメン">ラーメン</p>
<input type="submit" value="送信する">
</form>
```

PHPでの取り扱い

PHP 定義済み変数

1. PHP には定義済みの変数や定数が多く用意されており、スクリプトで使用することができる。

- 外部から来る変数
- 組み込みの環境変数
- 直近のエラーメッセージ
- 最後に取得したヘッダ
- その他



PHP 定義済み変数　スーパーグローバル

1. スーパーグローバル：スクリプト全体を通してすべてのスコープで使用可能な変数のこと。PHP の定義済み変数に含まれる。
2. 関数やメソッドの内部からアクセスする際にも global $variable; 等の記述不要

表　スーパーグローバル

変数	概要
$GLOBALS	スクリプトのグローバルスコープに現在定義されているすべての変数への参照を含む連想配列。 変数名が配列のキー
$_SERVER	ヘッダ、パス、スクリプトの位置のような 情報を有する配列。配列のエントリは、Web サーバーが生成
$_GET	URL パラメータで現在のスクリプトに渡された変数の連想配列
$_POST	HTTP POST メソッドから現在のスクリプトに渡された変数の連想配列
$_FILES	HTTP POST メソッドで現在のスクリプトにアップロードされた項目の連想配列
$_COOKIE	現在のスクリプトに HTTP クッキーから渡された変数の連想配列
$_SESSION	現在のスクリプトで使用できるセッション変数を含む連想配列
$_REQUEST	$_GET、 $_POST、$_COOKIE の内容をまとめた連想配列
$_ENV	環境変数として現在のスクリプトに渡された変数の連想配列

PHPによるPOST受信処理

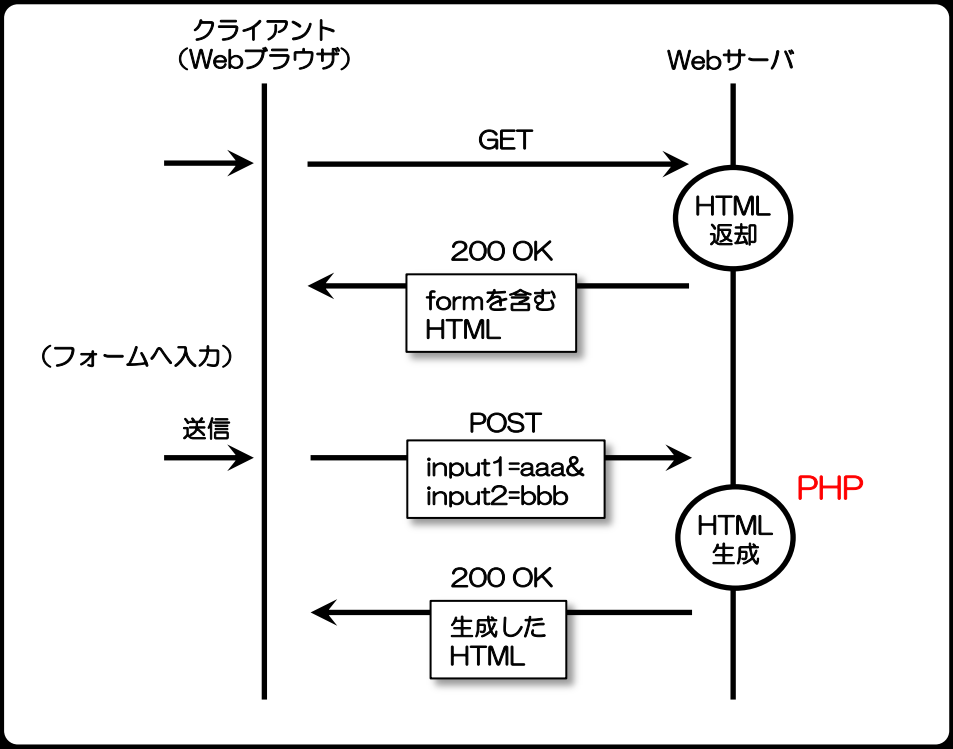
1. HTMLのフォームをPOSTメソッドで送信すると、CGIプログラムであるPHPプログラム上では各データが「$_POST」という名前の連想配列に格納される。

2. 先の例では、送信データ名がinput1とinput2の2つなので、以下の2つの連想配列にそれぞれのデータが格納される。

   $_POST['input1']

   $_POST['input2']

3. input1, input2というのは単にデータ名なので、HTMLファイルとCGIプログラム側で意識が合ってさえいれば任意の文字列を使える。
   なお、参考までに、GETメソッドで送信されたデータは$_GETという連想配列に格納される。



ラジオボタンの値の受信

1. $_POST['xxx']に選択結果が格納されている。

```
<p>今の気分は？</p>
<form action="/php/radio.php" method="POST">
<p><input type="radio" name="feel" value="良い">良い</p>
<p><input type="radio" name="feel" value="普通">普通</p>
<p><input type="radio" name="feel" value="良くない">良くない</p>
<input type="submit" value="送信する">
</form>
```

```
<?php
header('Content-Type: text/html; charset=UTF-8');
echo '<p>あなたは気分が' .$_POST[ 'feel' ] . "です。</p>¥n";
?>
```

チェックボックスの値の受信

1. $_POST['xxx']に配列として選択結果が格納されている。
   一つもチェックしなかった場合は配列が空なので注意。

```
if (empty($_POST['food'])) {
    echo "<p>あなたは好きなものがありません。</p>¥n";
    return;
}
foreach ($_POST['food'] as $food) {
    echo '<p>あなたは' . $food . "が好きです。</p>¥n";
}
```

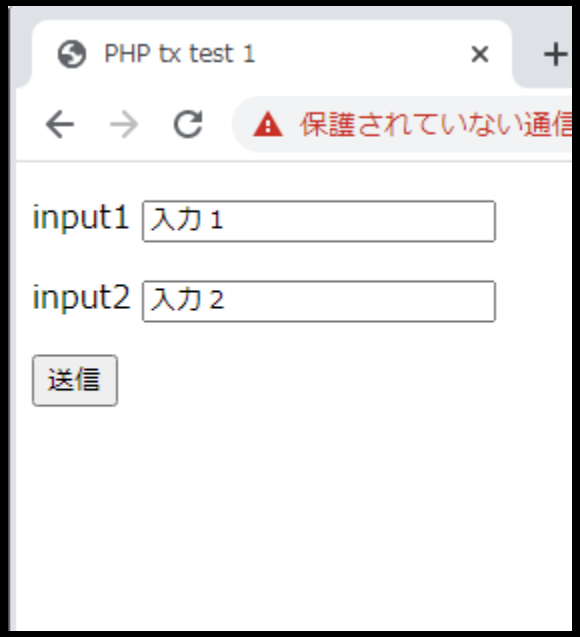
演習

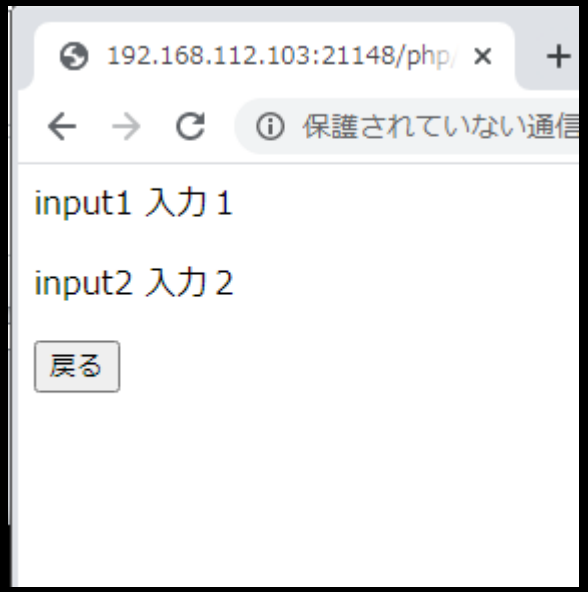
実

演習06-01　送信フォーム

1. HTMLのフォームを使って二つの入力のあるページを作成する。
2. PHPのCGIを呼び出して、POSTされた2つの文字列を表示する。

● 入力画面と表示画面をキャプチャし、SOLAに提出すること



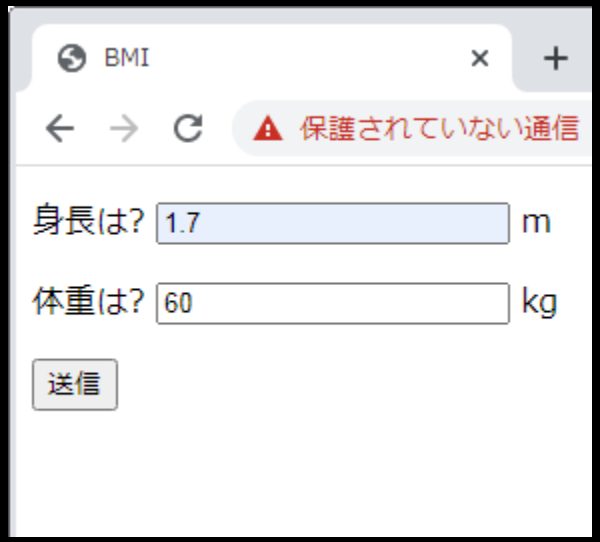


実

演習06-02　BMI算出

- 身長、体重を入力し、BMIを計算し出力する。
- 入力画面と表示画面をキャプチャし、SOLAに提出すること

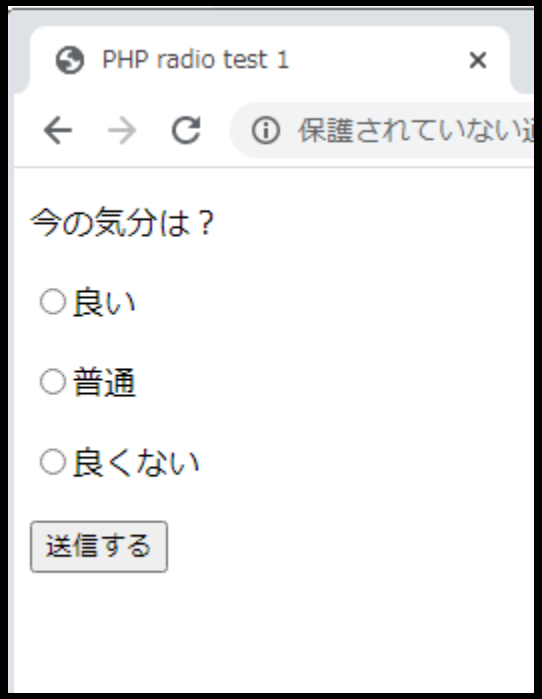
※BMI = 体重(kg) / 身長(m)^2

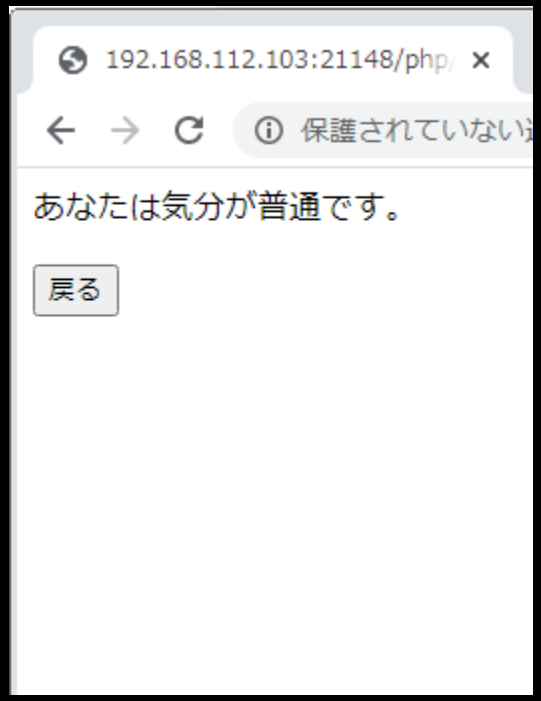


実

演習06-03 ラジオボタン

- ラジオボタンを用意し、選択した文字が表示されるページを作成する。
- 入力画面と表示画面をキャプチャし、SOLAに提出すること



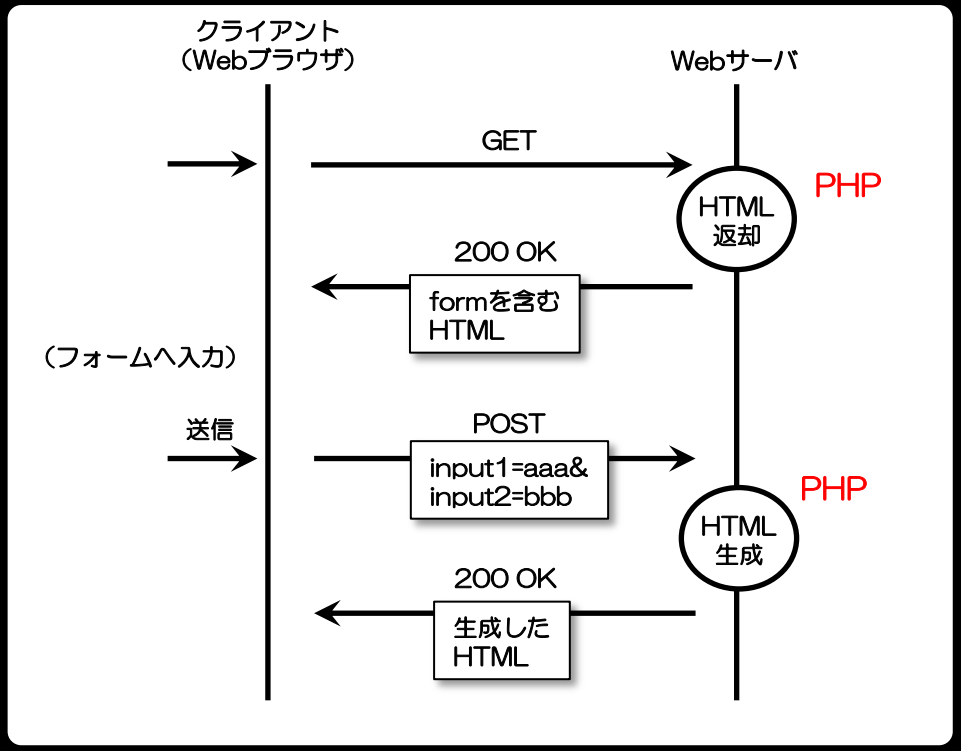


実

演習06-04　PHP

- 一つのページ(PHPのファイル)で演習06-03の内容を実現すること
- 作成したPHPファイルをSOLAに提出すること

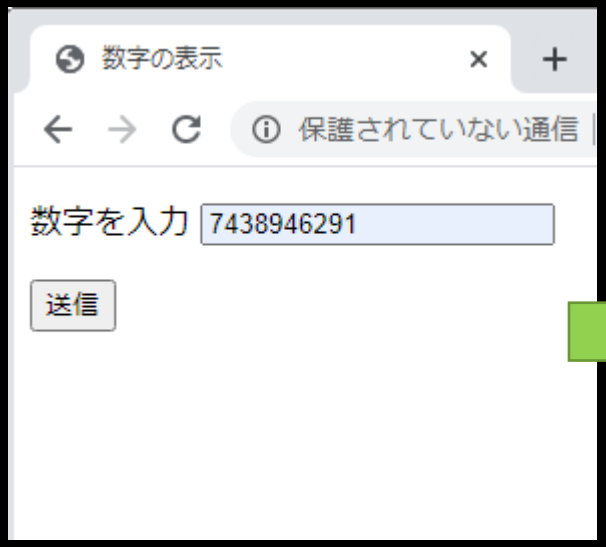
※メソッドGETでリクエストされた場合、$_POST[]の中身は空



実

宿題06-01　数字の表示

- 数字を入力したら、その数字を画像で表示する。
- 作成したファイル(php, html等)と入力画面と表示画面をキャプチャし、SOLAに提出すること
  ※数字の画像ファイルはSOLAにあります。



本日の内容のまとめ

1. HTTP
   - プロトコル概要
   - レイヤ構成
   - 基本シーケンス
   - メソド　種類、起動
2. POST
3. PHPでの取り扱い
4. 演習

おわり

サーバシステムプログラミング及び演習

― 第07回 MySQL ―

本日の内容

1. 初期設定
2. MySQLの操作
3. 演習

ファイルの準備

1. SOLAから下記のファイルをubuntuへダウンロード

   11-files.tar.gz

2. ダウンロードファイルのあるディレクトリで下記のコマンドを実施

   tar zxvf 11-files.tar.gz

3. PHPファイルを/var/www/html/php へ移動

4. HTMLファイルを/var/www/html へ移動

5. settings3.sql のあるディレクトリで下記のコマンドを実施

   mysql -udbuser -pdbuser < settings3.sql

実

MySQL設定

```
student02@8d0146a86745:~/share$
student02@8d0146a86745:~/share$
student02@8d0146a86745:~/share$ mysql -udbuser -pdbuser
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or ¥g.
Your MySQL connection id is 8
Server version: 5.7.24-0ubuntu0.18.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '¥h' for help. Type '¥c' to clear the current input statement.

mysql> select * from mydb.mytable;
+---------+
| counter |
+---------+
|       0 |
+---------+
1 row in set (0.00 sec)

mysql> select * from shop_db.list_t;
+------+--------------------------------------+--------------------------------------------+
| id_c | name_c                               | add_c                                      |
+------+--------------------------------------+--------------------------------------------+
|    1 | 料亭たなか                           | 東京都葛飾区何処町1-2                      |
|    2 | 洋風居酒屋アベリタ                   | 神奈川県横浜市中区何処町5-2                |
|    3 | 炭火焼肉田中屋                       | 広島県広島市何処町3-1                      |
|    4 | 無国籍料理トランス                   | 大阪府大阪市中央区何処町2-3                |
|    5 | 本格寿司あべ膳                       | 北海道札幌市何処町2-4                      |
|    6 | 台湾料理とらんす                     | 岩手県盛岡市何処町5-1                      |
|    7 | 甘味処ナルミ亭                       | 埼玉県さいたま市大宮区何処町4-1            |
|    8 | 南欧家庭料理タナカッターラ           | 佐賀県佐賀市何処町3-5                      |
|    9 | うまいものや阿部亭                   | 秋田県秋田市何処町4-3                      |
|   10 | インド料理トランース                 | 東京都葛飾区何処町1-6                      |
+------+--------------------------------------+--------------------------------------------+
10 rows in set (0.00 sec)
```

MySQL設定

```
mysql> use shop_db;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> desc list_t;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id_c   | int(11)     | NO   | PRI | NULL    | auto_increment |
| name_c | varchar(16) | YES  |     | NULL    |                |
| add_c  | varchar(32) | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> quit
Bye
student02@8d0146a86745:~/share$
```

実

【確認】　データベース設定

1. データベースに値を設定し、設定内容を確認する。
2. 確認時の画面をキャプチャしSOLAに提出する。

選択student01@23d66656fc10: /var/www/html/php

```
student01@23d66656fc10:/var/www/html/php$ mysql -udbuser -pdbuser
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or ¥g.
Your MySQL connection id is 19
Server version: 8.0.25-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '¥h' for help. Type '¥c' to clear the current input statement.

mysql> select * from shop_db.list_t;
+------+--------------------------------------+--------------------------------------------+
| id_c | name_c                               | add_c                                      |
+------+--------------------------------------+--------------------------------------------+
|    1 | 料亭たなか                               | 東京都葛飾区何処町1-2                                |
|    2 | 洋風居酒屋アベリタ                            | 神奈川県横浜市中区何処町5-2                              |
|    3 | 炭火焼肉田中屋                               | 広島県広島市何処町3-1                                |
|    4 | 無国籍料理トランス                            | 大阪府大阪市中央区何処町2-3                              |
|    5 | 本格寿司あべ膳                               | 北海道札幌市何処町2-4                                |
|    6 | 台湾料理とらんす                             | 岩手県盛岡市何処町5-1                                |
|    7 | 甘味処ナルミ亭                               | 埼玉県さいたま市大宮区何処町4-1                            |
|    8 | 南欧家庭料理タナカッターラ                        | 佐賀県佐賀市何処町3-5                                |
|    9 | うまいものや阿部亭                            | 秋田県秋田市何処町4-3                                |
|   10 | インド料理トランース                           | 東京都葛飾区何処町1-6                                |
+------+--------------------------------------+--------------------------------------------+
10 rows in set (0.00 sec)

mysql>
```

MySQL

リレーショナル・データベース

1. 2次元の表でデータを表すデータベース
2. 表全体をテーブルという
3. それぞれの列をフィールドという
4. それぞれの行をレコードという

テーブル（表）

フィールド（列）

顧客名簿

顧客ID	氏名	住所	年齢	e-mail
1	麻生 太郎	東京都渋谷区神山町	74	asou@ed.tus.ac.jp
2	小泉 進次郎	神奈川県横須賀市	38	koizumi@ed.tus.ac.jp
3	安倍 晋三	東京都渋谷区富ヶ丘	60	abe@ed.tus.ac.jp

レコード（行）

リレーション

1. リレーショナル・データベースでは、データベース中のテーブル間にリレーション（関係）を持たすことができる。

注文DB

注文ID	顧客ID	製品ID	数量	納期
1	3	11	1	2015年9月30日
2	1	3	38	2015年10月28日
3	2	8	60	2015年11月20日

製品DB

顧客DB

顧客ID	氏名	住所	年齢	e-mail
1	麻生 太郎	東京都渋谷区神山町	74	asou@ed.tus.ac.jp
2	小泉 進次郎	神奈川県横須賀市	38	koizumi@ed.tus.ac.jp
3	安倍 晋三	東京都渋谷区富ヶ丘	60	abe@ed.tus.ac.jp

SQL

1. SQLはリレーショナル・データベースを扱うための言語

2. 主に次の3種類の機能を持つ
   a. データ定義（データベースの生成、削除、変更）
   b. データ操作（データ挿入、更新、削除、検索）
   c. データ制御（データベース操作権限、データベース一貫性の制御）

3. SQLは標準化されてはいるものの、リレーショナルデータベースの種類によって若干の違いがある。

SQLの実行方法

1. 主に次の3種類の方法がある

   a. リレーショナルデータベースに直接SQL命令を発行する。リレーショナルデータベースに付属のコマンドラインユーティリティーを経由してSQL命令を実行させられる。MySQLの場合、mysqlというのはコマンドラインユーティリティ。

   b. 高級言語で書いたプログラム中にSQL命令を書いておき、ツールで変換してSQL命令を発行するようなプログラムを生成する（埋め込みSQL）。

   c. ライブラリとして、プログラム中からアクセスできるAPIを用意し、API経由でSQL命令を発行する。プログラム中でSQLを実行する場合は、現在はこちらが主流。

SQL構文：データベース操作（1）

1. データベース生成

   構文: create␣database␣<データベース名>␣[<オプション>]

   例: create database mydb default character set utf8

   （mydbというデータベースを文字コードをutf8で作成）

2. データベース削除

   構文: drop␣database␣<データベース名>

   例: drop database mydb

   （mydbというデータベースを削除）

3. データベース構成変更

   構文: alter␣database␣<データベース名>␣[<オプション>]

   例: alter database mydb default character set utf8

   （既にあるmydbというデータベースの文字コードをutf8に変更）

SQL構文：データベース操作（2）

4. データベース一覧

構文: show␣databases

例: show databases

（存在するデータベース名を全て表示する）

5. データベース選択

構文: use␣<データベース名>

例: use mydb

（以降、特にデータベース名を指定しない場合は、mydbが操作対象と判断される）

SQL構文：テーブル操作（1）

1. テーブル生成

   構文: create␣table␣<テーブル名>␣[<オプション>]

   例: create table mytable (counter int)

   （整数のcounterというフィールドを持つmytableというテーブルを生成）

2. テーブル削除

   構文: drop␣table␣<テーブル名>

   例: drop table mytable

   （mytableというテーブルを削除）

3. テーブル一覧

   構文: show␣tables

   例: show tables

   （データベース中のテーブル名をすべて表示する）

SQL構文：テーブル操作（2）

4.　テーブル構成参照

構文: desc␣<テーブル名>

例: desc mytable

（テーブル内のフィールドを一覧）

SQL構文：レコード操作

1. レコード追加

構文: insert␣into␣<テーブル名>␣(<フィールド名>)␣values␣(<値>)

例: insert into mytable (counter) values (0)

（mytableというテーブルに、counterというフィールドの値が0のレコードを追加）

2. レコード検索

構文: select␣<フィールド名>␣from␣<テーブル名>

例: select counter from mytable;

（mytableというテーブル中の、counterというフィールドを検索して返す）

3. レコード更新

構文: update␣<テーブル名>␣set␣<フィールド名>=<値>

例: update mytable set counter=100

（mytableというテーブル中の、counterというフィールドを全て100にする）

MySQL　コマンドライン操作

1. mySQLに直接SQL命令を発行するためには、コマンドラインツールであるmysqlを使う。

2. mysqlの使い方

   構文: mysql␣-u<ユーザ名>␣-p<パスワード>␣[<オプション>]

   例: mysql　-uroot　-psecret

   （ユーザ名root、パスワードsecretを使って接続）

3. mysqlでのコマンド投入方法

   SQL命令入力後にセミコロン「;」を入力してエンターを押す。

   セミコロンが必要なのは、長いSQL命令を複数行に渡って入力できるようにという配慮。

   セミコロンを入れ忘れてエンターを押した場合は、次の行でセミコロンだけを入力してエンターを押す。

データベース操作（1）

1. データベース参照
   - mysql -udbuser -pdbuser
   - mysql> show databases;

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.00 sec)

mysql>
```

2. テーブル参照
   - mysql> use mydb;
   - mysql> show tables;

```
mysql> use mydb
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+----------------+
| Tables_in_mydb |
+----------------+
| mytable        |
+----------------+
1 row in set (0.01 sec)

mysql>
```

データベース操作（2）

3. テーブルの内容を表示
   - mysql> desc mytable;

```
mysql> desc mytable;
+---------+---------+------+-----+---------+-------+
| Field   | Type    | Null | Key | Default | Extra |
+---------+---------+------+-----+---------+-------+
| counter | int(11) | YES  |     | NULL    |       |
+---------+---------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql>
```

4. データの値を表示
   - mysql> select counter from mytable;

```
mysql> select counter from mytable;
+---------+
| counter |
+---------+
|       0 |
+---------+
1 row in set (0.00 sec)

mysql>
```

データベース操作（3）

5. データの値を変更
   - mysql> update mytable set counter=100;

6. データの値を+1する
   - mysql> update mytable set counter=counter+1;

7. データの値を表示
   - mysql> select counter from mytable;

```
mysql> select counter from mytable;
+---------+
| counter |
+---------+
|     101 |
+---------+
1 row in set (0.00 sec)

mysql>
```

データベース操作（4）

検索

- mysql> select * from list_t where idc_c = 1

あいまい検索

- mysql> select * from list_t where name_c like ‘%たな%’ and add_c like ‘%東京%’

挿入

- mysql> insert into <table> set <フィールド名>=<値>, <フィールド名>=<値>,

PHPからのデータベースアクセス

1. PHPには、データベース（MySQL）にアクセスする関数が用意されている。

2. 旧来型の記法と、新しいオブジェクト指向の記法がある。ここでは、より簡便な（短く書ける）オブジェクト指向記法のmysqliを使うことにする。

3. おおまかな手順
   ① MySQLとのコネクションを張る。パラメータは、サーバ名、ユーザ名、パスワード、データベース名の4つ。
   ② コネクションに対し、SQL命令を発行する。これに対して結果が帰ってくる。
   ③ 結果の中身を見る。
   ④ コネクションを閉じる。

プログラム概要

① MySQLとのコネクションを張る。

```
$conn = new mysqli('localhost', ‘dbuser', ‘dbuser', 'mydb');
if ($conn->connect_errno) {
    die('Connection failed' . $conn->connect_error);
}
echo "<p>Connection succeeded</p>¥n";
```

② コネクションに対し、SQL命令を発行する。

```
$sql = 'select counter from mytable';
$result = $conn->query($sql);
if (!$result) {
    die('Query failed' . $conn->error);
}
echo "<p>Query succeeded</p>¥n";
```

③ 結果の中身を見る。

```
$counter = $result->fetch_object()->counter;
echo '<p>counter=' . $counter . "</p>¥n";
```

④ コネクションを閉じる。

```
$conn->close();
```

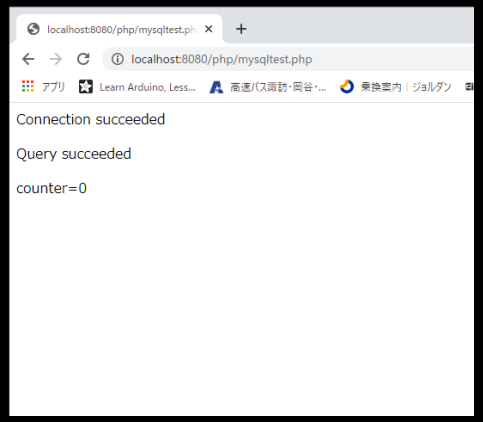
実

演習07-01　データベース参照

1. PHPのプログラムを作成し、PHPを通して、MySQLの値を参照する。
2. 下記の図で示すDB内のcounterの値を取得し、WEBに表示する。

● 作成したプログラムと表示画面のキャプチャファイルをSOLAに提出すること

```
root@23d0b0b0rc10:/var/Windows/11-11res# mysql -udbuser -pdbuser
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or ¥g.
Your MySQL connection id is 14
Server version: 8.0.25-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '¥h' for help. Type '¥c' to clear the current input statement.

mysql> desc mydb.mytable;
+---------+------+------+-----+---------+-------+
| Field   | Type | Null | Key | Default | Extra |
+---------+------+------+-----+---------+-------+
| counter | int  | YES  |     | NULL    |       |
+---------+------+------+-----+---------+-------+
1 row in set (0.00 sec)

mysql>
```



実

CLIからのスクリプト実行

1. 11-mysql.phpをコマンドラインで実行する。Ubuntuの端末画面から以下のように実行。
   ① php /var/www/html/php/XXX.php

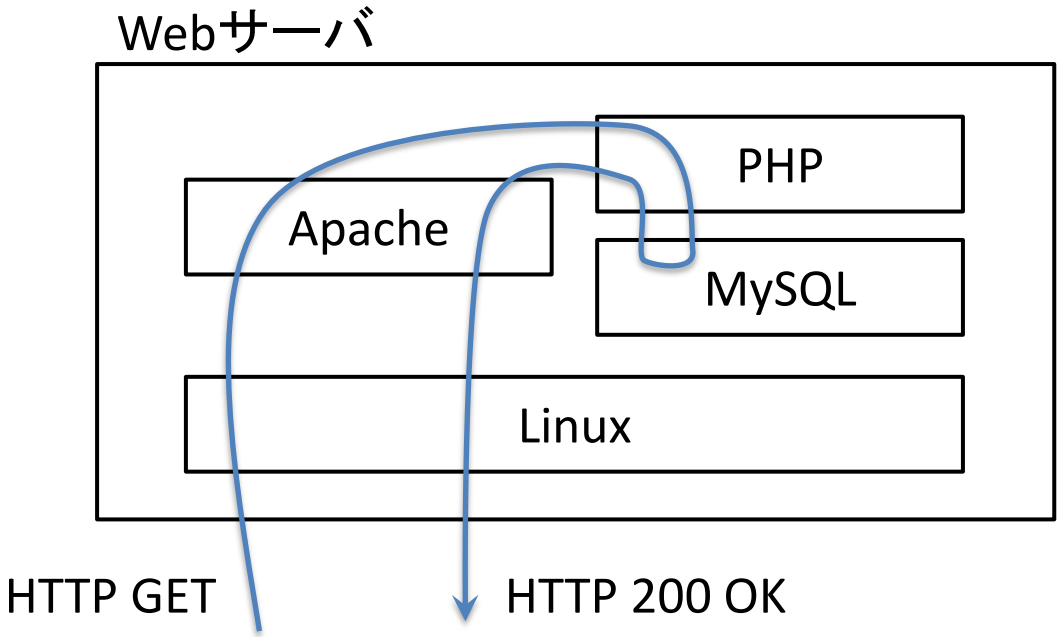
2. 以下のように表示される。

```
<p>Connection succeeded</p>
<p>Query succeeded</p>
<p>counter=0</p>
```

実

エラー出力

1. XXX.php にブラウザからアクセス。
2. counterの値が表示されれば成功。
3. Apacheのログは正常時は/var/log/apache2/access.log、エラー時は/var/log/apache2/error.logに記録される。端末アプリで、sudo tail -f /var/log/apache2/error.logのように監視する。



データベース更新

1. データベース参照の際は、以下のSQL命令を発行

   select counter from mytable

2. データベース中のcounterフィールドの値を、0にするには、以下の様なSQL命令を発行

   update mytable set counter=0

3. データベース中のcounterフィールドの値を、+1するには、以下の様なSQL命令を発行

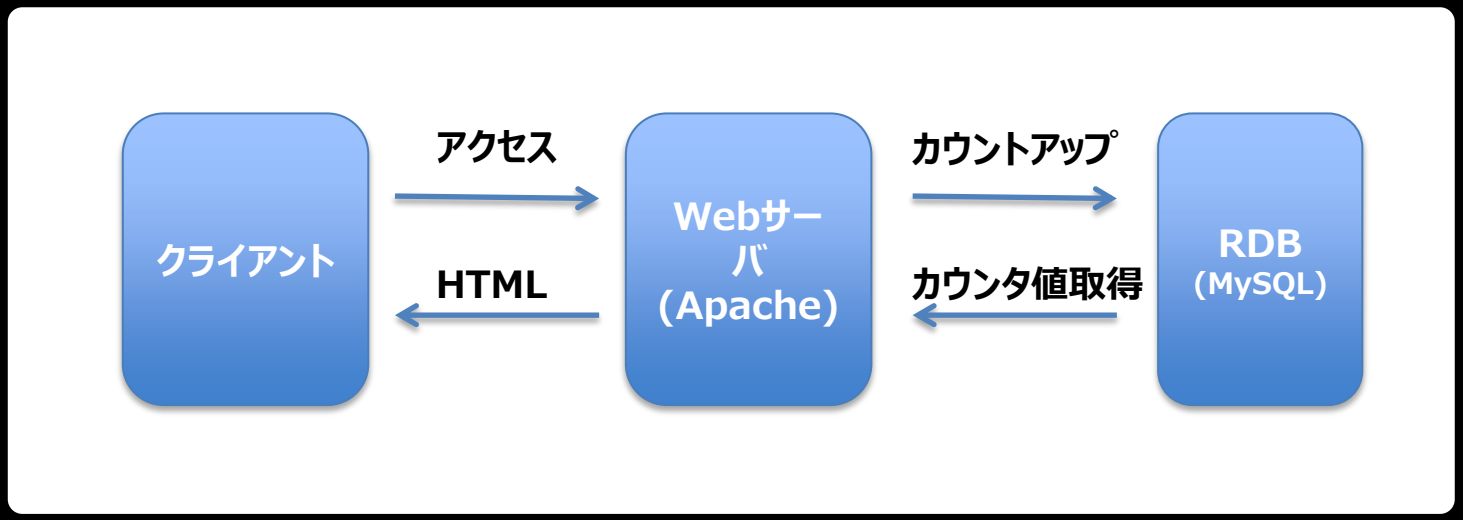
   update mytable set counter=counter+1

実

演習07-02　カウンタ

1. DB内のcounterを用いて、作成したページにアクセスした回数をカウントするPHPのプログラムを作成する。
2. アクセスした回数をcounterに書き込み、「あなたはxx人目のお客様です。」と表示する

● 作成したプログラムと表示画面のキャプチャファイルをSOLAに提出すること



実

演習07-03　カウンタ(2)

1. DB内のcounterを用いて、作成したページにアクセスした回数をカウントするPHPのプログラムを作成する。
2. アクセスした回数をcounterに書き込み、「あなたはxx人目のお客様です。」と表示する。
3. 表示する数字は前回の授業で用いた画像を用いて表示する。

● 作成したプログラムと表示画面のキャプチャファイルをSOLAに提出すること

あなたは 15 人目のお客様です

ファイルの準備

1. PHPファイルを/var/www/html/php へ移動

2. HTMLファイルを/var/www/html　へ移動

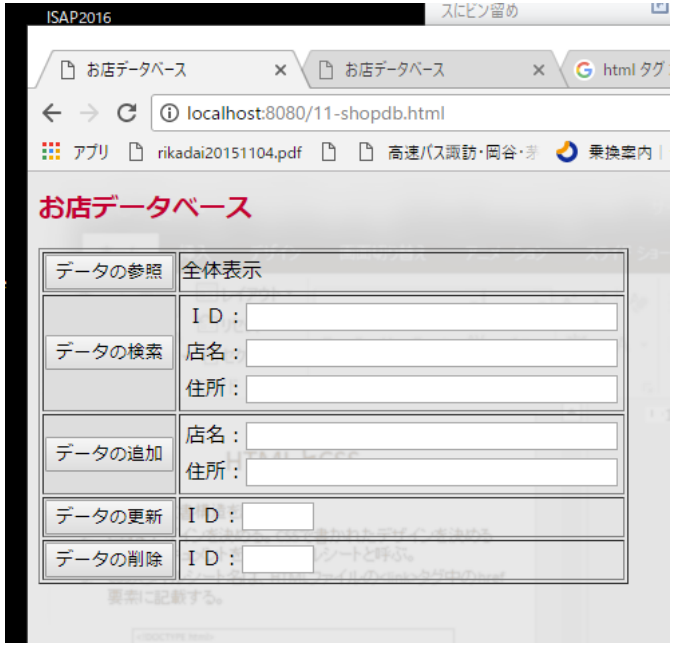
演習準備　ファイル構成等

◆店名と住所を管理するシステム

◆外観　11-shopdb.html

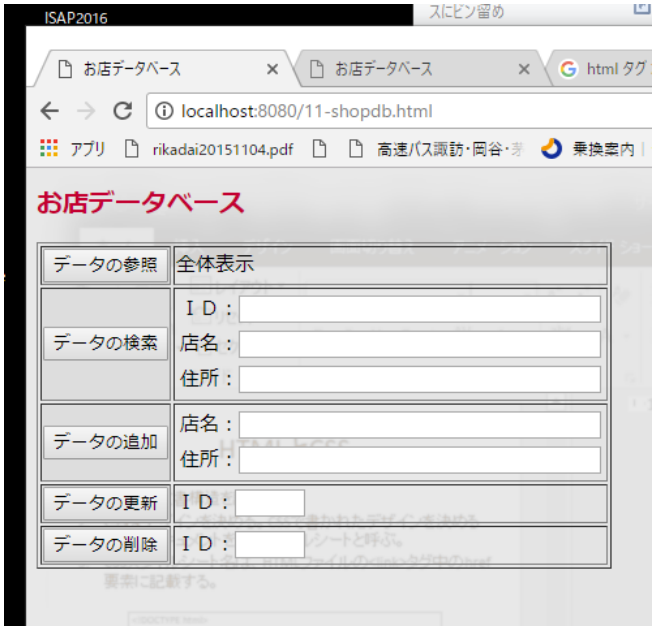
◆ファイル構成

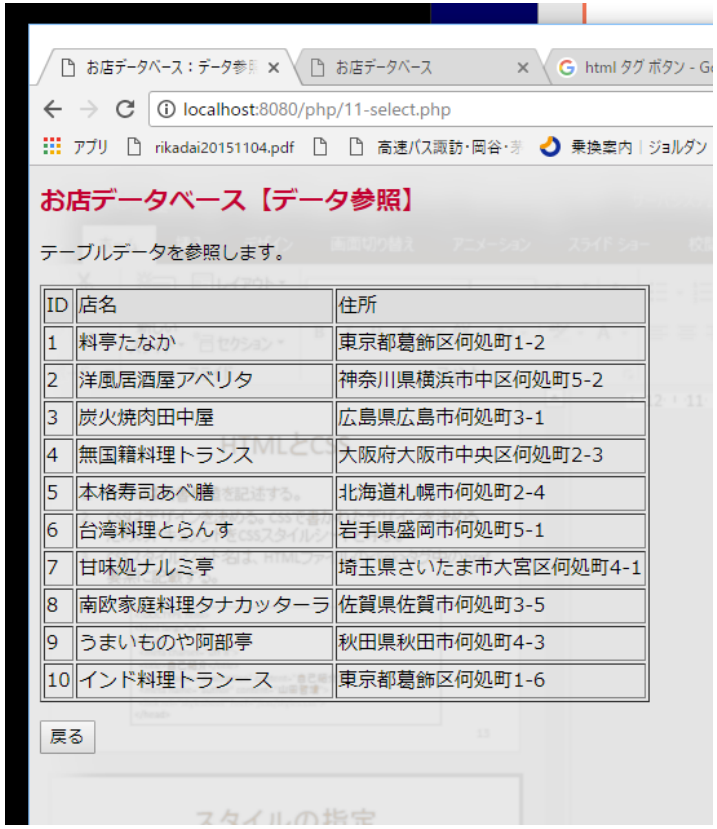
機能	ファイル
メイン	11-shopdb.html
データの参照	11-select.php
データの検索	11-search.php
データの追加	11-insert.php
データの変更	11-update.php ⇒ 11-update_data.php
データの削除	11-delete.php ⇒ 11-delete_data.php



演習準備　データの参照

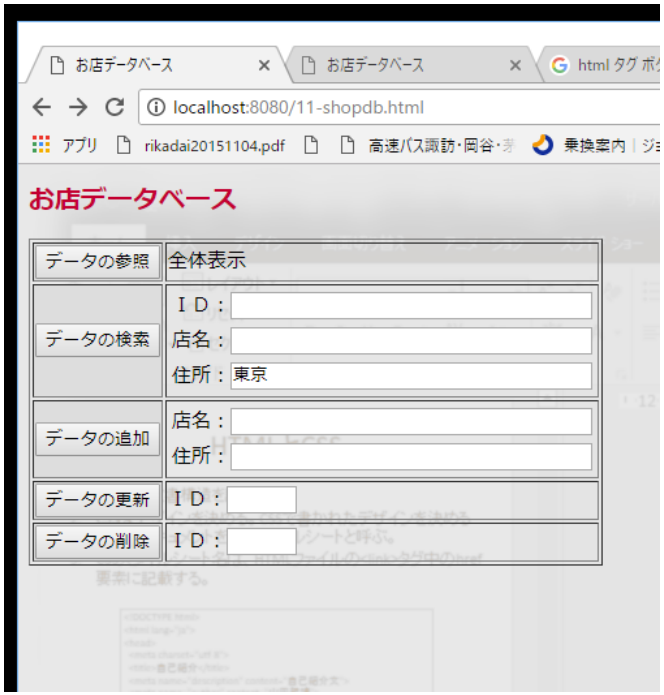
◆データベースのデータを表示

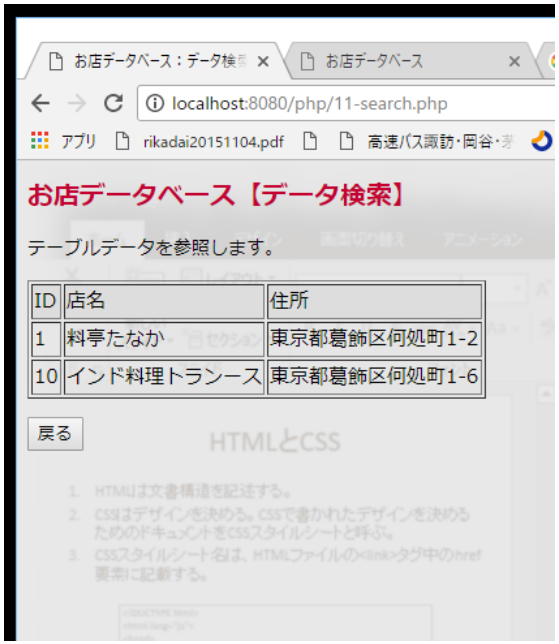




演習準備　データの検索

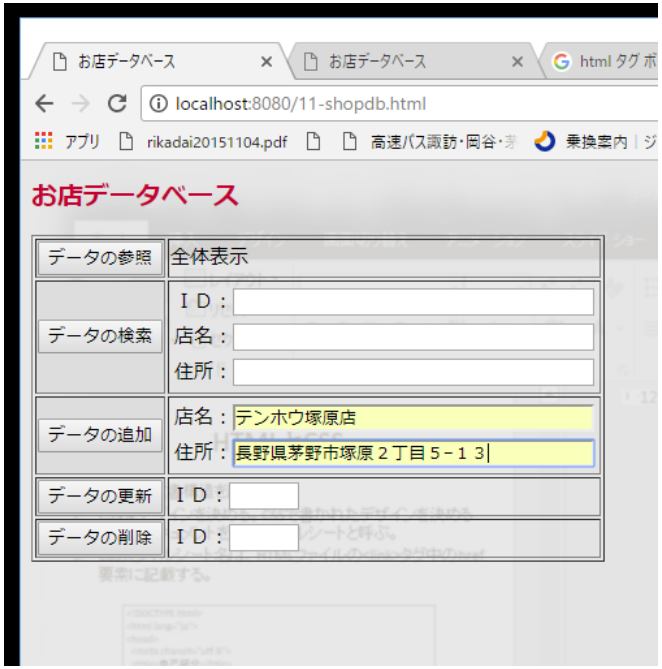
◆指定のデータを表示

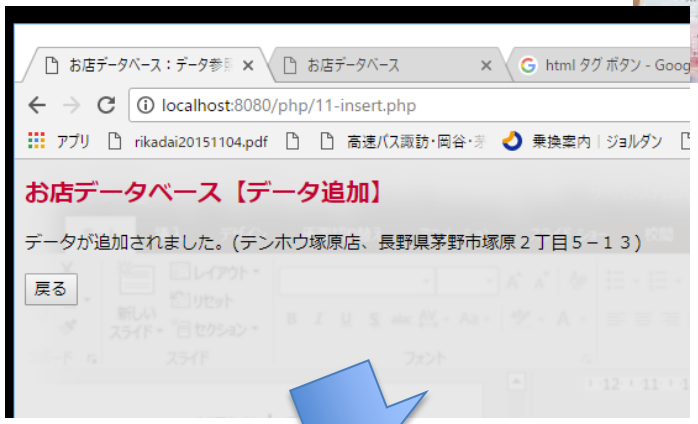


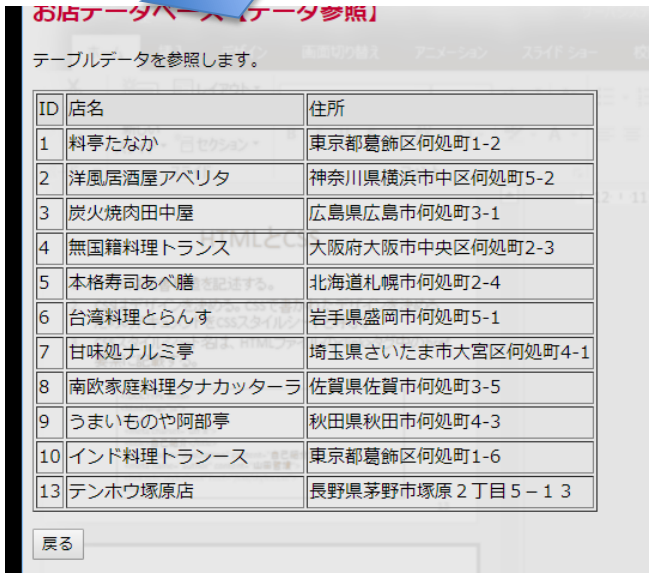


演習準備　データの追加

◆データを追加

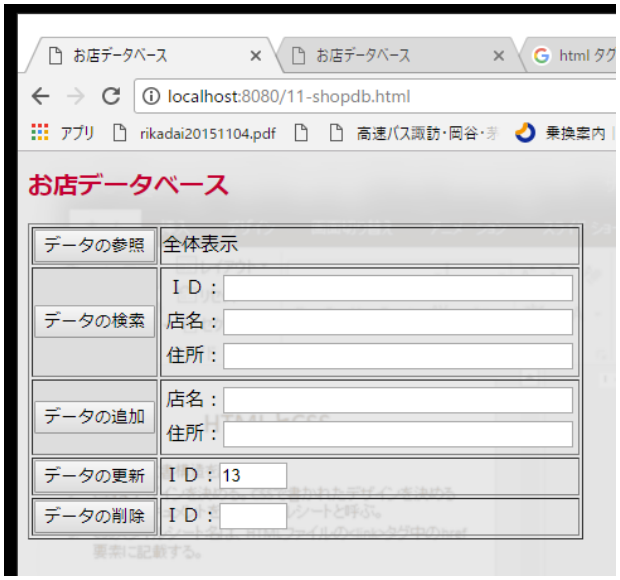


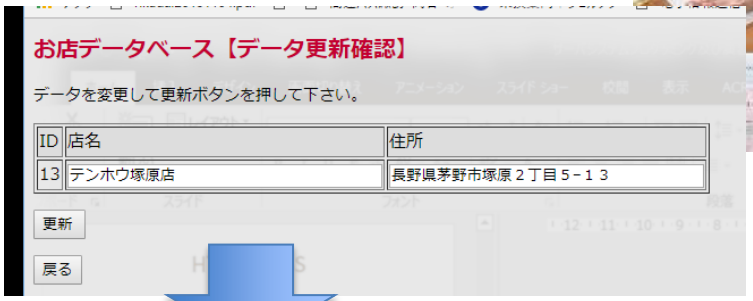


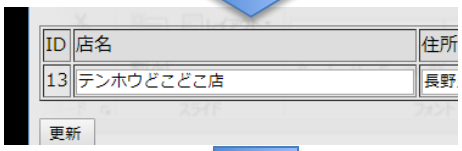


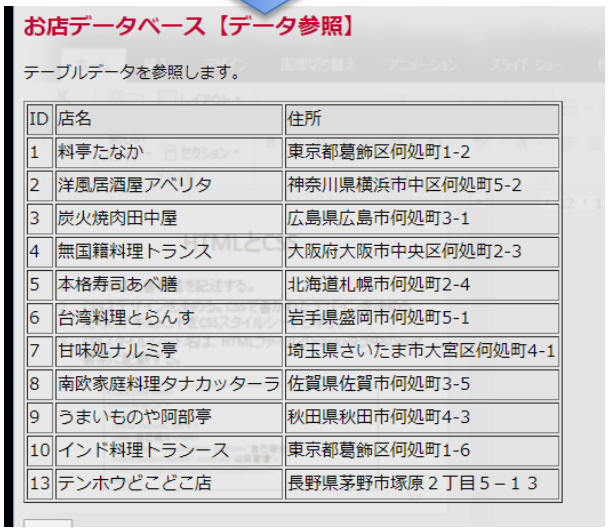
演習準備　データの変更

◆データを変更



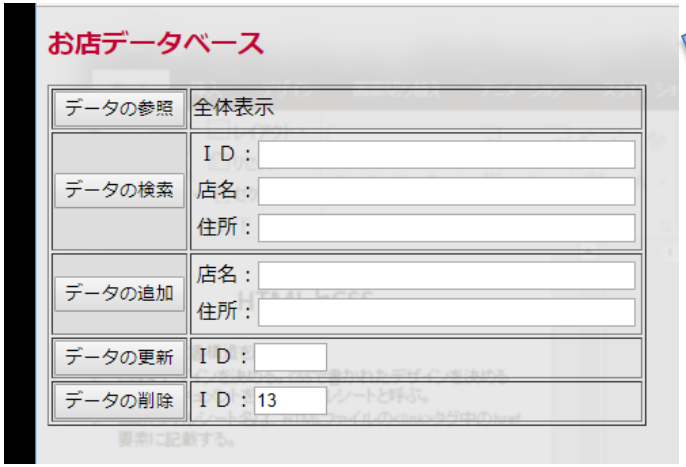


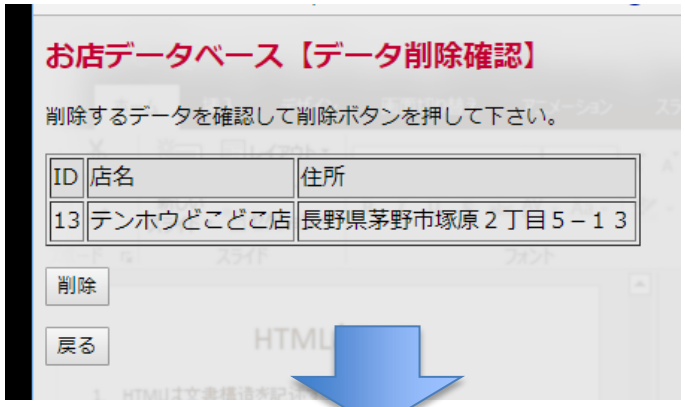


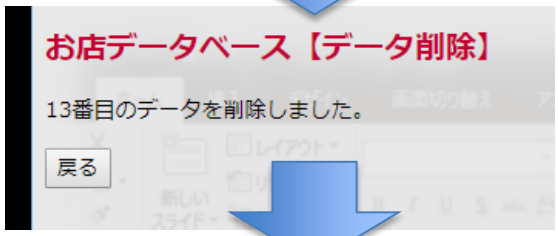


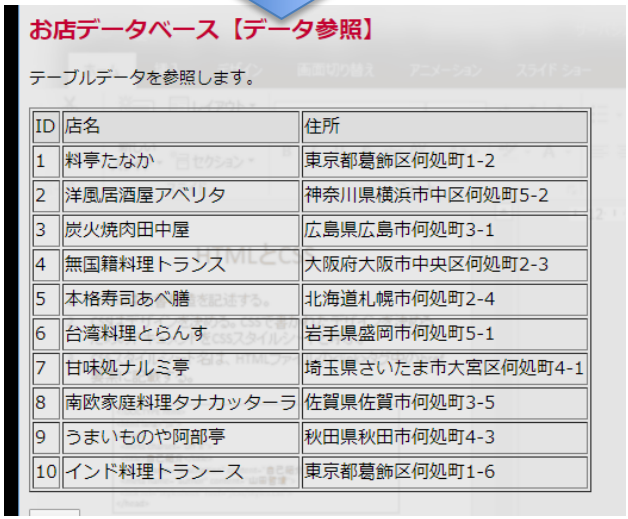
演習準備　データの削除

◆データを削除









演習準備　11-shopdb.html

```
<!DOCTYPE html>
<html lang="ja">

<head>
<meta charset="utf-8">
<title>お店データベース</title>
<meta name="description" content="shop data base">
<meta name="author" content="理科大太郎">
<link rel="stylesheet" href="/css/styles.css">
</head>
```

（つづく）

```
<body>

<h1>お店データベース</h1>
<table border="1">


    <!----------------------------------------
    *
    *  データの参照
    *
    ----------------------------------------->
    <tr>
        <form action="/php/11-select.php" method="post">
        <td> <input type="submit" value="データの参照"> </td>
        <td>全体表示</td>
        </form>
    </tr>
```

（つづく）

演習準備　11-shopdb.html

```
<!----------------------------------------
*
* データの検索
*
----------------------------------------->
<tr>
    <form action="/php/11-search.php" method="post">
    <td> <input type="submit" value="データの検索"> </td>
    <td>
    <table border="0"><tr><td>
    ＩＤ：<input type="text" name="id" size="35">
    </tr></td> <tr><td>
    店名：<input type="text" name="name" size="35">
    </tr></td> <tr><td>
    住所：<input type="text" name="add" size="35">
    </tr></td></table>
    </td>
    </form>
</tr>
```

（つづく）

```
<!----------------------------------------
    *
    * データの追加
    *
    ----------------------------------------->
  <!----------------------------------------
    *
    * データの更新
    *
    ----------------------------------------->
  <!----------------------------------------
    *
    * データの削除
    *
    ----------------------------------------->
</table>
</body>
</html>
```

演習07-04　結果提出(1)

データを更新するためのファイル11−update.phpには、ポストデータを受け取るコード、MySQLへ送るクエリを作成するコード、が抜けています。

正しいコードを追記し更新が出来るようにして下さい。

更新前、更新後のデータ表示画面をキャプチャ

キャプチャファイル、PHPファイルを提出

形式

・キャプチャファイル：JPEGまたはPNG形式

・PHP：　PHPファイル

提出方法

SOLAに提出

演習07-05　結果提出(2)

データを検索するためのファイル11−search.phpには、ポストデータを受け取るコードが抜けています。正しいコードを追記し更新が出来るようにして下さい。検索結果画面をキャプチャ

キャプチャファイル、PHPファイルを提出

形式

・キャプチャファイル：JPEGまたはPNG形式

・PHP：　PHPファイル

提出方法

SOLAに提出

演習07-06　結果提出(3)

課題

データを追加するためのファイル11−insert.phpには、ヘッダ部分のコードしか記載されていません。

正しくデータが追加できるよう必要なコードを追記して下さい。追加前、追加後のデータ表示画面をキャプチャし、キャプチャファイルと作成したコードを提出

形式

・キャプチャファイル：JPEGまたはPNG形式

・PHP：　PHPファイル

提出方法

SOLAに提出

演習07-07　結果提出

counterをゼロにリセットするc−reset.phpを作成する。c−reset.phpを実行することでアクセスカウンタをリセットする。11−gcounter.phpにアクセスし「あなたは1人目のお客様です」と表示された画面キャプチャーを取得する。PHPファイルとキャプチャファイルを提出

形式

・キャプチャファイル：JPEGまたはPNG形式

・PHP：　PHPファイル

提出方法

SOLAに提出

サーバシステムプログラミング及び演習

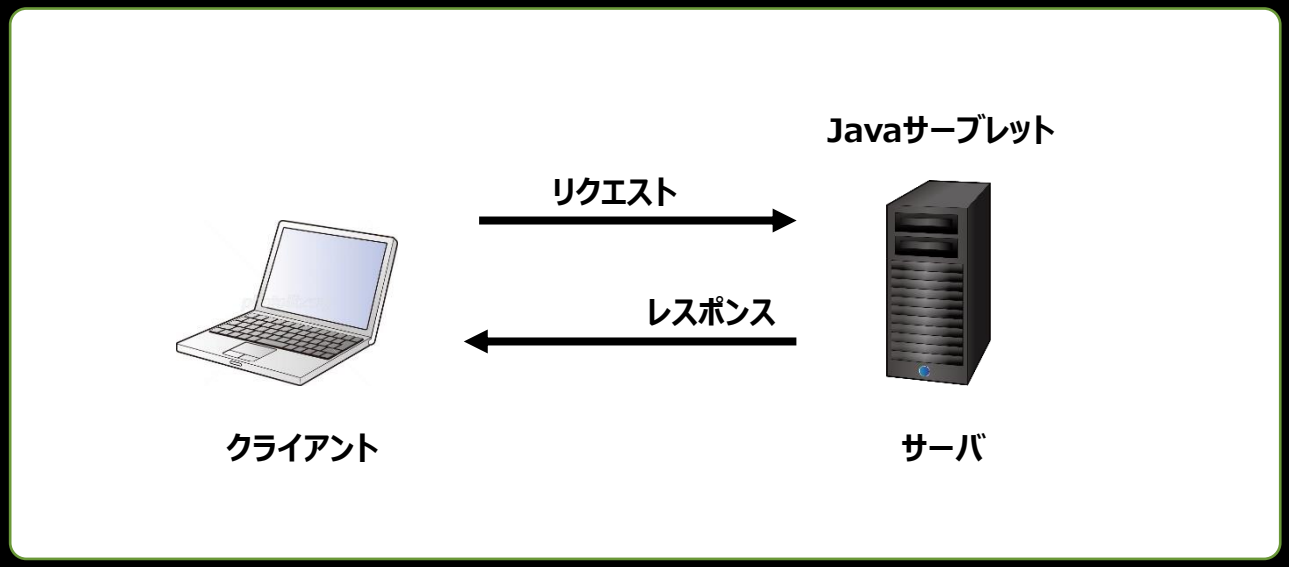
– 第8回 JSP –

今回の学習内容

1. サーブレット
2. サーブレットコンテナ　Tomcat
3. JSP

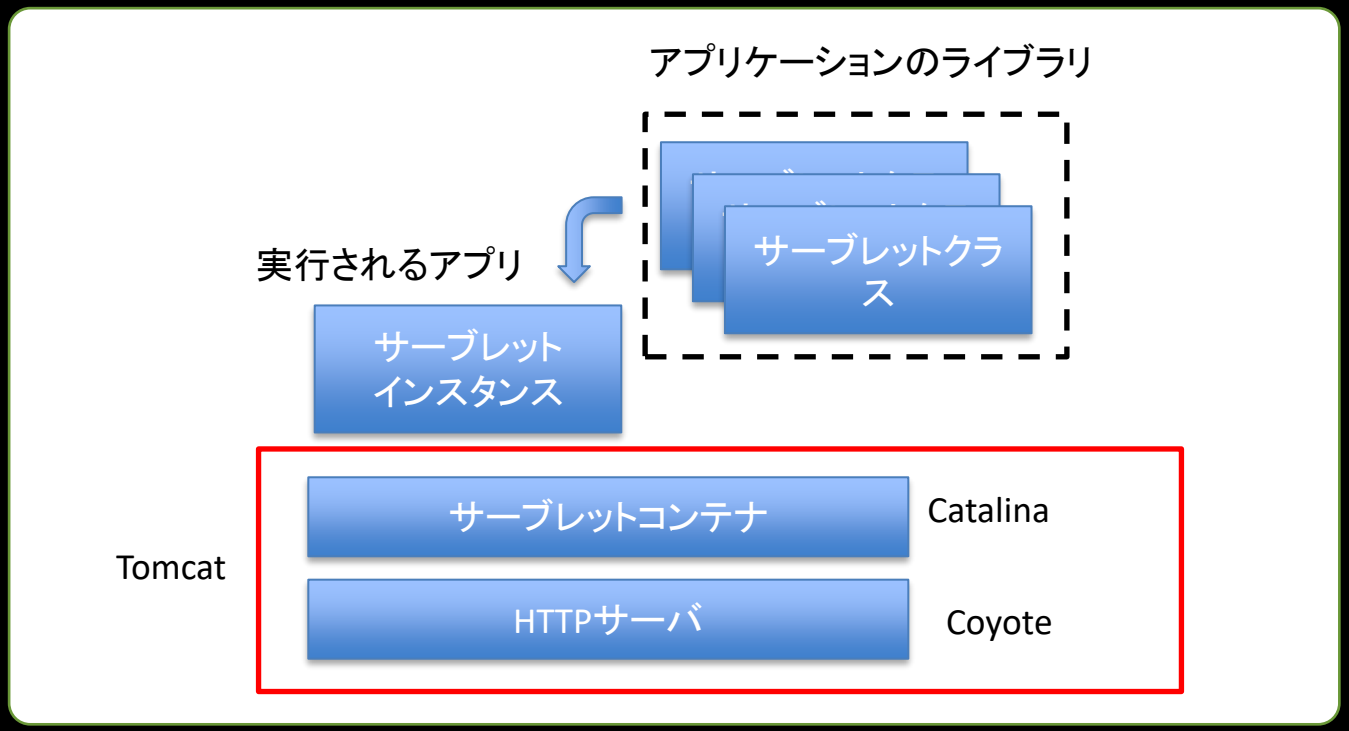
1. Javaサーブレット

- Javaサーブレット：　サーバ上で動作するJavaプログラム
- 動的Webアプリケーションを構築するためのサーバサイド技術　(cf. クライアントサイド技術、ex. JavaScript)
- ブラウザからのリクエストに対し、Javaサーブレットがサーバ上で実行される。(cf. CGIプログラム)
- Javaサーブレットはサーバ上のサーブレットコンテナという実行環境上で実行される。



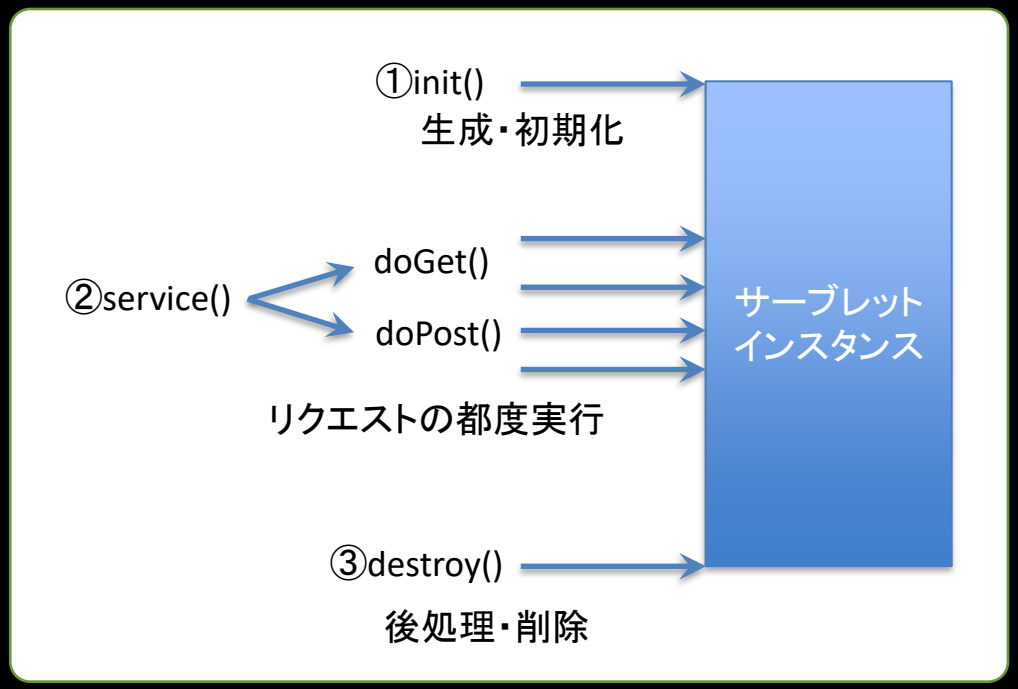
2. サーブレットコンテナ　Apache Tomcat

- サーブレットコンテナ：　Javaサーブレットの実行環境
- Apache Tomcat：Apache Software Foundationが開発しているオープンソースのサーブレットコンテナ
- Apache Tomcatはコンテナと HTTPサーバから構成　(コンテナ：Catalina、HTTPサーバ：Coyote　)
- Apache HTTP Serverとも連携可

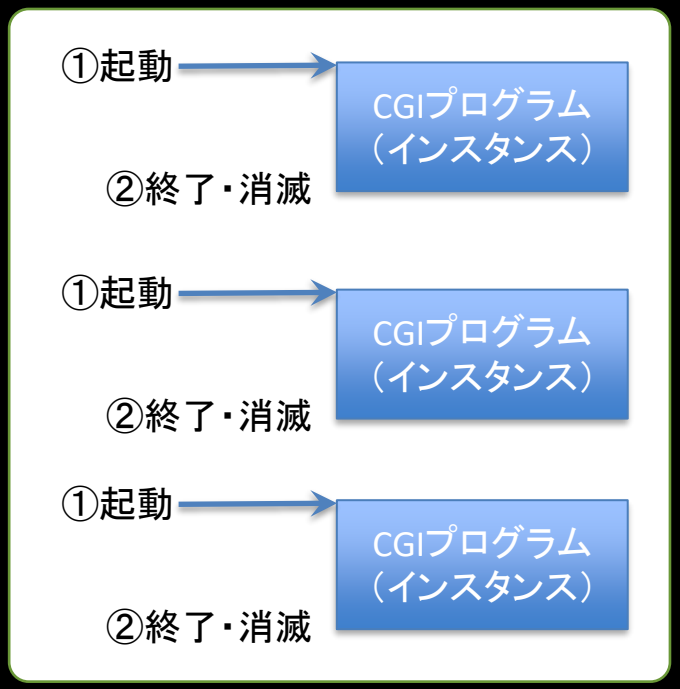


サーブレットのライフタイム

- サーブレットのライフタイム：init()からdestroy()まで
- ライフタイム中は複数のリクエストに対しサービスを提供　(cf.　CGI：リクエストの都度インスタンスが生成⇒消滅)
- サービスは並列実行可能　(要注意)



サーブレット



CGI

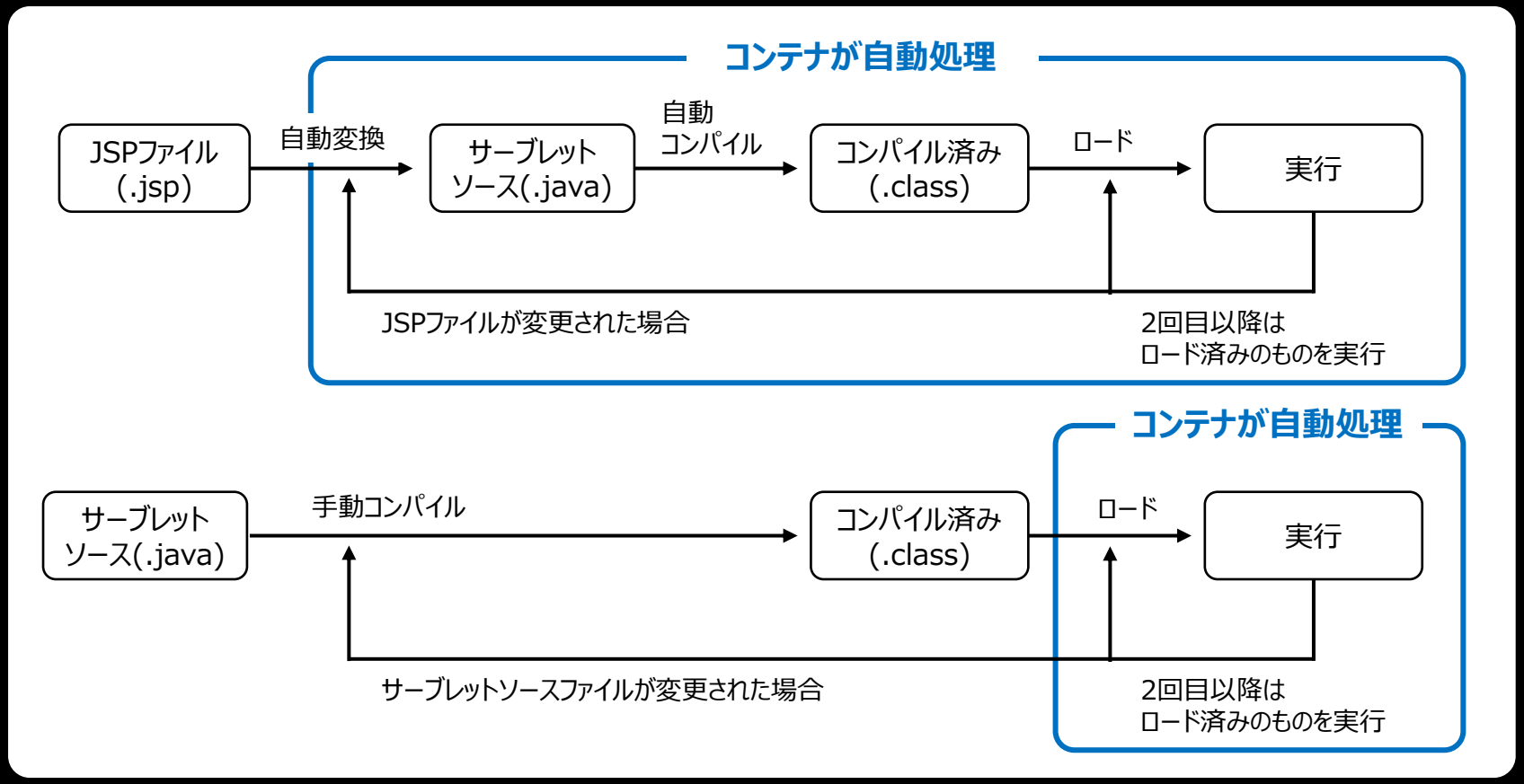
3. JSP

- JSP：　JavaServer Pages
- 動的Webアプリケーションを構築するためのサーバサイド技術
- HTMLの中に断片的なJavaプログラムを埋め込む簡易なサーバサイド処理技術

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>JSPの基礎</title>
8  </head>
9  <body>
10 <% out.print("JSPの基礎を学習します。"); %>
11 </body>
12 </html>
```

3. JSP

- JSP：　JavaServer Pages
- 本質はJavaサーブレットと同じ？だが形態が異なる。



4. JSPファイルの作成・実行

1. エクリプスの起動
2. プロジェクト（動的Web）の作成
3. JSPファイルの作成
4. 作成ファイルの実行

実

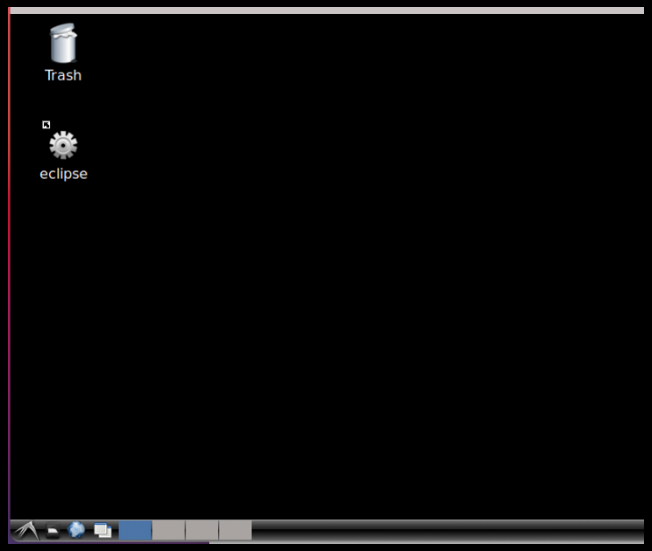
演習08-01　JSPの作成

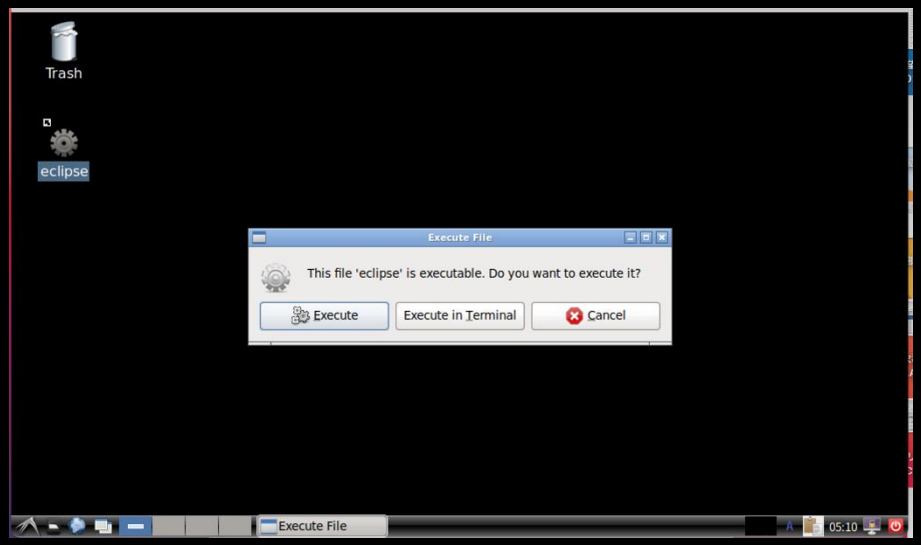
1. 下記のJSPファイルを作成し、eclipse上で実行せよ。
2. 表示画面のキャプチャファイルをSOLAに提出すること

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>JSPの基礎</title>
8  </head>
9  <body>
10 <% out.print("JSPの基礎を学習します。"); %>
11 </body>
12 </html>
```

演習08-01　JSPの作成　（補足）

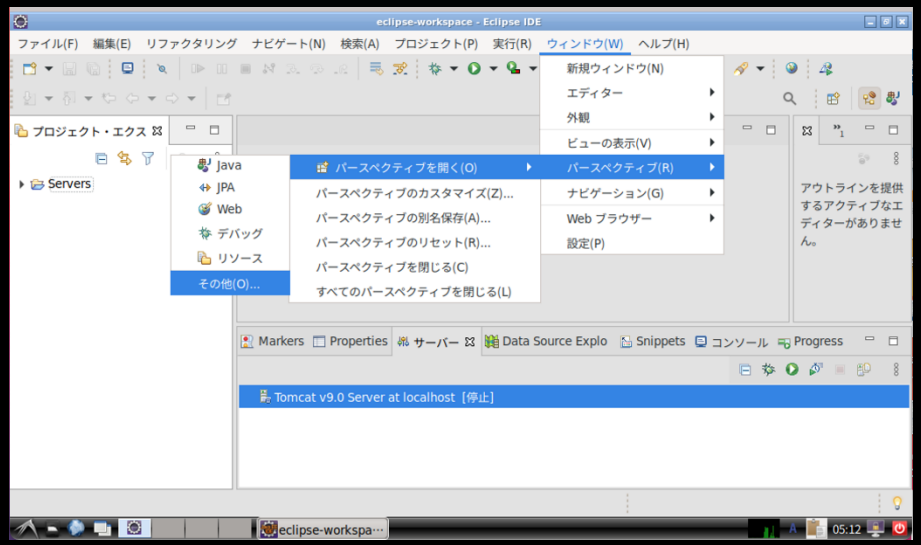
◆Eclipse の起動

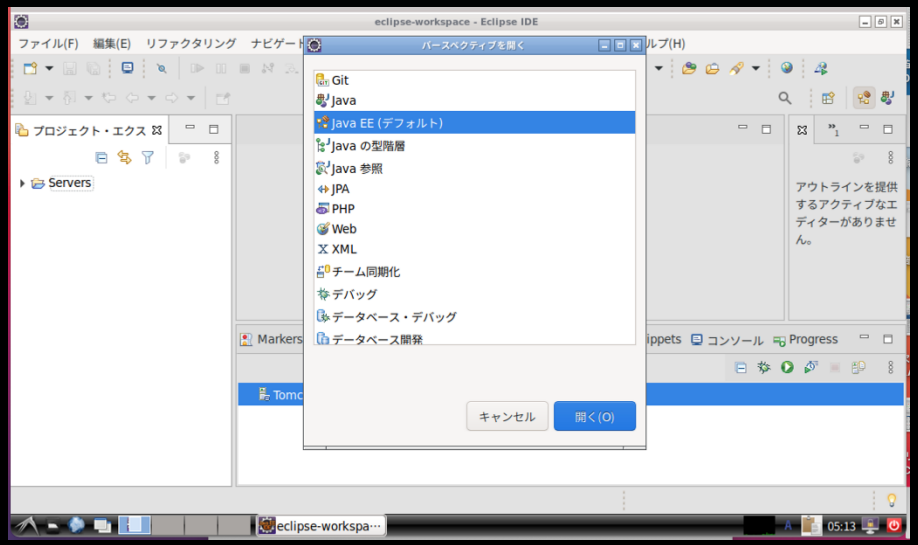




演習08-01　JSPの作成　（補足）

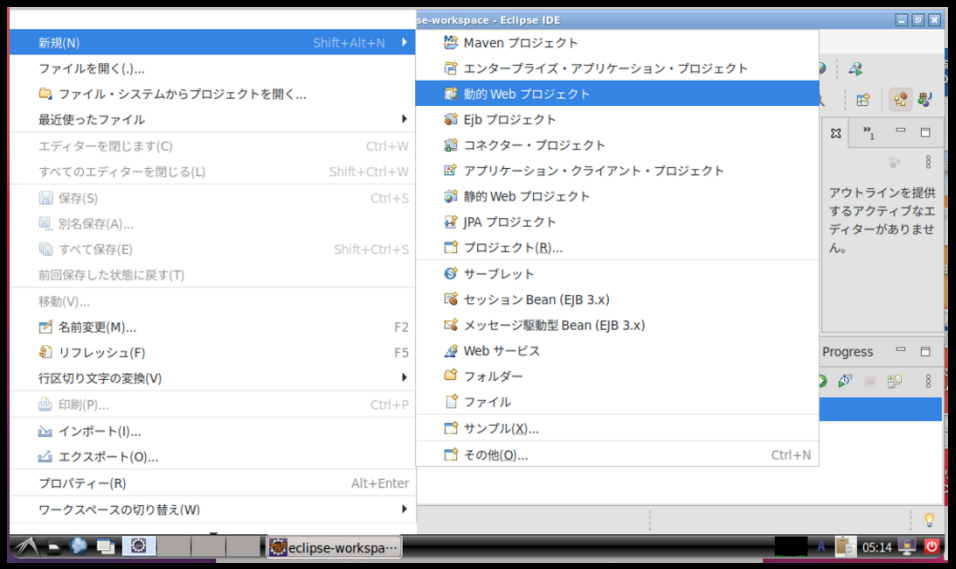
◆Eclipse のパースペクティブの選択

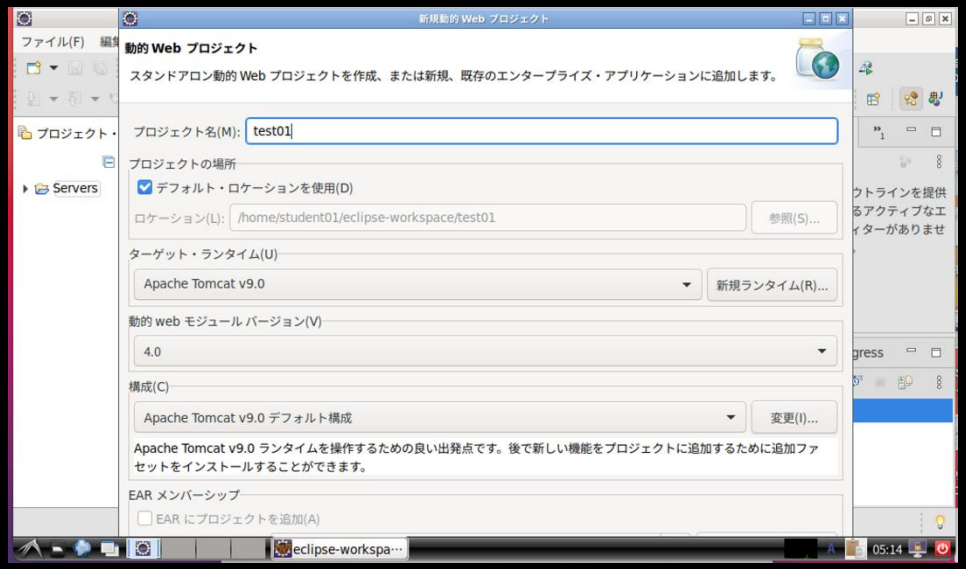


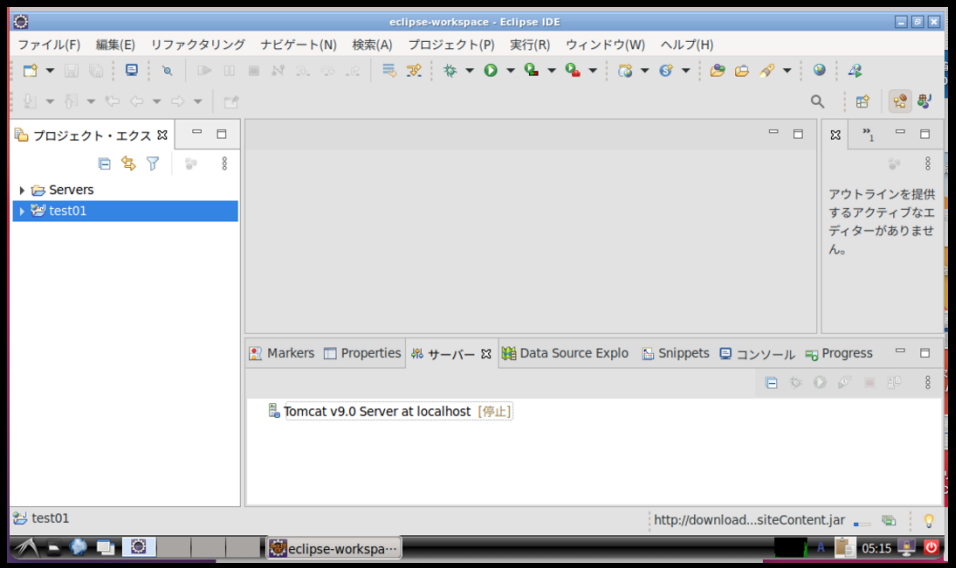


演習08-01　JSPの作成　（補足）

◆動的Webプロジェクトの作成

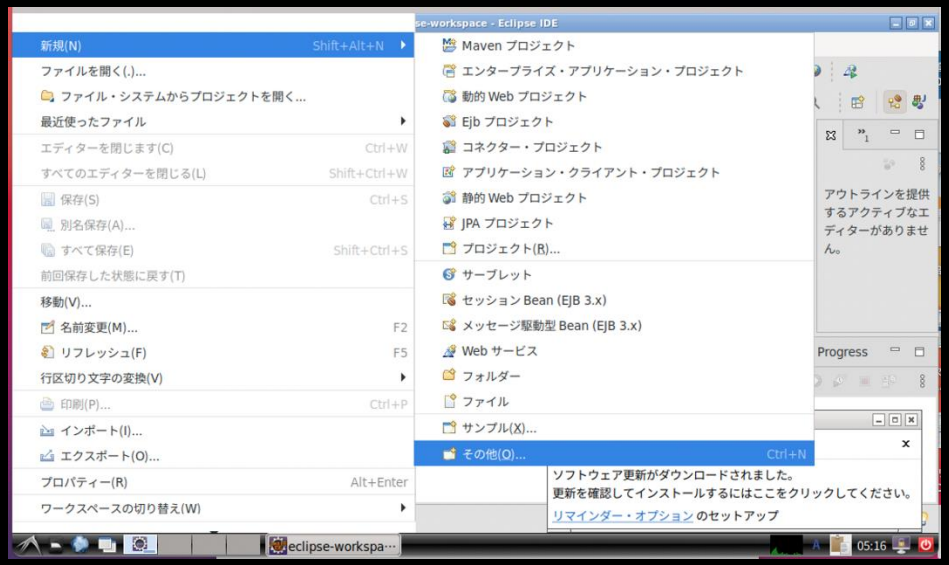


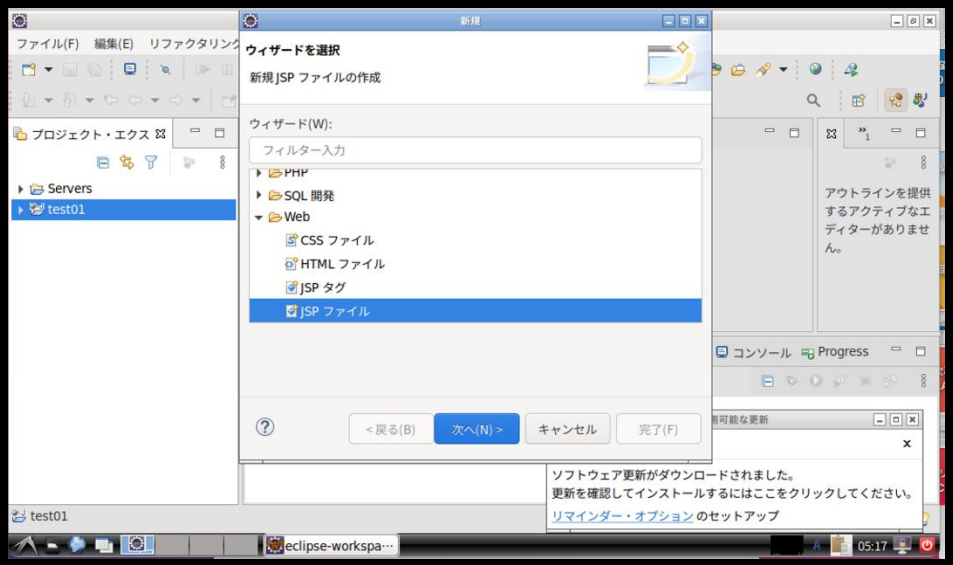


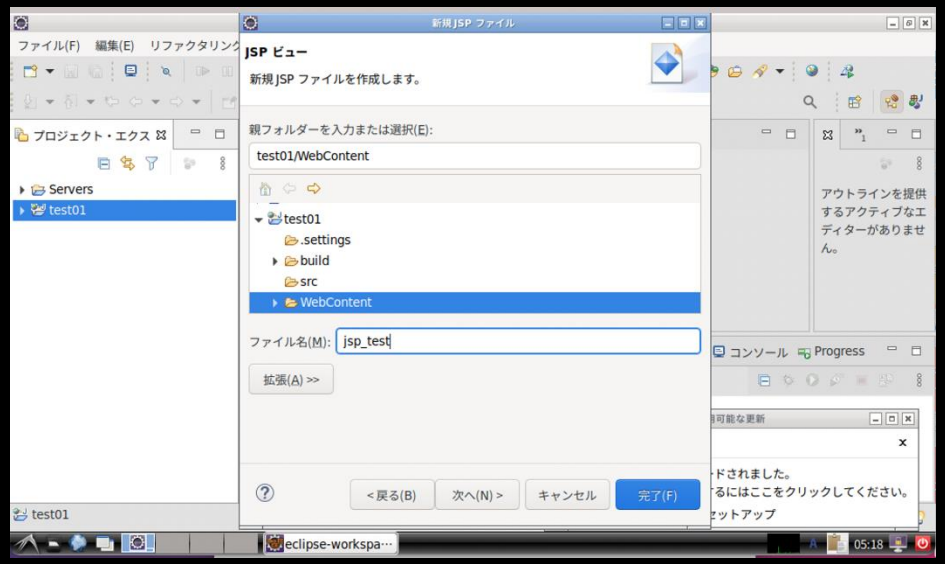


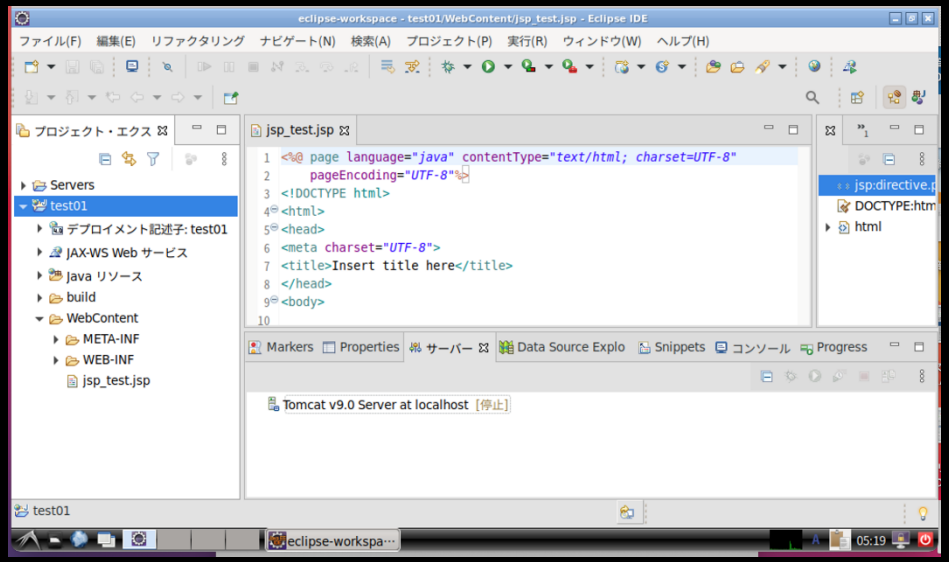
演習08-01　JSPの作成　（補足）

◆JSPファイルの作成



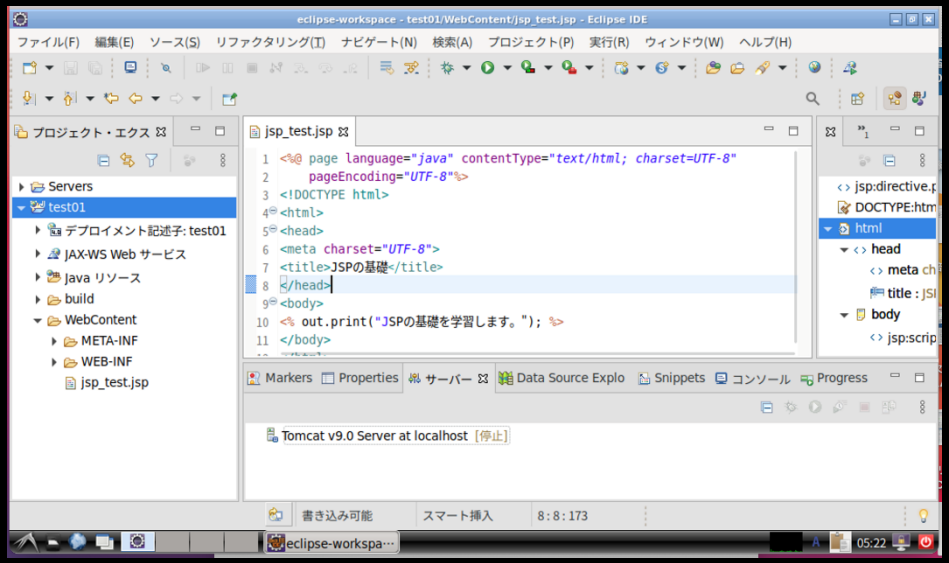


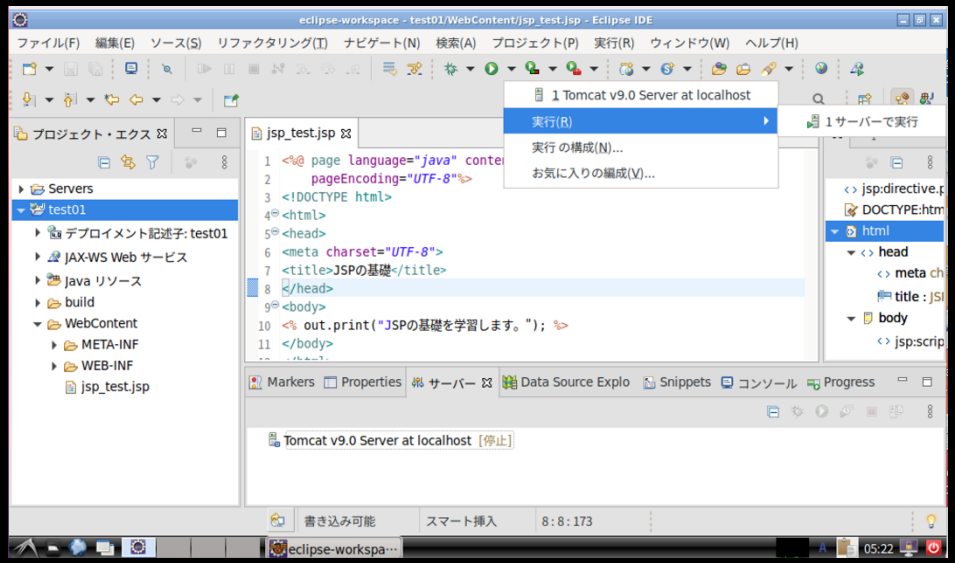


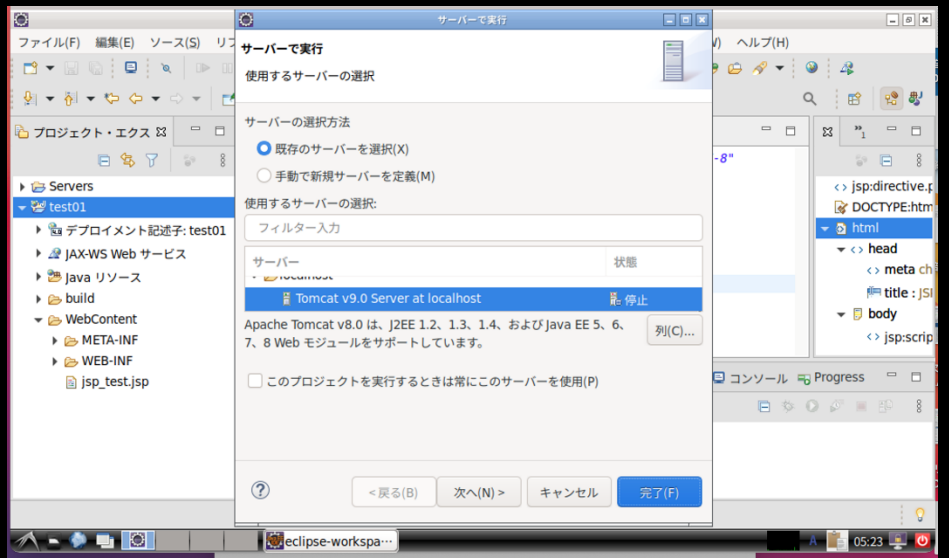


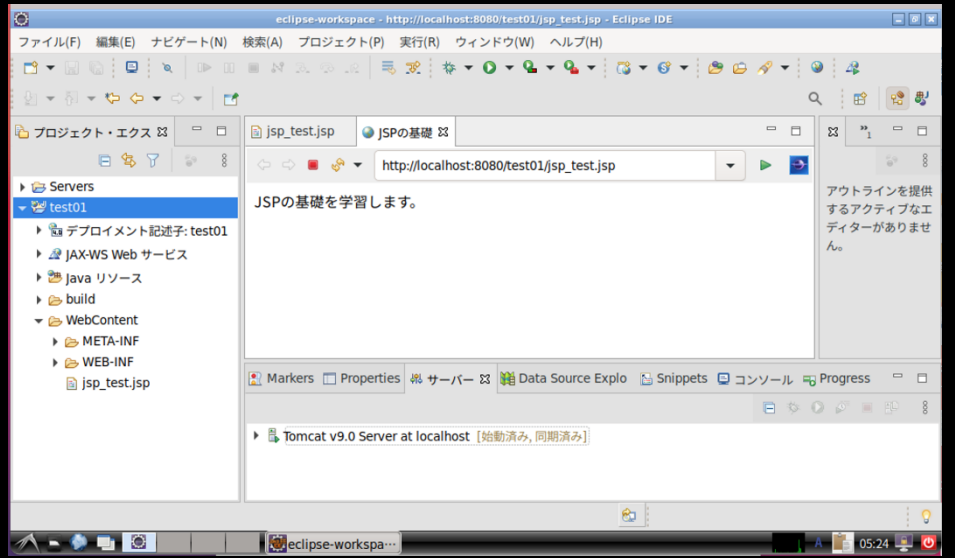
演習08-01　JSPの作成　（補足）

◆JSPファイルの編集と実行









JSP

JSP

1. JSPの基本
2. ディレクティブ
3. スクリプレット

1. JSPの基本

- HTMLの中に断片的なJavaプログラムを埋め込む
- 埋め込む要素：ディレクティブ、スクリプレット
- ディレクティブ：JSPページの処理方式をコンテナに伝える。
- スクリプレット：基本、Java

ディレクティブ

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>JSPの基礎</title>
7  </head>
8  <body>
9  <% out.print("JSPの基礎を学習します。"); %>
10 </body>
11 </html>
12
```

スクリプレット

ディレクティブ

- JSPページの処理方式をコンテナに伝える。
- 書式：　<%@ ・・・ %>
- 原則として同じ属性を2回以上指定することはできない。

```
<%@ ディレクティブ名 属性名1="属性値1" 属性名2="属性値2" 属性名3="属性値3" ･･･%>
```

○

```
<%@ page buffer="none" %>
<%@ page contentType="text/html; charset=EUC-JP" %>
```

×

```
<%@ page contentType="text/html; charset=Windows-31J" %>
<%@ page contentType="text/html; charset=EUC-JP" %>
```

ディレクティブ	概要
@page	ページの処理方法を決める。
@include	外部ファイルをインクルードする。
@taglib	ページ内で使用するタグライブラリを宣言する。
@tag	タグファイルの処理方法を決める。
@attribute	タグファイルでの利用可能な属性を宣言する。

ディレクティブ　@page

- ページの基本的な処理を規定

属性	概要
contentType	コンテンツの種類と出力の文字コードを指定 デフォルト：text/html;charset=ISO-8859-1
pageEncoding	.jspファイルを記述した文字コードを宣言 省略した場合はcontentTypeの文字コード
buffer	ページを生成し出力する際のバッファサイズ デフォルト：8KB (none:蓄積なし)
atutoFlush	指定バッファサイズを超えた時の動作 true：自動出力、 false：エラー出力 デフォルト：true
import	使用するクラスライブラリを指定
session	セッションの有効無効を指定 true：有効、 false：無効 デフォルト：false

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
```

スクリプレット

- HTMLに埋め込む小さなJavaプログラム
- 書式：　<%　・・・　%>
- 一つ以上の命令文で、命令文はセミコロン(;)で終わる。
- 暗黙オブジェクト　：　インスタンス化手続きなしで使用できるオブジェクト

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>JSPの基礎</title>
7  </head>
8  <body>
9  <% out.print("JSPの基礎を学習します。"); %>
10 </body>
11 </html>
```

※ out ：暗黙オブジェクトのひとつ

暗黙オブジェクト	概要
application	ユーザ間で共有できる情報を管理する。
config	アプリケーションの設定情報を管理する。
exception	例外(エラー)情報を管理する。
out	クライアントへの出力を行う。
pageContext	他の暗黙オブジェクトへのアクセス手段を提供する。
request	クライアントからの要求情報を受け取る。
response	クライアントへの出力を制御する。
session	セッションで共有できる情報を管理する。

繰り返し

- スクリプレットの中はJavaのコード
- 変数、制御文も使える。
- 宣言部　書式：　<%!　・・・　%>
- 宣言部：変数、メソッドの宣言のみ。宣言された変数はコンテナのライフタイムの間、有効、

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%!
3  String[] names = { "松岡", "薄井", "本多", "川村", "日尾" };
4  %>
5  <!DOCTYPE html>
6  <html>
7  <head>
8  <meta charset="UTF-8" />
9  <title>JSPの基本(1)</title>
10 </head>
11 <body>
12 <%
13 int i;
14 int count = names.length;
15 for(i = 0; i < count; i++) {
16   out.println("<p>こんにちは、" + names[i] + "さん！ </p>");
17 }
18 %>
19 </body>
20 </html>
```

実

演習08-02　JSPの作成 (2)

1. 下記のJSPファイルを作成し、eclipse上で実行せよ。
2. 表示画面のキャプチャファイルをSOLAに提出すること

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%!
3  String[] names = { "松岡", "薄井", "本多", "川村", "日尾" };
4  %>
5  <!DOCTYPE html>
6  <html>
7  <head>
8  <meta charset="UTF-8" />
9  <title>JSPの基本(1)</title>
10 </head>
11 <body>
12 <%
13 int i;
14 int count = names.length;
15 for(i = 0; i < count; i++) {
16   out.println("<p>こんにちは、" + names[i] + "さん！ </p>");
17 }
18 %>
19 </body>
20 </html>
```

フォームからの入力　POST

- これまでに学習してきた内容と基本的に同じ

quest1.jsp

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>JSPの基礎(2)アンケート</title>
7  </head>
8  <body>
9  <form method="POST" action="quest2.jsp">
10 <div>
11     この本はおもしろいですか？<br />
12     <label>
13         <input type="radio" name="interest" value="1" checked="checked" />おもしろい！
14     </label>
15     <label>
16         <input type="radio" name="interest" value="0" />おもしろくない！
17     </label>
18 </div>
19 <div>
20     <input type="submit" value="登録" />
21 </div>
22 </form>
23 </body>
24 </html>
25
```

フォームからの入力　POST

- これまでに学習してきた内容と基本的に同じ
- POSTされたパラメータは request.getParameter() で取得　（※GETでのURLパラメータも取得可）

quest2.jsp

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>アンケート結果</title>
7  </head>
8  <body>
9  <%
10 String answer = request.getParameter("interest");
11 if(Integer.parseInt(answer) == 1) {
12     out.println("よかったぁ！");
13 } else {
14     out.println("えぇ、なんでぇ？？？");
15 }
16 %>
17 </body>
18 </html>
```

URLパラメータ

1. URLパラメータとは、サーバーに情報を送るためにURLに付け加える変数
2. クエッションマーク「?」をURLの末尾に付けることで、「変数（パラメータ） = 値」の形式でサーバーに送信するデータをURLに含めることが可能
3. アクティブパラメータとパッシブパラメータの2種類

```
https://wwww.example.jp/hogehoge/?category=hight
```

```
https://wwww.example.jp/hogehoge/?category=hight&category=lenght
```

実

演習08-03　POST処理

1. quest1.jsp, quest2.jsp を作成し、eclipse上で実行せよ。
2. quest1.jsp, quest2.jsp の表示画面のキャプチャファイルをSOLAに提出すること

1.4 JSPの要素　式 (Expression)

- スクリプレットの代わりをする記法
- 書式：　<%= ・・・ %>
- 注意：式の中にスクリプレットを書くことはできない。

```
<title><%=msg %></title>
```

```
<title><% out.print(msg); %></title>
```

実

演習08-04　Expression

1. 演習08-01の内容をExpressionを用いて書き換えよ。
2. Jspファイルと表示画面のキャプチャファイルをSOLAに提出すること

JSPの実行

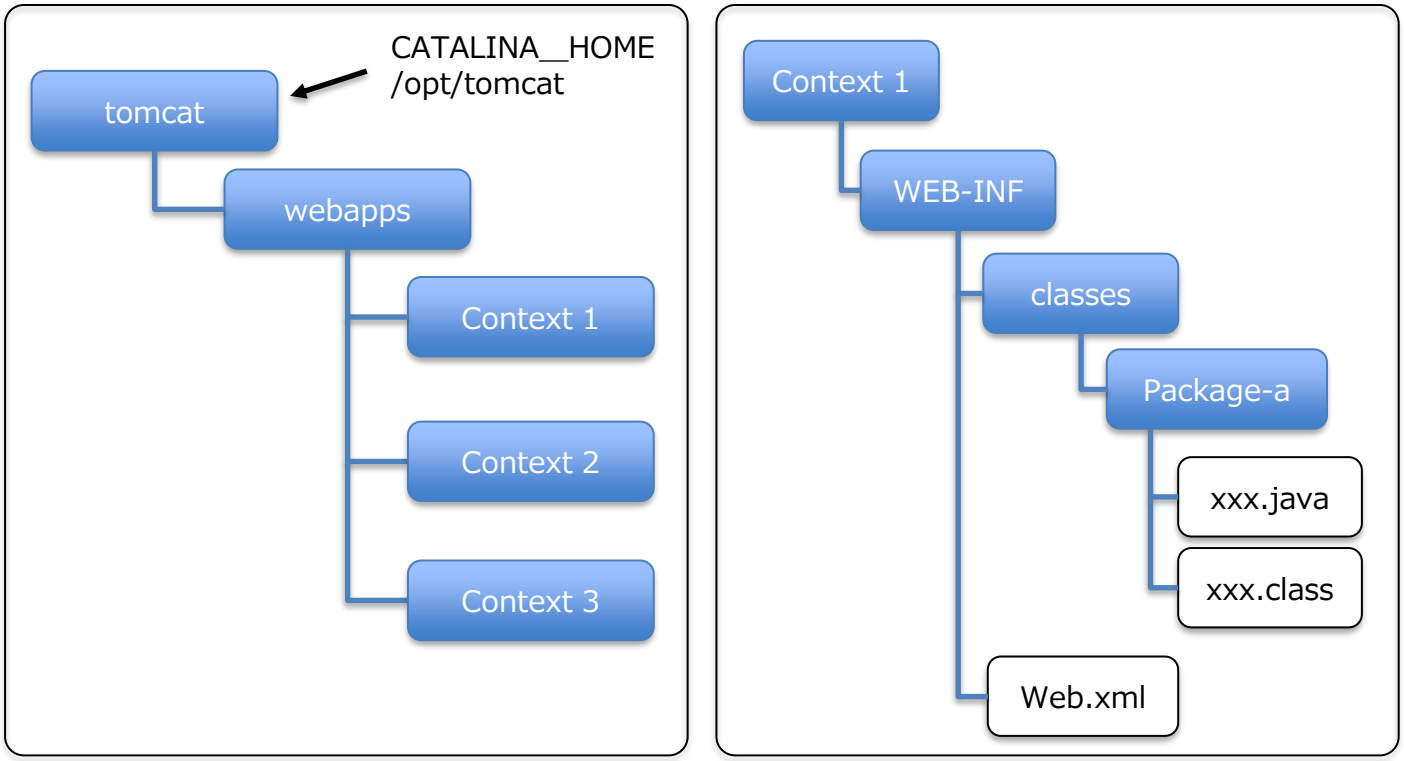
Tomcatでの実行

実

1. WARファイルの作成
2. Tomcatのディレクトリにコピー
3. コンテナ(catalina) の起動　　sudo　/opt/tomcat/bin/catalina.sh　start

※ WARファイル　：Web Application aRchive 。 Javaベースの
Webアプリケーションやコンポーネントを圧縮し納めたファイル

【注意】EclipseのサーバとTomcatのサーバは両方起動するとコンフリクトを起こす。



おわり

サーバシステムプログラミング及び演習

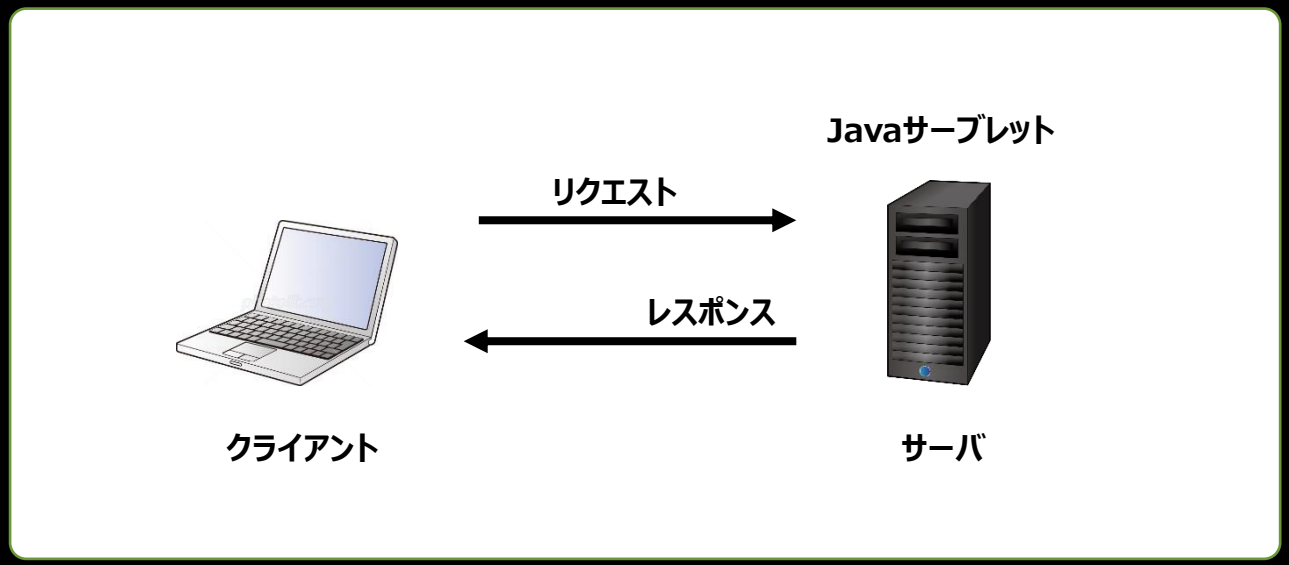
– 第09回 JSP(2) –

前回の学習内容

1. サーブレット
2. サーブレットコンテナ　Tomcat
3. JSP

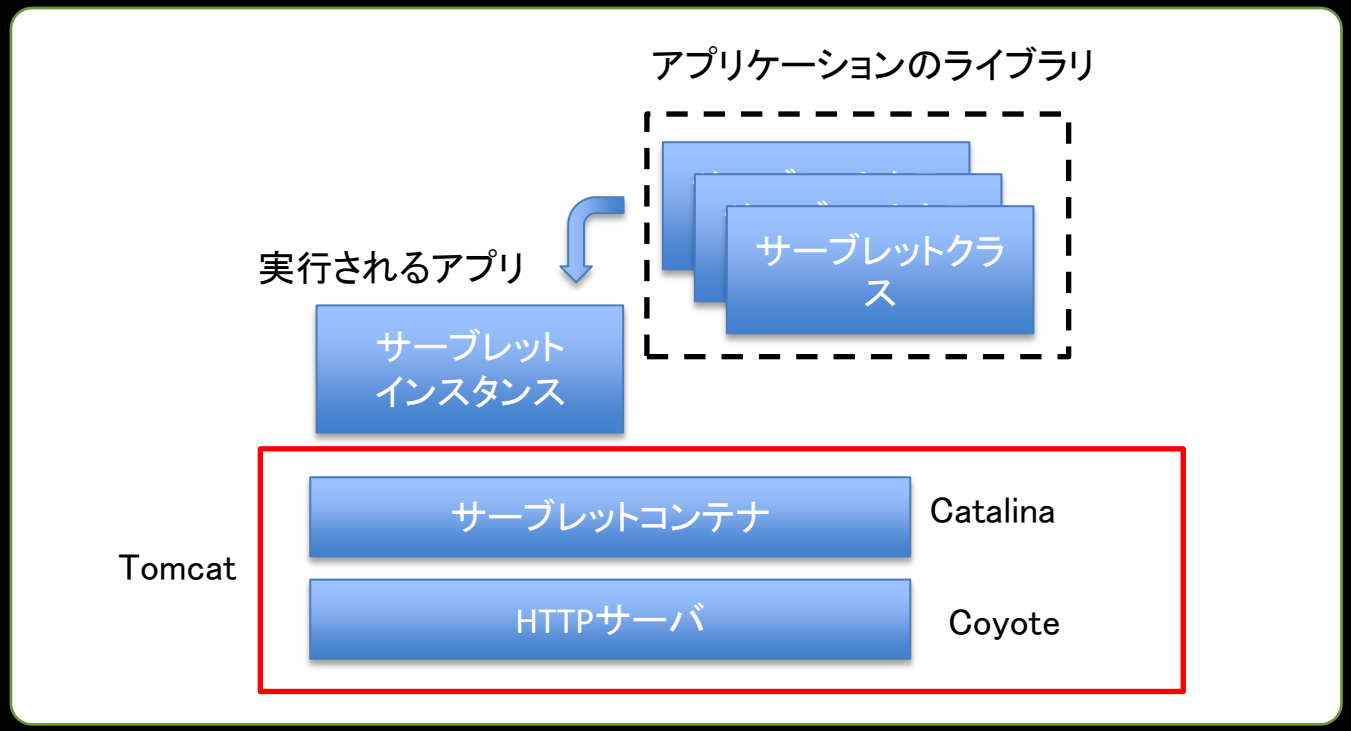
1. Javaサーブレット

- Javaサーブレット：　サーバ上で動作するJavaプログラム
- 動的Webアプリケーションを構築するためのサーバサイド技術　(cf. クライアントサイド技術、ex. JavaScript)
- ブラウザからのリクエストに対し、Javaサーブレットがサーバ上で実行される。(cf. CGIプログラム)
- Javaサーブレットはサーバ上のサーブレットコンテナという実行環境上で実行される。



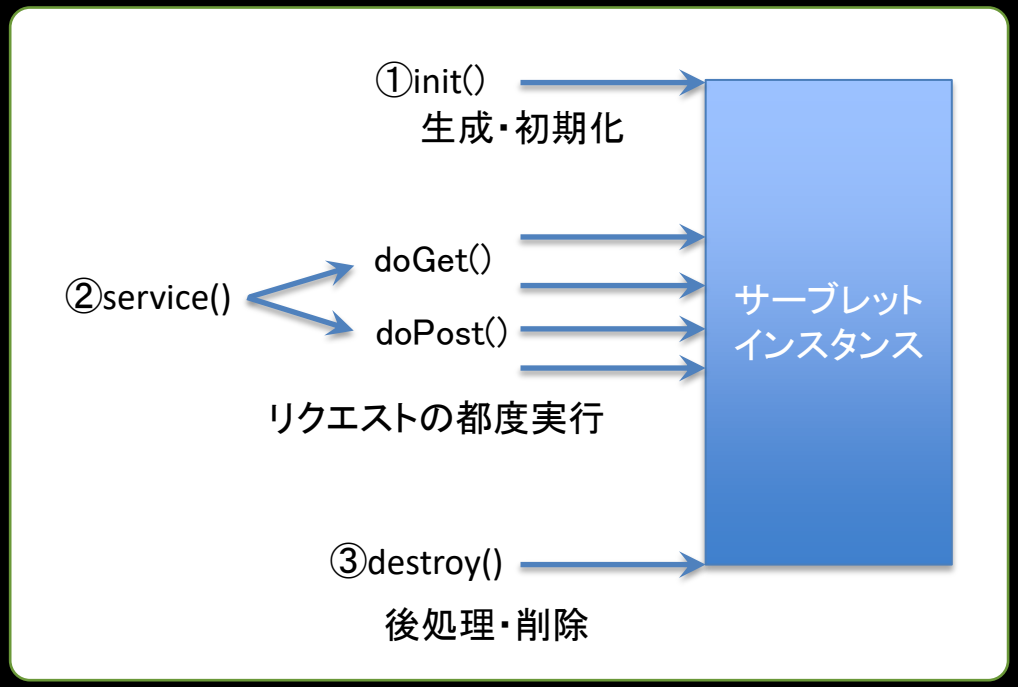
2. サーブレットコンテナ　Apache Tomcat

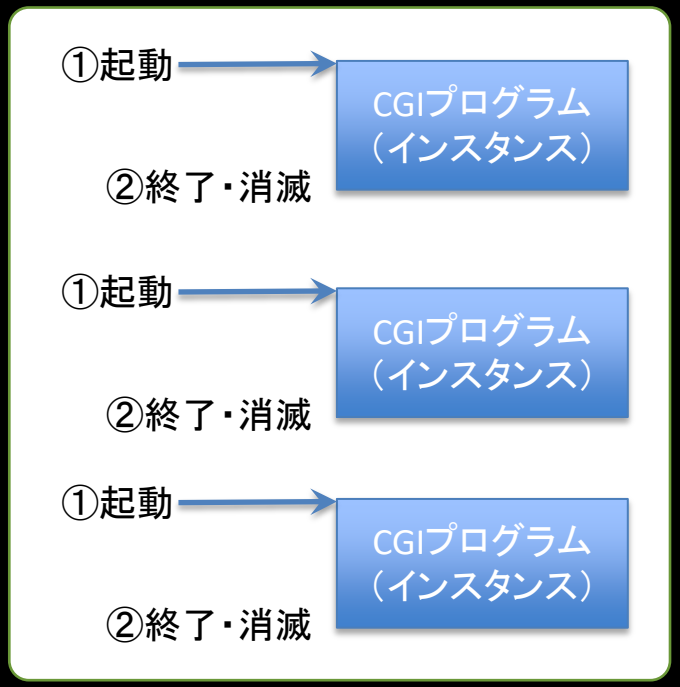
- サーブレットコンテナ：　Javaサーブレットの実行環境
- Apache Tomcat：Apache Software Foundationが開発しているオープンソースのサーブレットコンテナ
- Apache Tomcatはコンテナと HTTPサーバから構成　(コンテナ：Catalina、HTTPサーバ：Coyote　)
- Apache HTTP Serverとも連携可



サーブレットのライフタイム

- サーブレットのライフタイム：init()からdestroy()まで
- ライフタイム中は複数のリクエストに対しサービスを提供　(cf.　CGI：リクエストの都度インスタンスが生成⇒消滅)
- サービスは並列実行可能　(要注意)





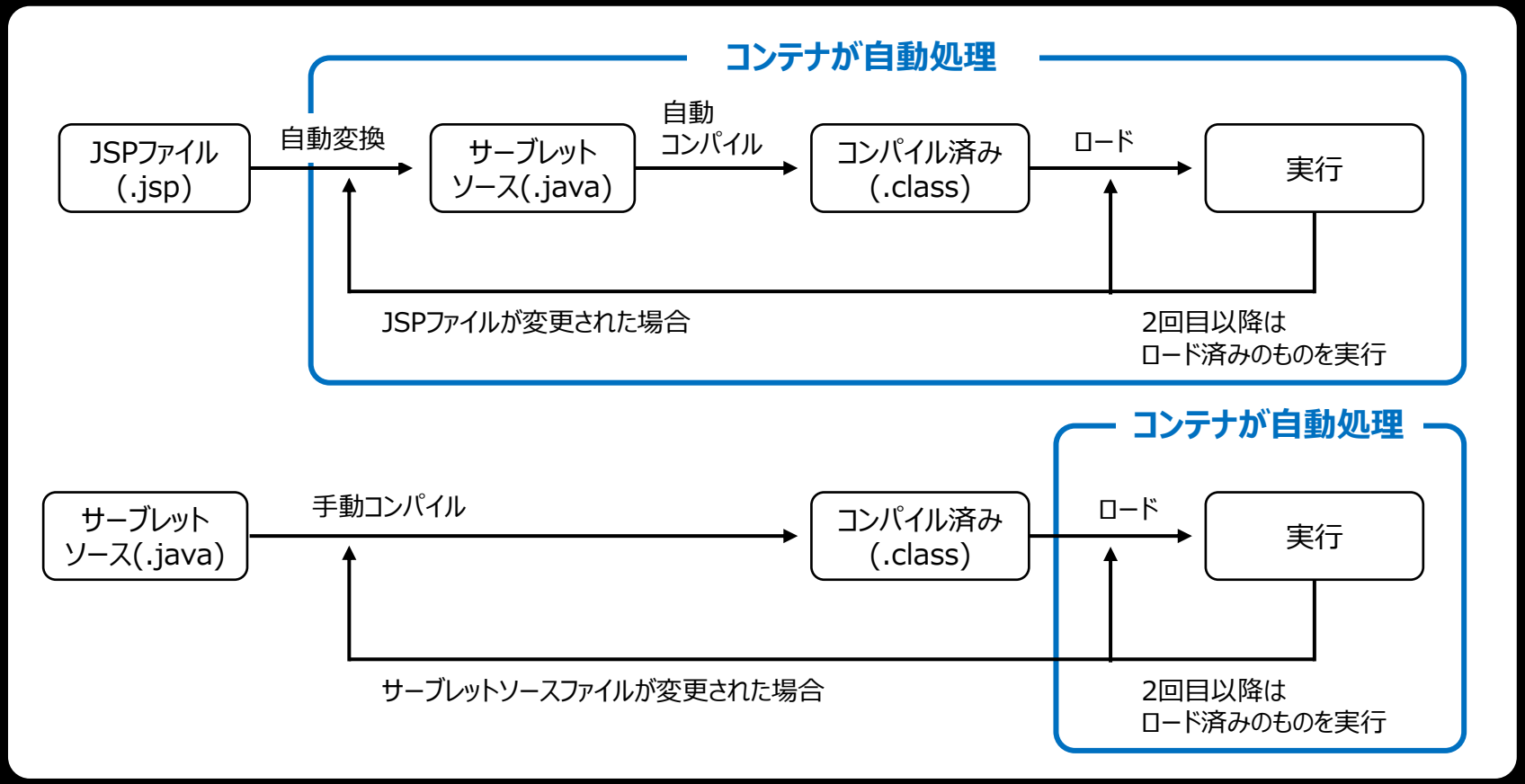
3. JSP

- JSP :　JavaServer Pages
- 動的Webアプリケーションを構築するためのサーバサイド技術
- 本質はJavaサーブレットと同じ？だが形態が異なる。
- HTMLの中に断片的なJavaプログラムを埋め込む簡易なサーバサイド処理技術

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>アンケート結果</title>
7  </head>
8  <body>
9  <%
10 String answer = request.getParameter("interest");
11 if(Integer.parseInt(answer) == 1) {
12     out.println("よかったぁ！");
13 } else {
14     out.println("えぇ、なんでぇ？？？");
15 }
16 %>
17 </body>
18 </html>
```

4. Servlet と JSP

- ServletはHTMLを出力する。
- JSPはHTMLに埋め込まれる。
- Servletはロジックの記述は得意だが、静的なHTMLを大量に出力するのは苦手
- JSPはHTMLは得意？だが、ロジックは苦手　(ページの構造が見えづらくなる)



JSP (2)

JSP（2）

1. アクションタグ
2. Cookie

1. JSPの要素

- HTMLの中に断片的なJavaプログラムを埋め込む
- 埋め込む要素：ディレクティブ、スクリプレット等
- 表記載の要素以外のところを**HTMLテンプレート(固定テンプレート)**と呼ぶ。

要素	書式	概要
ディレクティブ	<%@ ～ %>	ページの処理方法を決める。
宣言部	<%! ～ %>	変数、定数、ユーザ定義メソッドを宣言する。
スクリプレット	<% ～ %>	小さなJavaのプログラム
式 (Expression)	<%= ～ %>	指定された式を出力する。
アクションタグ	<jsp: ～ >	定型的な処理をタグの形式で記述する。
コメント	<%-- ～ --%>	処理の際には無視される。備考等を記述する。

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>JSPの基礎</title>
7  </head>
8  <body>
9  <% out.print("JSPの基礎を学習します。"); %>
10 </body>
11 </html>
```

HTMLテンプレート

HTMLテンプレート

1.1 JSPの要素　ディレクティブ

- JSPページの処理方式をコンテナに伝える。
- 書式：　<%@ ・・・ %>
- 原則として同じ属性を2回以上指定することはできない。

```
<%@ ディレクティブ名 属性名1="属性値1" 属性名2="属性値2" 属性名3="属性値3" ・・・%>
```

○

```
<%@ page buffer="none" %>
<%@ page contentType="text/html; charset=EUC-JP" %>
```

×

```
<%@ page contentType="text/html; charset=Windows-31J" %>
<%@ page contentType="text/html; charset=EUC-JP" %>
```

ディレクティブ	概要
@page	ページの処理方法を決める。
@include	外部ファイルをインクルードする。
@taglib	ページ内で使用するタグライブラリを宣言する。
@tag	タグファイルの処理方法を決める。
@attribute	タグファイルでの利用可能な属性を宣言する。

1.1.1 JSPの要素　ディレクティブ @page

- ページの基本的な処理を規定

属性	概要
contentType	コンテンツの種類と出力の文字コードを指定 デフォルト：text/html;charset=ISO-8859-1
pageEncoding	.jspファイルを記述した文字コードを宣言 省略した場合はcontentTypeの文字コード
buffer	ページを生成し出力する際のバッファサイズ デフォルト：8KB (none:蓄積なし)
atutoFlush	指定バッファサイズを超えた時の動作 true：自動出力、 false：エラー出力 デフォルト：true
import	使用するクラスライブラリを指定
session	セッションの有効無効を指定 true：有効、 false：無効 デフォルト：false

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
```

1.2 JSPの要素　宣言部

- 宣言部：変数、メソッドの定義のみ　※コードは記述できない。
- 書式：　<%! ・・・ %>
- 宣言部では暗黙オブジェクトが利用できない。
- 変数の有効期間：コンテナが起動している間 (cf. スクリプレットの変数：処理単位)

```
1  <%@ page language="java" contentType="text/html; charset=UTF-7"
2      pageEncoding="UTF-8"%>
3  <%!
4  final int M_SIZE = 100000;
5
6  int             num;
7  int[]   e_num   = new int[M_SIZE];
8  byte[]  map             = new byte[M_SIZE];
9
10 public void CreateMap() {
11         int     i, j;
12
13         if( num > 0 )
14                 return;
15
16         for( i=0; i<M_SIZE; i++ )
17                 map[i] = 1;
18
19
20                                 ▪
21
22                                 ▪
23
24                                 ▪
25
26
27
28 }
29  %>
30 <!DOCTYPE html>
```

1.3 JSPの要素 スクリプレット

- HTMLに埋め込む小さなJavaプログラム
- 書式：　<%　・・・　%>
- 一つ以上の命令文で、命令文はセミコロン(;)で終わる。
- 暗黙オブジェクト　：　インスタンス化手続きなしで使用できるオブジェクト

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>JSPの基礎</title>
7  </head>
8  <body>
9  <% out.print("JSPの基礎を学習します。"); %>
10 </body>
11 </html>
```

※ out ：暗黙オブジェクトのひとつ

暗黙オブジェクト	概要
application	ユーザ間で共有できる情報を管理する。
config	アプリケーションの設定情報を管理する。
exception	例外(エラー)情報を管理する。
out	クライアントへの出力を行う。
pageContext	他の暗黙オブジェクトへのアクセス手段を提供する。
request	クライアントからの要求情報を受け取る。
response	クライアントへの出力を制御する。
session	セッションで共有できる情報を管理する。

1.3.1 JSPの要素 スクリプレット　繰り返し

- スクリプレットの中はJavaのコード
- 変数、制御文も使える。
- 宣言部　書式：　<%!　・・・　%>
- 宣言部：変数、メソッドの宣言のみ。宣言された変数はコンテナのライフタイムの間、有効、

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <%!
3  String[] names = { "松岡", "薄井", "本多", "川村", "日尾" };
4  %>
5  <!DOCTYPE html>
6  <html>
7  <head>
8  <meta charset="UTF-8" />
9  <title>JSPの基本(1)</title>
10 </head>
11 <body>
12 <%
13 int i;
14 int count = names.length;
15 for(i = 0; i < count; i++) {
16   out.println("<p>こんにちは、" + names[i] + "さん！ </p>");
17 }
18 %>
19 </body>
20 </html>
```

1.3.1 JSPの要素 スクリプレット　フォームからの入力　POST

- これまでに学習してきた内容と基本的に同じ

quest1.jsp

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>JSPの基礎(2)アンケート</title>
7  </head>
8  <body>
9  <form method="POST" action="quest2.jsp">
10 <div>
11     この本はおもしろいですか？<br />
12     <label>
13         <input type="radio" name="interest" value="1" checked="checked" />おもしろい！
14     </label>
15     <label>
16         <input type="radio" name="interest" value="0" />おもしろくない！
17     </label>
18 </div>
19 <div>
20     <input type="submit" value="登録" />
21 </div>
22 </form>
23 </body>
24 </html>
25
```

1.3.1 JSPの要素 スクリプレット　フォームからの入力　POST

- これまでに学習してきた内容と基本的に同じ
- POSTされたパラメータは request.getParameter() で取得　　(※GETでのURLパラメータも取得可)

quest2.jsp

```
1  <%@ page contentType="text/html;charset=UTF-8" %>
2  <!DOCTYPE html>
3  <html>
4  <head>
5  <meta charset="UTF-8" />
6  <title>アンケート結果</title>
7  </head>
8  <body>
9  <%
10 String answer = request.getParameter("interest");
11 if(Integer.parseInt(answer) == 1) {
12     out.println("よかったぁ！");
13 } else {
14     out.println("えぇ、なんでぇ？？？");
15 }
16 %>
17 </body>
18 </html>
```

URLパラメータ

1. URLパラメータとは、サーバーに情報を送るためにURLに付け加える変数
2. クエッションマーク「?」をURLの末尾に付けることで、「変数（パラメータ） = 値」の形式でサーバーに送信するデータをURLに含めることが可能
3. アクティブパラメータとパッシブパラメータの2種類

```
https://wwww.example.jp/hogehoge/?category=hight
```

```
https://wwww.example.jp/hogehoge/?category=hight&category=lenght
```

1.4 JSPの要素　式 (Expression)

- スクリプレットの代わりをする記法
- 書式：　<%= ・・・ %>
- 注意：式の中にスクリプレットを書くことはできない。

```
<title><%=msg %></title>

<title><% out.print(msg); %></title>
```

1.5 JSPの要素　アクションタグ

- 典型的な処理を**Javaコードを記述することなく**記述する仕組み
- Javaを知らなくてもページを記述できる。

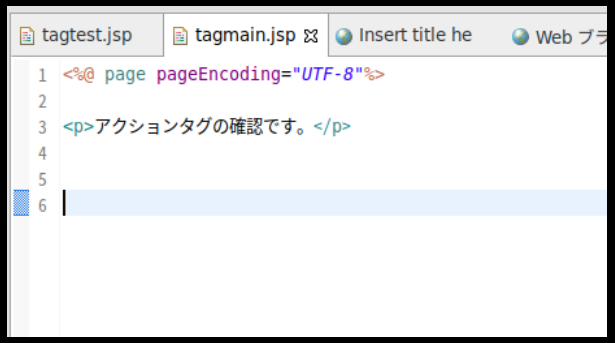
分類	アクションタグ	概要
一般	<jsp:forward>	ページの処理を転送
	<jsp:include>	外部ファイルをインクルード
タグファイル	<jsp:attribute>	属性値を定義
	<jsp:body>	タグ本体を定義
	<jsp:doBody>	タグ本体を実行
	<jsp:invoke>	フラグメントを実行
JavaBeans	<jsp:getProperty>	JavaBeansクラスのプロパティ値を取得
	<jsp:setProperty>	JavaBeansクラスのプロパティ地を設定
	<jsp:useBean>	JavaBeansクラスをインスタンス化

```
2  <body>
3  <jsp:include page="test.jsp" flush="true" />
4  </body>
5
```

実

演習09-01　アクションタグ(include)

1. 2つのJSPファイル　tagtest.jsp, tagmain.jsp　を作成して下さい。
2. tagmain.jsp は下記の通りとして下さい。
3. tagtest.jsp で tagmian.jsp をインクルードし、表示できるように、tagtest.jspを作成して下さい。
4. tagtest.jsp および実行画面のキャプチャファイルをSOLAに提出して下さい。



実

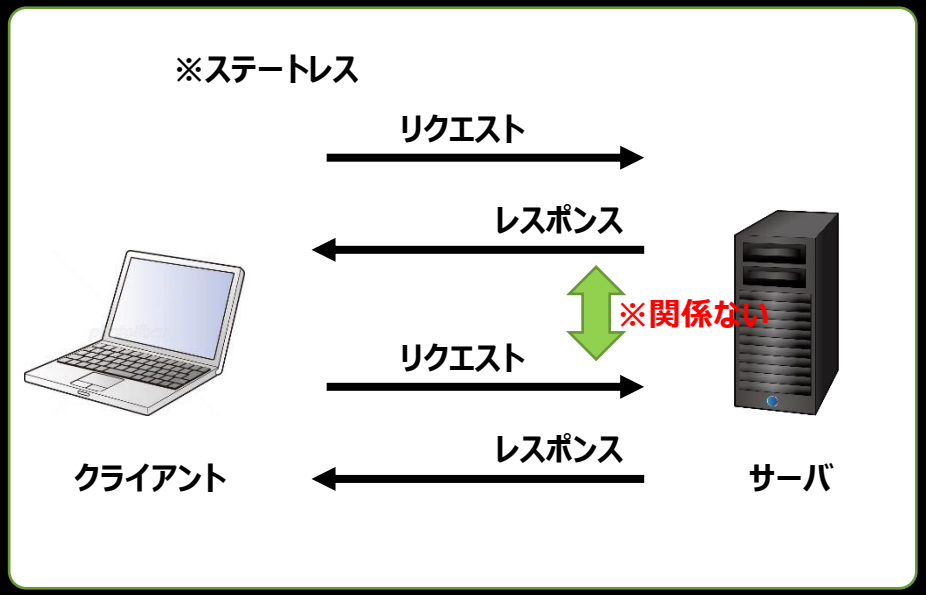
演習09-02　宣言部での変数

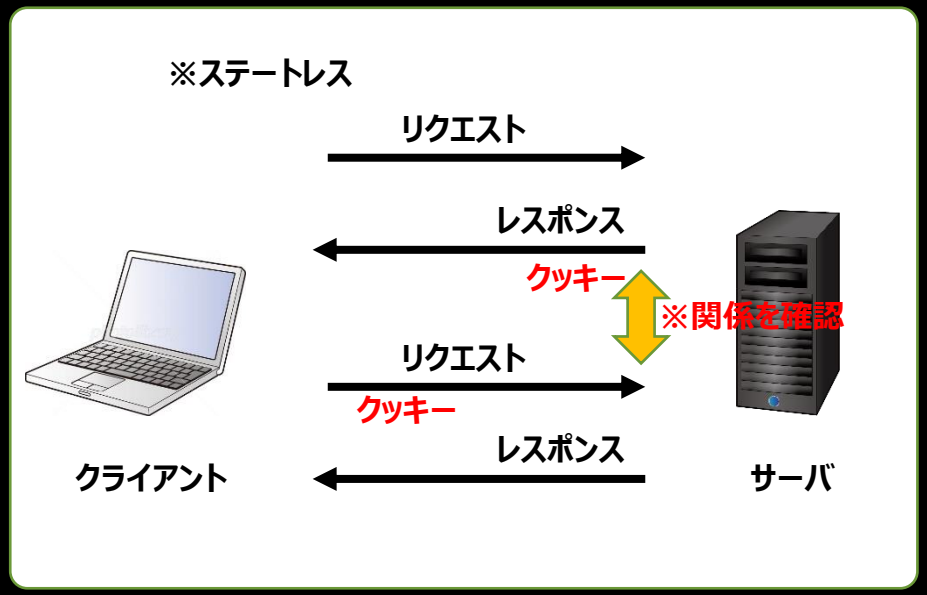
1. 宣言部に表示回数を保持する変数を用意し、表示回数を表示するページをJSPで作成して下さい。
2. 作成したJSPファイルおよび実行画面のキャプチャファイルをSOLAに提出して下さい。

Cookie

1. Cookie 概要

- クッキー：クライアント側に保存される小さなテキスト
- セキュリティ上、一般的にはサーバはクライアントのファイルは読み書き不可
- クッキーは例外
- HTTPのステートレスを補完
- クライアント単位で維持したい情報の管理が容易に



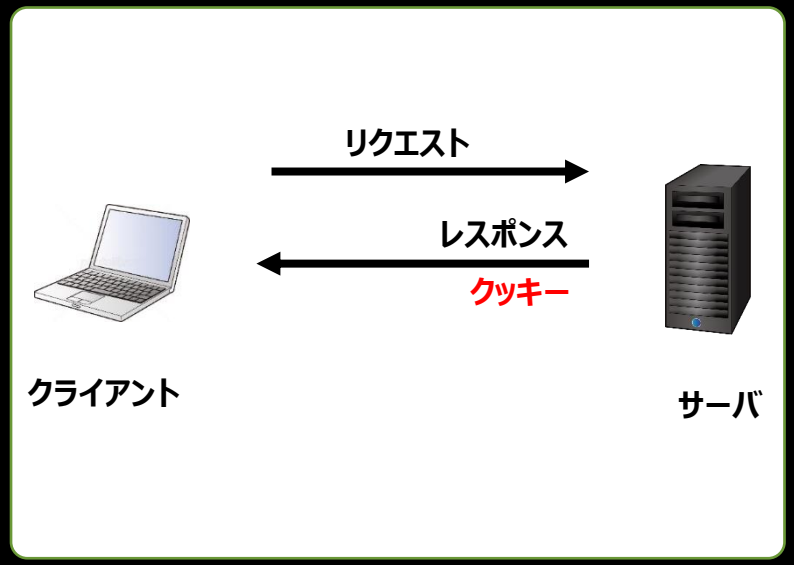


2. Cookie 内容設定

◆ Cookieオブジェクト：サーバで生成し、パラメータを設定する。

◆Cookieコンストラクタ

Cookie (String name, String value)
name　　：クッキー名
value　　：クッキー値



メソッド名	概要
setDomain	クッキーを送信すべき対象となるドメイン。デフォルトは現在のドメイン
setHttpOnly	JavaScriptからのクッキーへのアクセスを拒否
setMaxAge	クッキーの有効期限(秒単位)
setPath	クッキーを送信すべき対象となるパス
setSecure	クッキーをセキュリティ保護環境でのみ送信可とするか
setValue	クッキーの値
setVersion	クッキーのバージョン

2. Cookie 内容設定

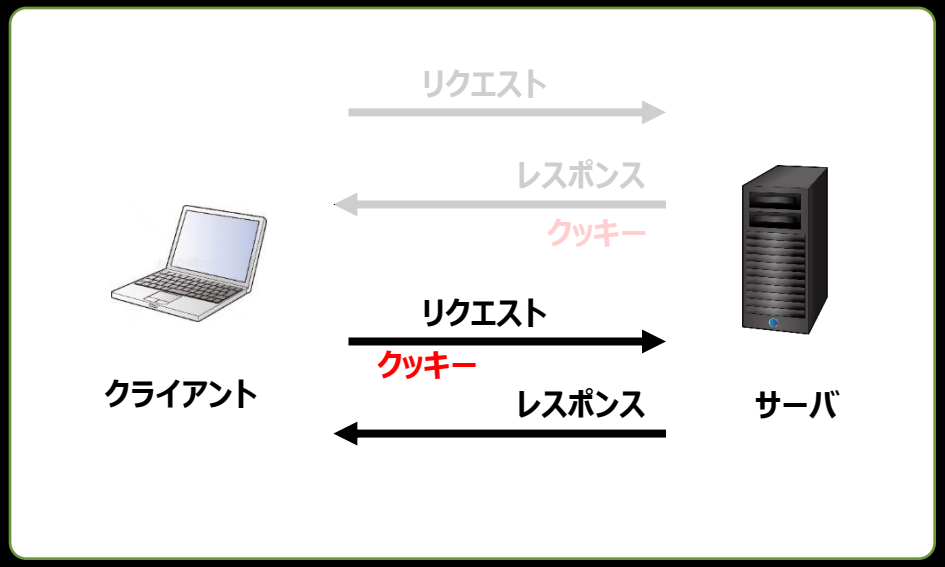
- Domain：クッキーを送信すべき対象となるドメイン(後方一致)
- Path：クッキーを送信すべき対象となるパス(前方一致)

◆Domain

「Domain=example.ac.jp」とすると、example.ac.jpか、そのサブドメイン（例：www.example.ac.jpやapp.example.ac.jp）へアクセスする場合にも送信される。

◆Path

「Path=/」とすると、http://example.ac.jp/loginで取得したCookieはサイト全体で有効になる。（例えばhttp://example.ac.jp/やhttp://example.ac.jp/app2などでも利用される）



3. Cookie 利用例

- リクエストにクッキーがあるか確認し、特定の情報(“email”)があれば、その値を保持
- リクエストに含まれるクッキーは request.getCookies() で取得
- POSTで“Cookie2.jsp”にアクセス

Cookie1.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3
4  <%
5  String email="";
6
7  Cookie[] cookies = request.getCookies();
8
9  if( cookies != null ) {
10         for( Cookie cook : cookies ) {
11                 if(cook.getName().equals("email")) {
12                         email = cook.getValue();
13                         break;
14                 }
15         }
16 }
17 %>
18 <!DOCTYPE html>
19 <html>
20 <head>
21 <meta charset="UTF-8">
22 <title>クッキー情報</title>
23 </head>
24 <body>
25         <form method="POST" action="Cookie2.jsp">
26                 メイルアドレス
27                 <input type="text" name="email" size="40" value="<%=email %>" />
28                 <input type="submit" value="送信" />
29         </form>
30 </body>
31 </html>
```

3. Cookie 利用例

実

- ポストされた値をクッキーとしてクライアントに送信
- レスポンスにクッキーを付けるには respose.addCookie()を使用

Cookie2.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <!DOCTYPE html>
4  <html>
5  <head>
6  <meta charset="UTF-8">
7  <title>クッキー情報</title>
8  </head>
9  <body>
10 <%
11 Cookie cook = new Cookie("email", request.getParameter("email"));
12 cook.setMaxAge( 60*60*24*180 );
13 response.addCookie(cook);
14 %>
15 クッキーが保存されました。
16 </body>
17 </html>
18
```

実

演習09-03　Cookie

1. 下記の2つのJSPファイル setCookie.jsp, getCookie.jsp を作成して下さい。
2. setCookie.jsp を表示した後、getCookie.jspを表示し、クッキーが設定されたことを確認して下さい。
3. 1分以上経過した後、再度、getCookie.jsp を表示し、クッキーが削除されたことを確認して下さい。
4. 2つの確認画面をキャプチャし、キャプチャファイルをSOLAに提出して下さい。

*setCookie.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <%
4  String s = "3424";
5  // set cookie
6  Cookie c = new Cookie( "counter", s);
7  c.setMaxAge( 60 * 1 );  // 1分で削除
8  response.addCookie(c);
9  %>
10 <!DOCTYPE html>
11 <html>
12 <head>
13 <meta charset="UTF-8">
14 <title>クッキーを設定</title>
15 </head>
16 <body>
17 <p>クッキーを設定しました。</p>
18 <p>couter : <%=s %></p>
19 </body>
20 </html>
21
22
```

*getCookie.jsp

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3  <%
4  int count = -1;
5  Cookie[] cookies = request.getCookies();
6  if( cookies != null ) {
7      for( Cookie c : cookies ) {
8          if( c.getName().equals("counter") == true ) {
9              count = Integer.parseInt(c.getValue());
10         }
11     }
12 }
13 %>
14 <!DOCTYPE html>
15 <html>
16 <head>
17 <meta charset="UTF-8">
18 <title>クッキーを取得</title>
19 </head>
20 <body>
21 <p>クッキーを取得</p>
22 <p>count <%=count %></p>
23 </body>
24 </html>
```

JSP 演習

演習09-04　エラトステネスの篩

実

- エラトステネスの篩を用いて素数を求める。
- 求めた素数はテーブルで表示する。
- 10万までの素数を求める。(9592個？)
- JSPで実現する。
- 作成したJSPファイルと実行画面のキャプチャをSOLAに提出する。

エラトステネスの篩(9592)

2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113	127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229	233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349	353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463	467	479	487	491	499	503	509	521	523	541
547	557	563	569	571	577	587	593	599	601	607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733	739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863	877	881	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997	1009	1013	1019	1021	1031	1033	1039	1049	1051	1061	1063	1069
1087	1091	1093	1097	1103	1109	1117	1123	1129	1151	1153	1163	1171	1181	1187	1193	1201	1213	1217	1223
1229	1231	1237	1249	1259	1277	1279	1283	1289	1291	1297	1301	1303	1307	1319	1321	1327	1361	1367	1373
1381	1399	1409	1423	1427	1429	1433	1439	1447	1451	1453	1459	1471	1481	1483	1487	1489	1493	1499	1511
1523	1531	1543	1549	1553	1559	1567	1571	1579	1583	1597	1601	1607	1609	1613	1619	1621	1627	1637	1657
1663	1667	1669	1693	1697	1699	1709	1721	1723	1733	1741	1747	1753	1759	1777	1783	1787	1789	1801	1811
1823	1831	1847	1861	1867	1871	1873	1877	1879	1889	1901	1907	1913	1931	1933	1949	1951	1973	1979	1987

演習09-04　エラトステネスの篩

実

- エラトステネスの篩を用いて素数を求める。
- 求めた素数はテーブルで表示する。
- 10万までの素数を求める。(9592個？)

```
1  <%@ page language="java" contentType="text/html; charset=UTF-7"
2      pageEncoding="UTF-8"%>
3  <%!
4  final int M_SIZE = 100000;
5
6  int             num;
7  int[]    e_num    = new int[M_SIZE];
8  byte[]   map              = new byte[M_SIZE];
9
10 public void CreateMap() {
11         int     i, j;
12
13         if( num > 0 )
14                 return;
15
16         for( i=0; i<M_SIZE; i++ )
17                 map[i] = 1;
18
19
20                                 ▪
21
22                                 ▪
23
24                                 ▪
25
26
27
28 }
29  %>
30 <!DOCTYPE html>
```

演習09-04　エラトステネスの篩

実

- エラトステネスの篩を用いて素数を求める。
- 求めた素数はテーブルで表示する。
- 10万までの素数を求める。(9592個？)

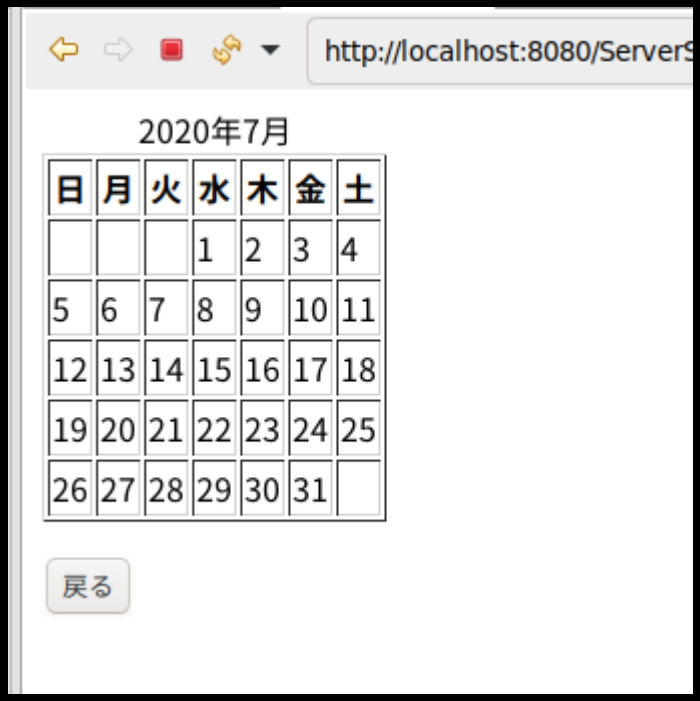
(続き)

```
30  <!DOCTYPE html>
31  <html>
32  <head>
33  <meta charset="UTF-8">
34  <title>エラトステネスの篩</title>
35  </head>
36  <body>
37          <% CreateMap(); %>
38          <table border="1">
39          <caption>エラトステネスの篩(<%=num %>)</caption>
40          <%
41          for( int i=0; i<num; i+=20 ) {
42                  out.print( "<tr>" );
43
44                                          ▪
45
46                                          ▪
47
48                                          ▪
49          }
50          %>
51          </table>
52  </body>
53  </html>
```

演習09-05　カレンダ

実

- 指定した年、月のカレンダを表示する。
- JSPを用いて作成する。
- 作成したJSPファイルとカレンダ表示画面のキャプチャをSOLAに提出する。



演習09-05　カレンダ

実

- 指定した年、月のカレンダを表示する。

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"
3      import="java.util.Calendar" %>
4  <%!
5  private int getDateInteger( int index ) {
6          Calendar calendar = Calendar.getInstance();
7          return calendar.get( index );
8  }
9
10 private int getDayOfWeek( int y, int m ) {
11         Calendar calendar = Calendar.getInstance();
12         calendar.set(y,m,1);
13         return calendar.get(Calendar.DAY_OF_WEEK );
14 }
15
16 private int getDayOfMonth( int y, int m ) {
17         Calendar calendar = Calendar.getInstance();
18         calendar.set(y,m,1);
19         return calendar.getActualMaximum(Calendar.DAY_OF_MONTH );
20 }
21
22 private String createOption(int start, int end, int def_value ) {
23         StringBuilder builder = new StringBuilder();
24         for(int i = start; i <= end; i++) {
25                 if( i==def_value )
26                         builder.append("<option value='" + i + "' selected>" + i + "</option>");
27                 else
28                         builder.append("<option value='" + i + "'>" + i + "</option>");
29         }
30         return builder.toString();
31 }
32 %>
33 <!DOCTYPE html>
```

おわり

サーバシステムプログラミング及び演習

– 第10回 JSP(3) –

前回の学習内容

1. JSPの要素
2. ディレクティブ
3. 宣言部
4. スクリプレット
5. 式 (Expression)
6. アクションタグ

JSP (3)

JSP（3）

1. JDBC
2. コネクションプーリング
3. コンテキスト定義ファイル
4. データベース操作

演習準備 (1)

※今までにDB設定をやっていない人だけ！

- MySQL：データベース作成
- Linuxのコマンドラインで下記のコマンドを実施

```
mysql -udbuser -pdbuser < settings3.sql
```

演習01　DB確認

※全員

実

- DB設定が完了しているか確認する。
- DBの内容を表示する。
- 表示画面をキャプチャし、キャプチャファイルをSOLAに提出する。

```
root@23d66656fc10:/#
root@23d66656fc10:/# mysql -udbuser -pdbuser
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor.  Commands end with ; or ¥g.
Your MySQL connection id is 14
Server version: 8.0.25-0ubuntu0.20.04.1 (Ubuntu)

Copyright (c) 2000, 2021, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '¥h' for help. Type '¥c' to clear the current input statement.

mysql> select * from shop_db.list_t;
+------+----------------------------------------+----------------------------------------+
| id_c | name_c                                 | add_c                                  |
+------+----------------------------------------+----------------------------------------+
|    1 | 料亭たなか                                | 東京都葛飾区何処町1-2                            |
|    2 | 洋風居酒屋アベリタ                            | 神奈川県横浜市中区何処町5-2                          |
|    3 | 炭火焼肉田中屋                               | 広島県広島市何処町3-1                            |
|    4 | 無国籍料理トランス                            | 大阪府大阪市中央区何処町2-3                          |
|    5 | 本格寿司あべ膳                               | 北海道札幌市何処町2-4                            |
|    6 | 台湾料理とらんす                              | 岩手県盛岡市何処町5-1                            |
|    7 | 甘味処ナルミ亭                               | 埼玉県さいたま市大宮区何処町4-1                        |
|    8 | 南欧家庭料理タナカッターラ                         | 佐賀県佐賀市何処町3-5                            |
|    9 | うまいものや阿部亭                             | 秋田県秋田市何処町4-3                            |
|   10 | インド料理トランース                            | 東京都葛飾区何処町1-6                            |
+------+----------------------------------------+----------------------------------------+
10 rows in set (0.00 sec)

mysql> quit
Bye
root@23d66656fc10:/#
```

RDBMS MySQL

- オープンソースのリレーショナルデータベース管理システム(RDBMS)
- SQLで操作 (Structured Query Language ? != プログラミング言語)

注文DB

注文ID	顧客ID	製品ID	数量	納期
1	3	11	1	2015年9月30日
2	1	3	38	2015年10月28日
3	2	8	60	2015年11月20日

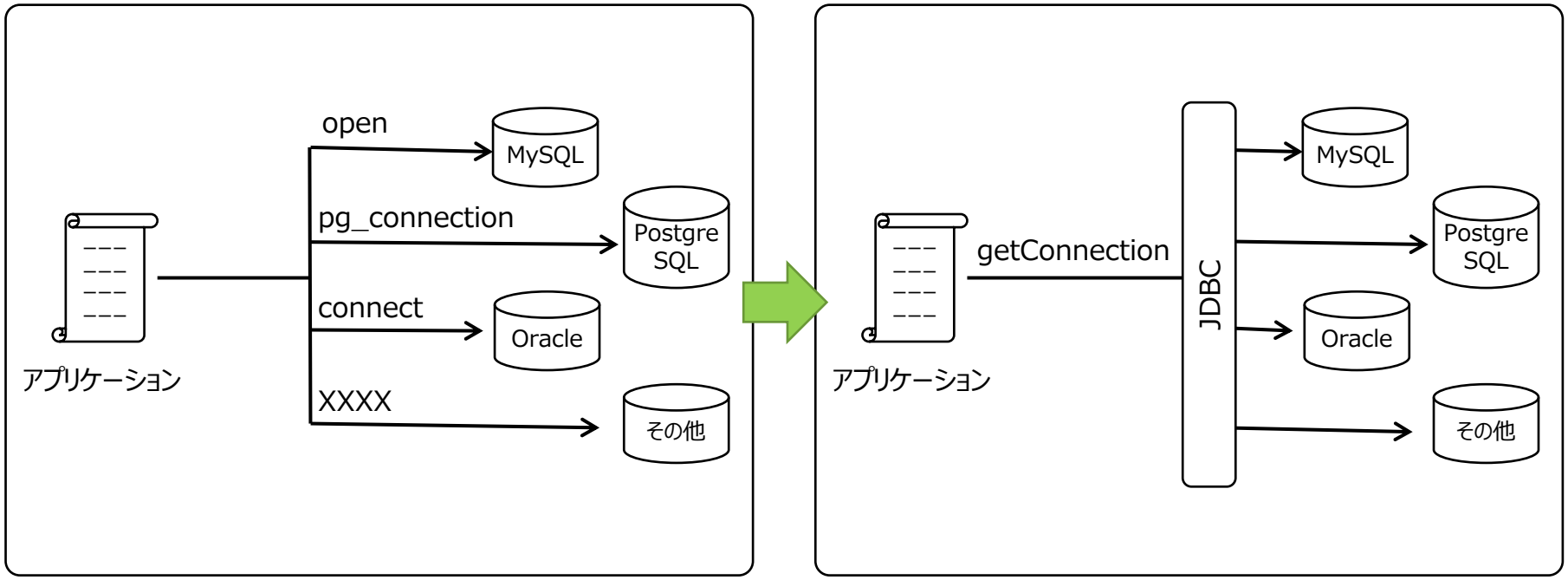
製品DB

顧客DB

顧客ID	氏名	住所	年齢	e-mail
1	麻生 太郎	東京都渋谷区神山町	74	asou@ed.tus.ac.jp
2	小泉 進次郎	神奈川県横須賀市	38	koizumi@ed.tus.ac.jp
3	安倍 晋三	東京都渋谷区富ヶ丘	60	abe@ed.tus.ac.jp

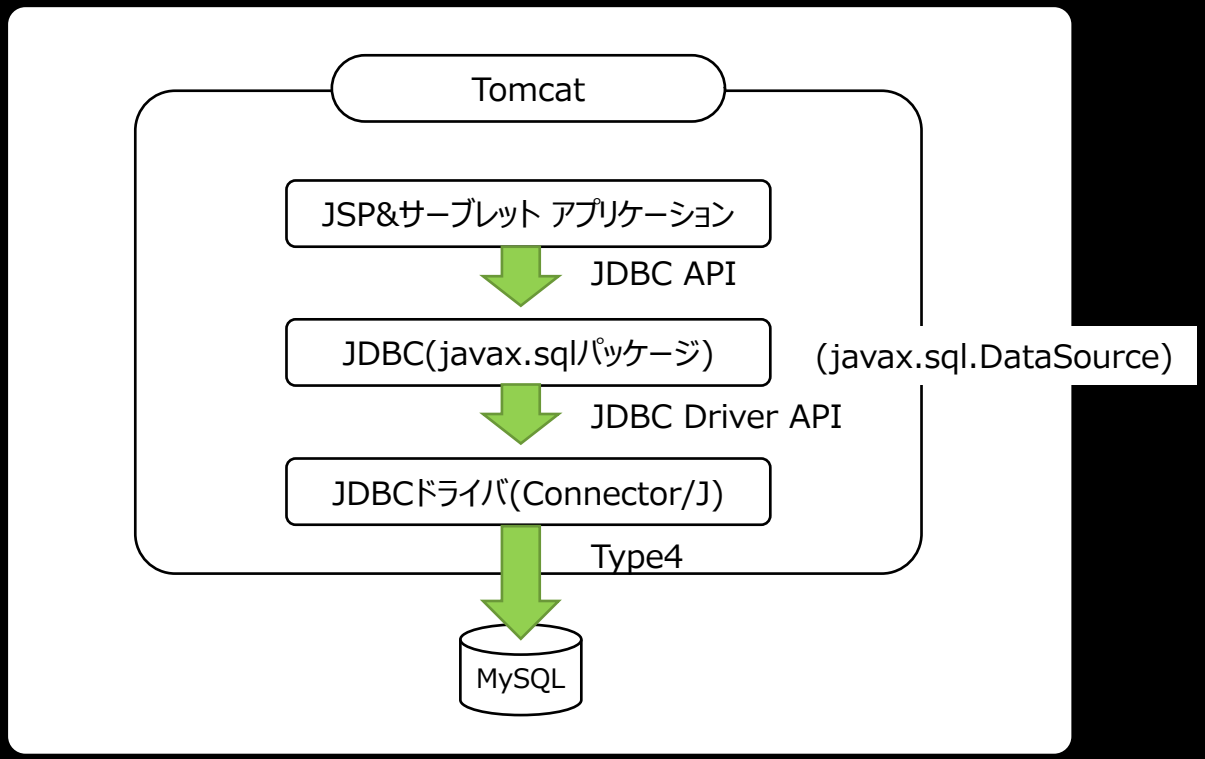
JDBC

- JDBC：Java Database Connectivity
- Javaと関連データベースの接続のためのAPI
- RDBMSなどと接続する機能を標準化
- インターフェイスの異なるDBに対し同一のプログラムで対応可
- MySQLにも対応



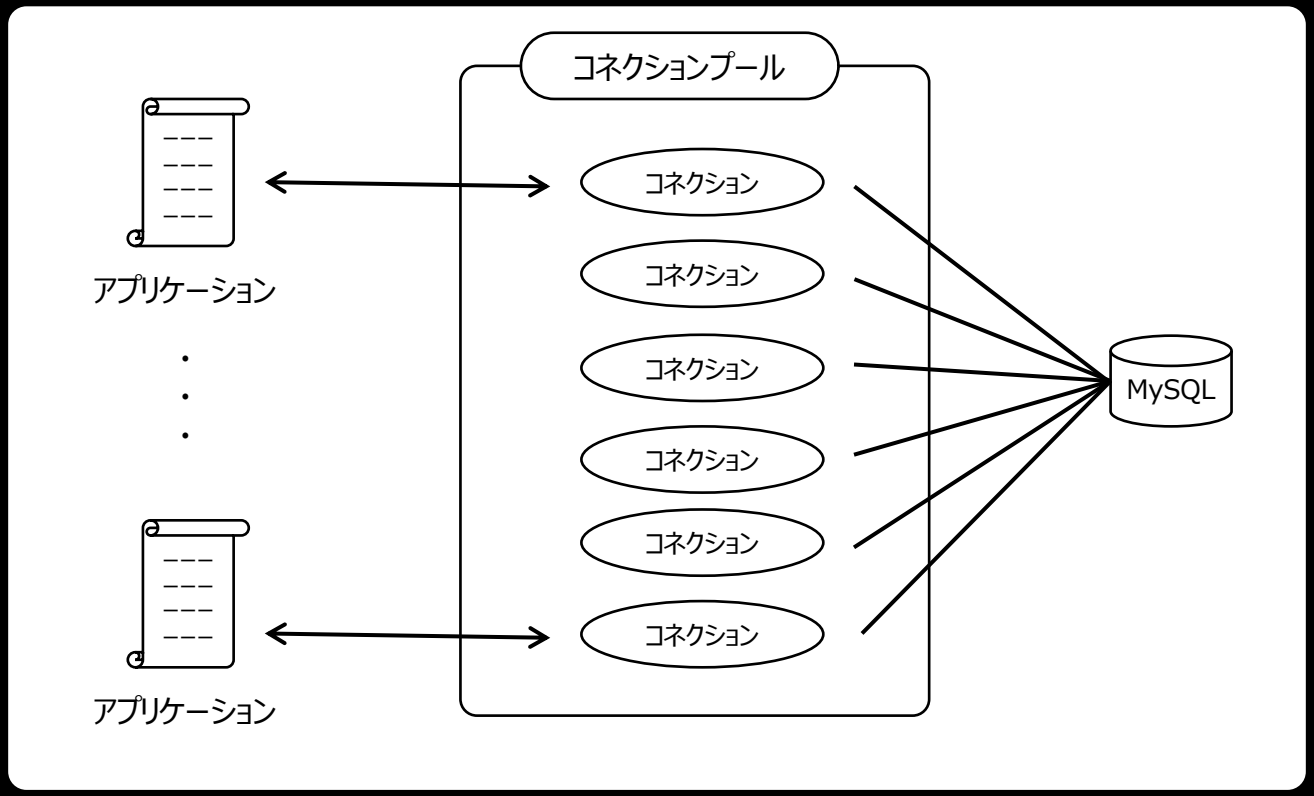
JDBC　構成

- 複数のライブラリの集合体
- Java SE の java.sql パッケージとして提供されるAPI　ただし、インターフェイス(メソッド名の定義のみ)
- データベースへの接続や操作などの実態は、DBごとに用意されたJDBCドライバー
- MySQL　のJDBCドライバ　：　Connector/J　(インストール済)



コネクションプーリング

- 不特定多数のユーザが同時に、絶え間なくデータベースにアクセス
  ⇒　データベースへの接続処理が増大　⇒　リソース消費
- コネクションプーリング：接続をプールしておくための仕組み
- あらかじめ用意されたプールから接続を取り出し使用　!= 要求の度に接続処理



コネクションプーリング　設定(コンテキスト定義)

- データソース(javax.sql.DataSource)：コネクションプーリングを管理する一連の手続きを規定
- インターフェイスのみ
- データソースの接続情報はコンテキスト定義ファイル(context.xml)で設定

プロパティ	概要
name	データソースを一意に識別する名前
auth	リソースの制御方法。データソース利用の場合、「Container」
type	リソースのデータ型。データソース利用の場合、「javax.sql.DataSource」
driverClassName	JDBCドライバクラスの完全限定名
url	データソースURL (接続文字列)
username	接続の際のユーザ名
password	接続の際のパスワード
maxActive	プーリングする最大接続数
maxIdle	待機時に最低限維持する接続数
maxWait	接続に対しての最大待ち時間　(単位：ミリ秒)
validationQuery	接続検証用のSQL命令

context.xml

```
1  <Context displayName="JSP shop db" docBase="shop_db" path="/shop_db" reloadable="true">
2          <Resource name="jdbc/shop_db"
3                  auth="Container"
4                  type="javax.sql.DataSource"
5                  username="dbuser" password="dbuser"
6                  driverClassName="org.gjt.mm.mysql.Driver"
7                  url="jdbc:mysql://localhost/shop_db?useUnicode=true&amp;useSSL=false&amp;characterEncoding=UTF-8"
8                  maxActive="4" maxWait="5000" maxIdle="2"
9                  validationQuery="SELECT count(*) schedules" />
10 </Context>
11
```

コンテキスト定義ファイル

- Context.xml
- Tomcatの設定ファイル：JSP、サーブレットアプリケーション全般の設定を定義
- Tomcat固有の機能について規定
- 配置場所によって設定が影響を与える範囲が異なる。

配置場所	適用範囲
%CATALINA_HOME%/conf/server.xml	全てのアプリケーション(非推奨)
%CATALINA_HOME%/conf/context.xml	全てのアプリケーション
%CATALINA_HOME%/<エンジン名>/<ホスト名>/context.xml.default	指定ホスト配下の全てのアプリケーション
%CATALINA_HOME%/<エンジン名>/<ホスト名>/<アプリケーション名>.xml	指定されたアプリケーション
%CATALINA_HOME%/<アプリケーション名>/META-INF/context.xml	アプリケーション内のみ

※CATALINA_HOME = /opt/tomcat

データベース接続・操作の流れ

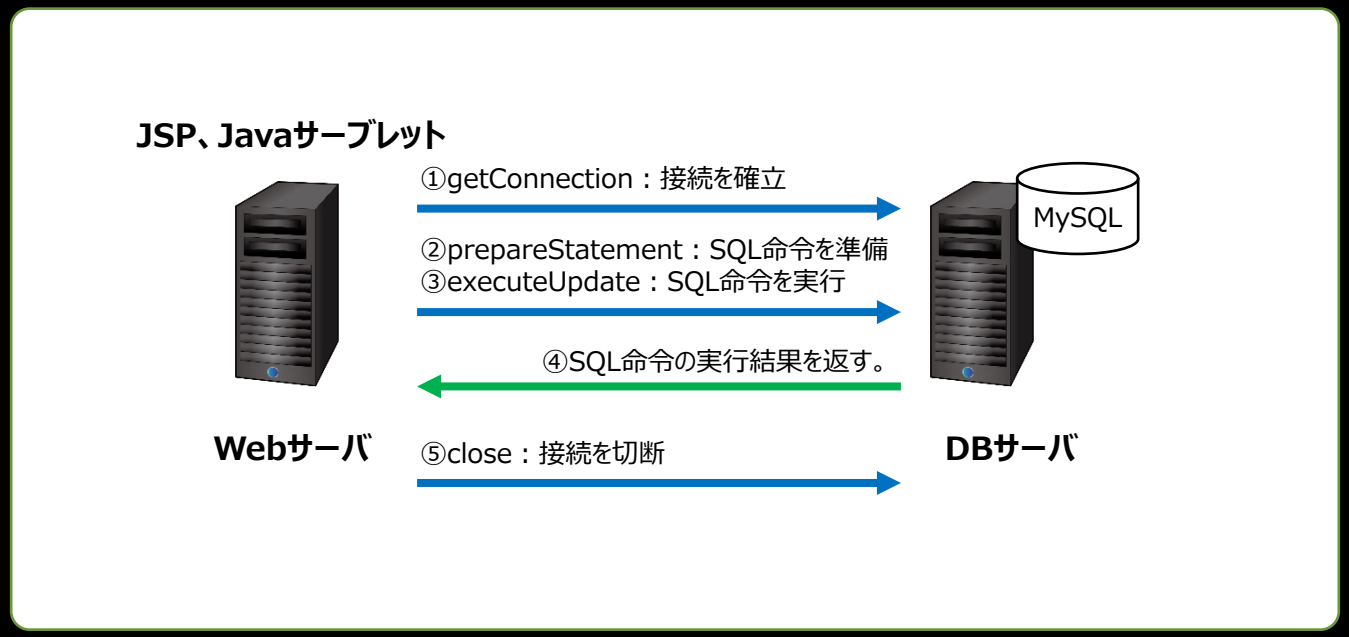
① DBサーバとの接続を確立

② SQL命令を準備

③ SQL命令を実行

④ SQL命令実行結果を受け取り

⑤ 接続を切断



データベース接続

- 登録したデータソースを取得するには、InitialContextクラスの lookup メソッドを使用
- lookup メソッドの引数：　“java:comp/env/<データソース名>
- lookup メソッドの戻り値：　Object　⇒　DataSourceオブジェクトに型変換
- DataSourceオブジェクト取得後、getConnectionメソッドで接続を確立

※ コネクションをプールから取得した場合、**必ずcloseで接続をプールに戻す。**

```
4
5   // 登録したデータソースを利用するには、InitialContextクラス(javax.namingパッケージ)
6   // のlookupメソッドを使用する。
7   Context context = new InitialContext();
8   DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shopdb");
9   Connection db = ds.getConnection();
10
```

データベース操作（レコード検索）

- SQL命令の管理：PreparedStatementオブジェクト
- PreparedStatementオブジェクト：Connectionオブジェクトdbから preparedStatementメソッドを呼び出すことで取得
- preparedStatementメソッドの？：プレイスホルダー
- setStringメソッドでプレイスホルダーに文字列を設定
- 結果を返すSQL文の実行　⇒　executeQueryメソッド
- executeQuery：①SQL命令の実行　②結果をResultSetオブジェクトに設定

```
41      /*********************************************
42       *
43       *  MySQLへクエリ送信
44       *
45       *********************************************/
46      PreparedStatement ps = db.prepareStatement( "SELECT * FROM list_t WHERE id_c = ? " );
47      ps.setString( 1, idStr );
48      ResultSet rs = ps.executeQuery();
49
50      /*********************************************
51       *
52       *  表示処理
53       *
54       *********************************************/
55      if( rs.next() ) {
56  %>
57          <p>テーブルデータを参照します。</p>
58          <table border="1">
59          <tr>
60              <td>ID</td>
61              <td>店名</td>
62              <td>住所</td>
63              </tr>
64
65  <%
66          do {
67
```

データベース操作（レコード検索）

- ResultSet.nextメソッド：レコード単位に結果を読み込む。読み込みに成功⇒true、失敗⇒false
- ResultSet.getXxxxメソッド：読み込んだレコードから指定されたフィールドの値を取得する。Xxxxの部分：Boolean、Byte、Double、Float、Int、Long、Short、String、Timeのいずれか

```
48      ResultSet rs = ps.executeQuery();
49
50      /*********************************************
51       *
52       *  表示処理
53       *
54       *********************************************/
55      if( rs.next() ) {
56  %>
57          <p>テーブルデータを参照します。</p>
58          <table border="1">
59          <tr>
60              <td>ID</td>
61              <td>店名</td>
62              <td>住所</td>
63              </tr>
64
65  <%
66          do {
67
68          out.print( "<tr><td>" + rs.getInt("id_c") + "</td><td>"
69          + rs.getString("name_c") + "</td><td>"
70          + rs.getString( "add_c" ) + "</td></tr>" );
71
72          } while( rs.next() );
73
74  %>
```

データベース操作（レコード追加）

- PreparedStatemet.executeUpdateメソッドでSQL命令を実行

```
30  /*********************************************
31   *
32   *  MySQLと接続
33   *
34   *********************************************/
35  // 登録したデータソースを利用するには、InitialContextクラス(javax.namingパッケージ)のlookupメソッドを使用する。
36  Context context = new InitialContext();
37  DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
38  Connection db = ds.getConnection();
39
40
41  /*********************************************
42   *
43   *  MySQLへクエリ送信
44   *
45   *********************************************/
46  PreparedStatement ps = db.prepareStatement( "INSERT INTO list_t (name_c,add_c) VALUES ( ?, ? )" );
47  ps.setString( 1, nameStr );
48  ps.setString( 2, addStr );
49  ps.executeUpdate();
50
51  ps.close();
52  db.close();
```

データベース操作（レコード削除）

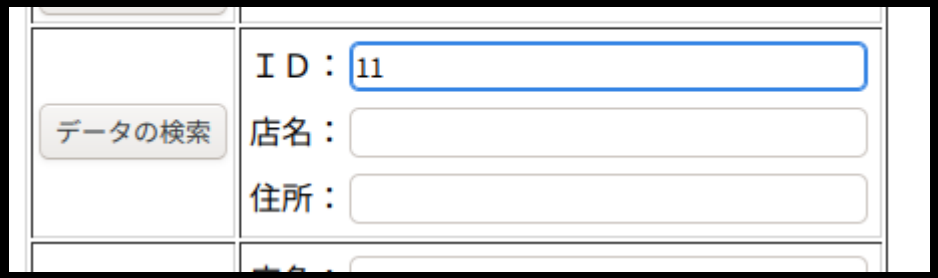
- レコードの追加と同じ

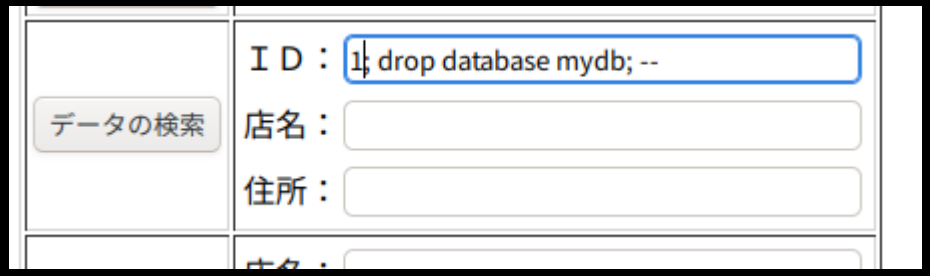
```
29      /**********************************************
30       *
31       *  MySQLと接続
32       *
33       **********************************************/
34       // 登録したデータソースを利用するには、InitialContextクラス(javax.namingパッケージ)
35      // のlookupメソッドを使用する。
36      Context context = new InitialContext();
37      DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
38      Connection db = ds.getConnection();
39
40
41      /**********************************************
42       *
43       *  MySQLへクエリ送信
44       *
45       **********************************************/
46      PreparedStatement ps = db.prepareStatement( "DELETE FROM list_t WHERE id_c=? " );
47      ps.setString( 1, idStr );
48      ps.executeUpdate();
49
50      ps.close();
51      db.close();
```

SQLインジェクション

- 入力にSQL命令を記述することで任意の命令を実行させる。
- 内容を確認するなどして対応
- プレイスホルダ　：　命令とパラメータを分けて渡す。

　　⇒　SQLサーバではパラメータにSQL命令があっても「値」して処理。命令を実行しない。





SELECT * FROM list_t WHERE id_c = **11**

SELECT * FROM list_t WHERE id_c = **1; drop database mydb; --**

```
ps = db.prepareStatement( "SELECT * FROM list_t WHERE id_c = ? " );
ps.setString( 1, idStr );
```

演習

演習準備 (2)

実

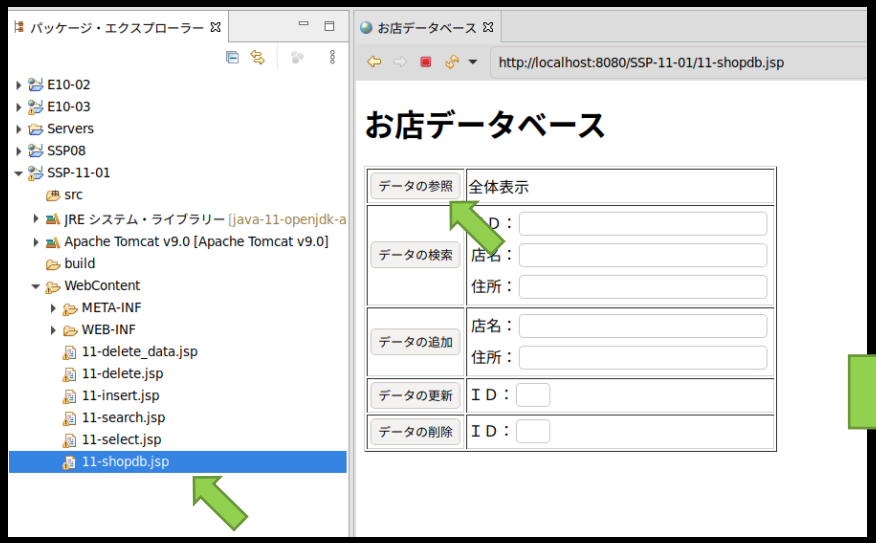
- JSPファイルをダウンロード　(プロジェクト名：SSP-11-01)
- 環境に合わせてアプリケーション名等の変更が必要
- コンテキスト定義ファイル(context.xml)

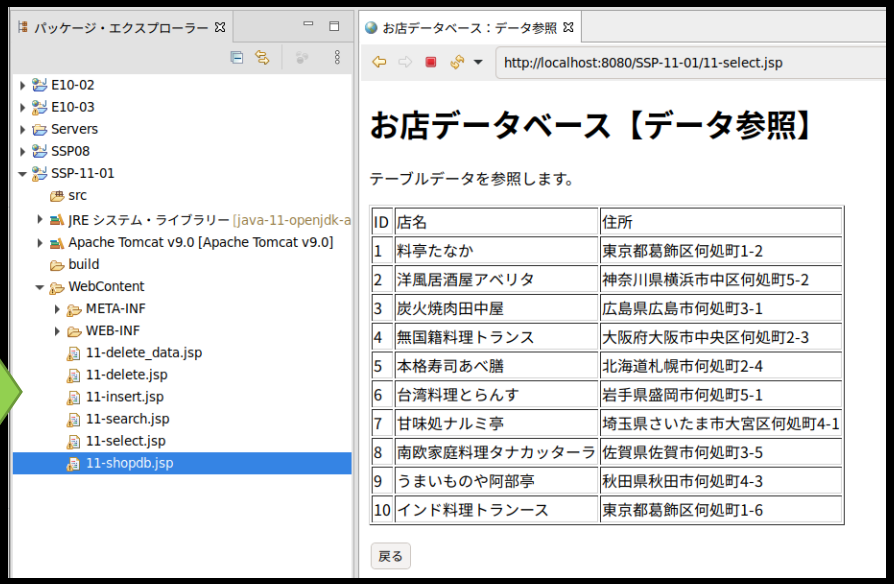
　　⇒　SSP-11-01/WebContent/META-INF

演習02　ファイルの確認

実

- 11-shopdb.jspを表示
- “データの参照”をクリックし、お店データベースを表示
- 画面をキャプチャし、キャプチャファイルをSOLAに提出



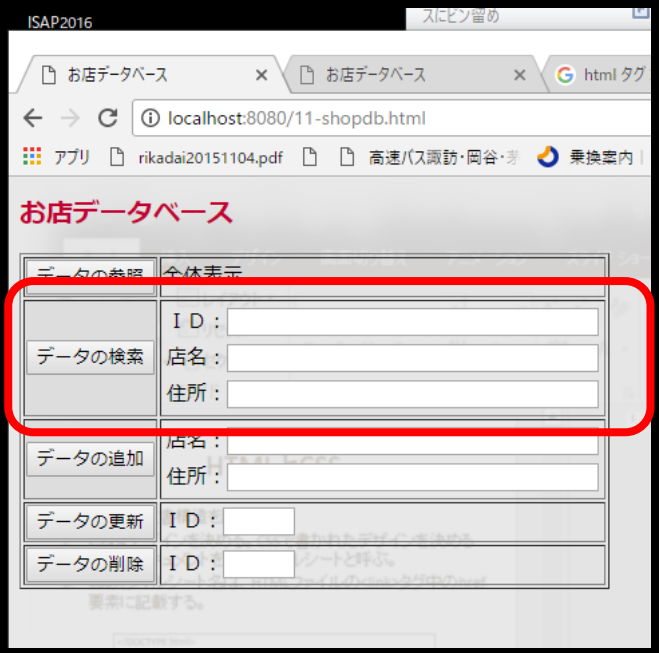


演習03　DBレコード検索

実

- “お店データベース”を完成させる。
- 11-search.jsp を作成する。
- 検索を実施し、実施画面をキャプチャする。
- キャプチャファイルをSOLAに提出

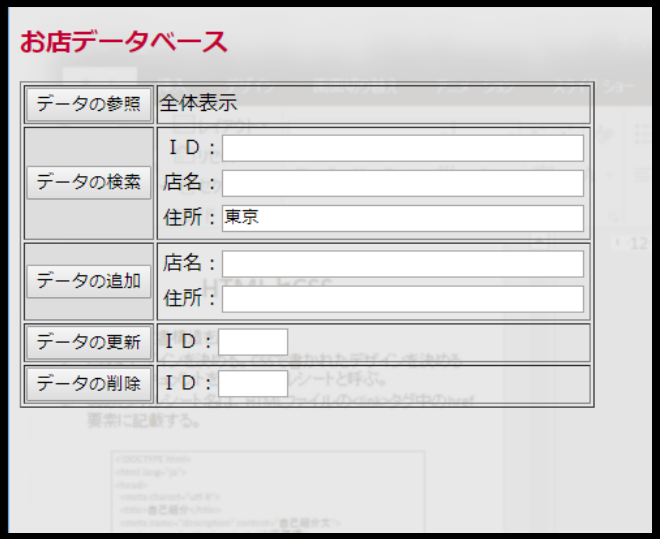
機能	ファイル
メイン	11-shopdb.jsp
データの参照	11-select.jsp
データの検索	**11-search.jsp**
データの追加	11-insert.jsp
データの変更	11-update.jsp ⇒ 11-update_data.jsp
データの削除	11-delete.jsp ⇒ 11-delete_data.jsp

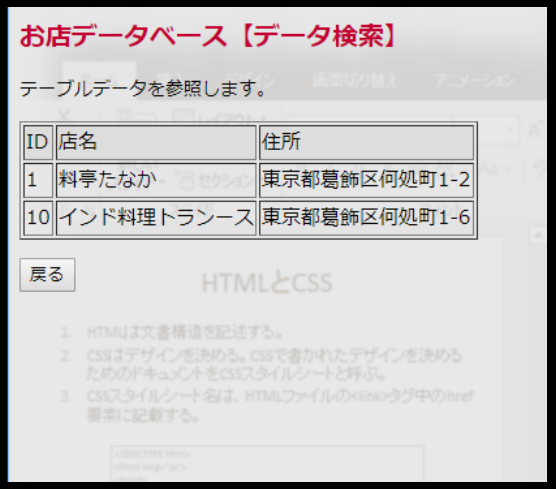


演習03　DBレコード検索

実

- “お店データベース”を完成させる。
- 11-search.jsp を作成する。
- 検索を実施し、実施画面をキャプチャする。
- キャプチャファイルをSOLAに提出





演習03　DBレコード検索

実

- “お店データベース”を完成させる。
- 11-search.jsp を作成する。

11-search.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"
    import="java.sql.*, javax.naming.*, javax.sql.*" %>
<!DOCTYPE html>
<html lang="ja">

<head>
<meta charset="utf-8">
<title>お店データベース：データ検索</title>
<meta name="description" content="shop data base">
<meta name="author" content="理科大太郎">
</head>

<body>

<h1>お店データベース【データ検索】</h1>

<%
request.setCharacterEncoding("UTF-8");

String idStr = request.getParameter("id");
String nameStr = request.getParameter("name");
String addStr = request.getParameter("add");

if( idStr.length() == 0 && nameStr.length() == 0 && addStr.length() == 0  ) {
    out.print("<p>検索条件を入力して下さい。</p>\n");
} else {


```

```java
/*********************************************
 *
 *  MySQLと接続
 *
 *********************************************/
 // 登録したデータソースを利用するには、InitialContextクラス(javax.namingパッケージ)のlookupメソッドを使
Context context = new InitialContext();
DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
Connection db = ds.getConnection();


/*********************************************
 *
 *  MySQLへクエリ送信
 *
 *********************************************/
 PreparedStatement ps;
if( idStr.length() != 0 ) {
    ps = db.prepareStatement( "SELECT * FROM list_t WHERE id_c = ? " );
    ps.setString( 1, idStr );
} else {
    ps = db.prepareStatement( "SELECT * FROM list_t WHERE name_c LIKE ? AND add_c LIKE ?" )
    ps.setString( 1, "%"+nameStr+"%" );
    ps.setString( 2, "%"+addStr+"%" );
}
ResultSet rs = ps.executeQuery();
```

演習03　DBレコード検索

実

- “お店データベース”を完成させる。
- 11-search.jsp を作成する。

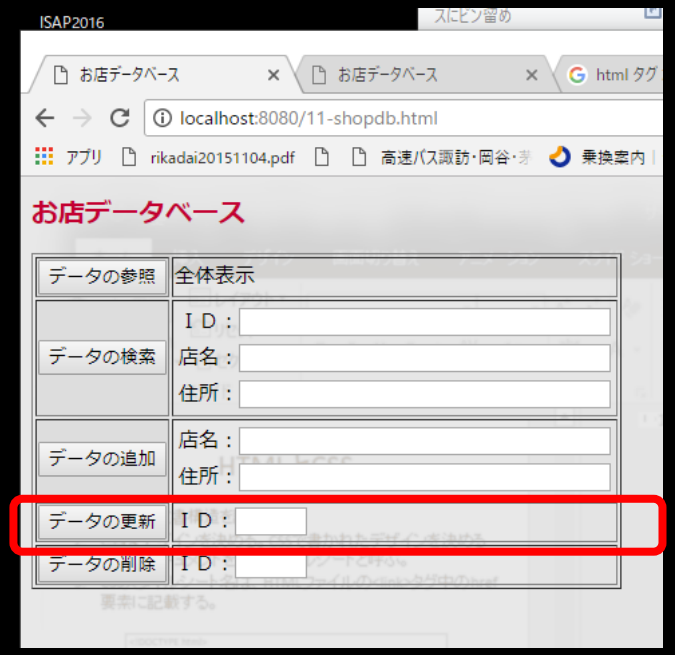
```
56
57      /*********************************************
58       *
59       *  表示処理
60       *
61       *********************************************/
62      if( rs.next() ) {
63  %>
64          <p>テーブルデータを参照します。</p>
65          <table border="1">
66          <tr>
67              <td>ID</td>
68              <td>店名</td>
69              <td>住所</td>
70              </tr>
71
72  <%
73          do {
74
75          out.print( "<tr><td>" + rs.getInt("id_c") + "</td><td>"
76          + rs.getString("name_c") + "</td><td>"
77          + rs.getString( "add_c" ) + "</td></tr>" );
78
79          } while( rs.next() );
80
81  %>
82           </table>
83  <%
84      } else {
85              out.print("<p>該当するデータが見つかりませんでした。</p>");
86      }
87
88      rs.close();
89      ps.close();
90      db.close();
91  }
92  %>
93  <p><a href="/SSP-11-01/11-shopdb.jsp"><button type="button" >戻る</button></a></p>
94
95
96  </body>
97  </html>
98
```

演習04　DBレコード変更

実

- “お店データベース”を完成させる。
- 11-update.jsp、11-update_data.jsp を作成する。
- 作成したjspファイルをSOLAに提出する。

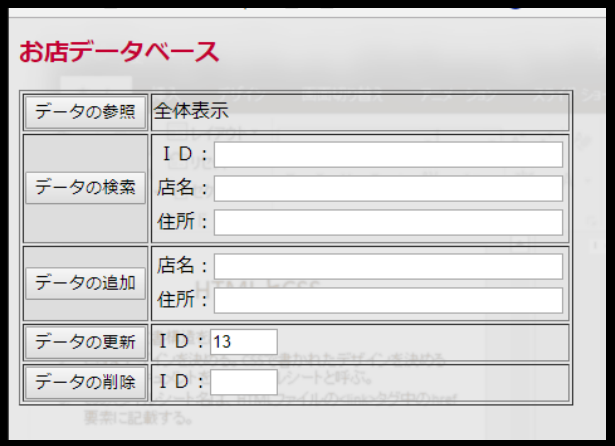
機能	ファイル
メイン	11-shopdb.jsp
データの参照	11-select.jsp
データの検索	11-search.jsp
データの追加	11-insert.jsp
データの変更	**11-update.jsp ⇒ 11-update_data.jsp**
データの削除	11-delete.jsp ⇒ 11-delete_data.jsp

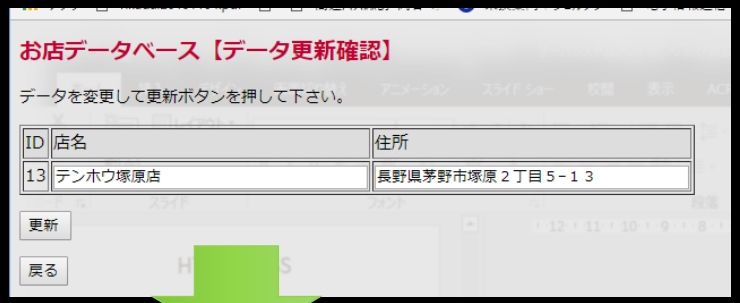


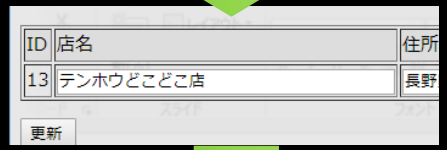
演習04　DBレコード変更

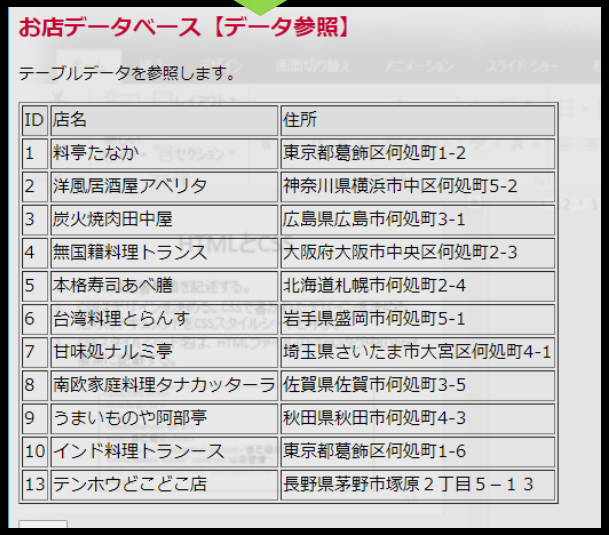
実

- “お店データベース”を完成させる。
- 11-update.jsp、11-update_data.jsp を作成する。
- 作成したjspファイルをSOLAに提出する。









おわり

サーバシステムプログラミング及び演習

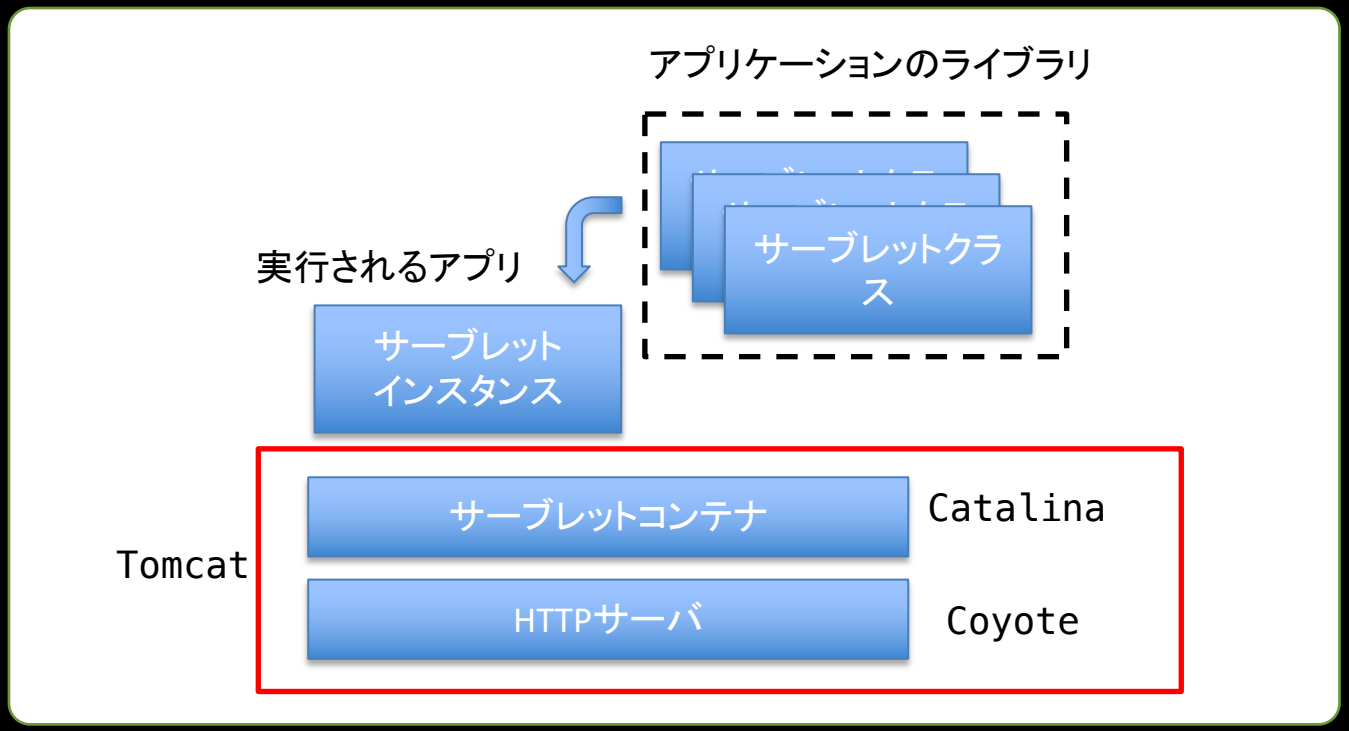
– 第11回 Servlet –

本日の内容

- Javaサーブレットはサーバ上で動作するJavaプログラム。動的Webアプリケーションを構築するためのサーバサイド技術
- Javaサーブレットはサーブレットコンテナ上で実行される
- Javaサーブレットのライフタイムは、最初にリクエストを受けてからコンテナが終了するまで
- Javaサーブレットは「シングルインスタンス・マルチスレッド」で実施される。
- JSPがHTMLに埋め込むJavaコードであるのに対し、JavaサーブレットはJavaでHTMLを出力
- Javaサーブレットはロジックの記述が得意。HTMLを大量に出力するのは苦手
- JavaとJSPで役割分担を行い、メンテナンスしやすいアプリケーションを実現
- JavaサーブレットのアプリケーションクラスはHttpServletを継承
- Httpメソッドのハンドラ：doGet()、doPost()
- Javaサーブレットでは、JSPのような暗黙のオブジェクトは存在しない。⇒インスタンス化が必要
- Tomcatでのファイル構成：Javaサーブレットのファイルは、アプリケーション＞WEB-INF＞classesの配下にパッケージに従って配置
- Javaサーブレットでのパラメータの受け取りはrequest.getParameter("XXX") を使用
- JavaサーブレットからJSPへのフォワード：getServletContext().getRequestDispathcer(転送パス).forward(HttpServletRequestオブジェクト、 HttpServletResponseオブジェクト)
- インスタンス変数は保持される。
- ライブラリが使えます。
- JavaBeans：汎用的に部品化できるよう幾つかの条件を満たしたクラス　⇒部品化のためのクラス
- JSP、Javaサーブレット、JavaBeansで役割分担　⇒メンテナンスしやすいアプリケーション構成

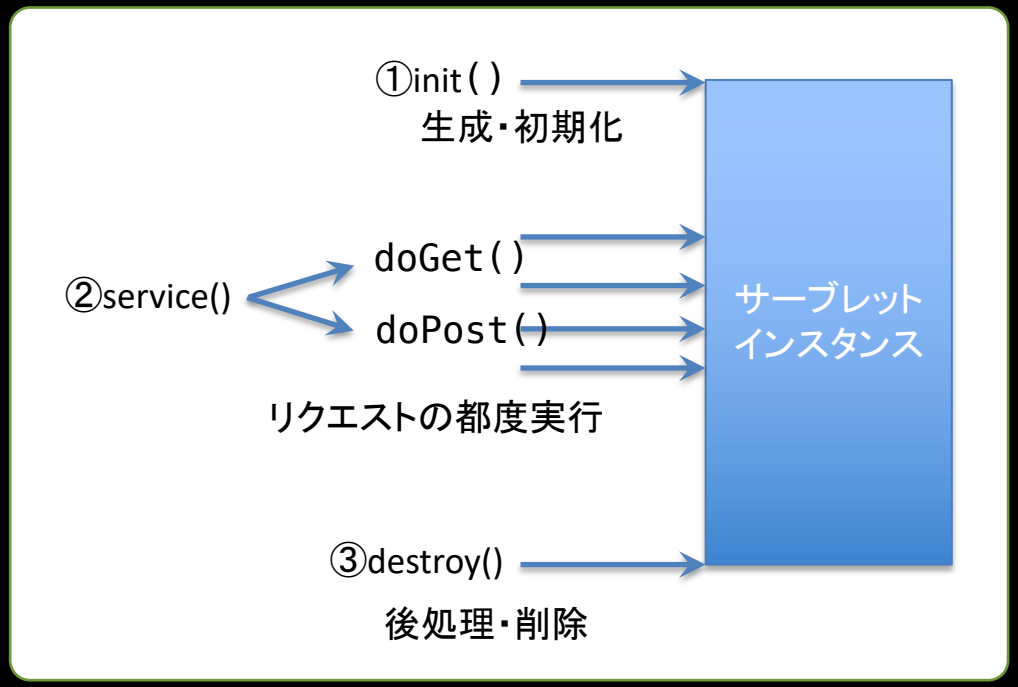
2. サーブレットコンテナ　Apache Tomcat

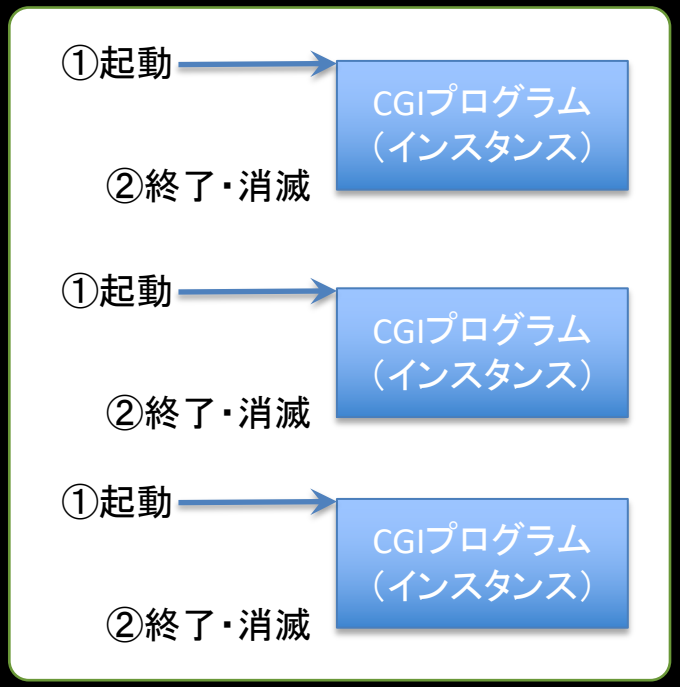
- サーブレットコンテナ：　Javaサーブレットの実行環境
- Apache Tomcat：Apache Software Foundationが開発しているオープンソースのサーブレットコンテナ
- Apache Tomcatはコンテナと HTTPサーバから構成　(コンテナ：Catalina、HTTPサーバ：Coyote　)
- Apache HTTP Serverとも連携可



サーブレットのライフタイム

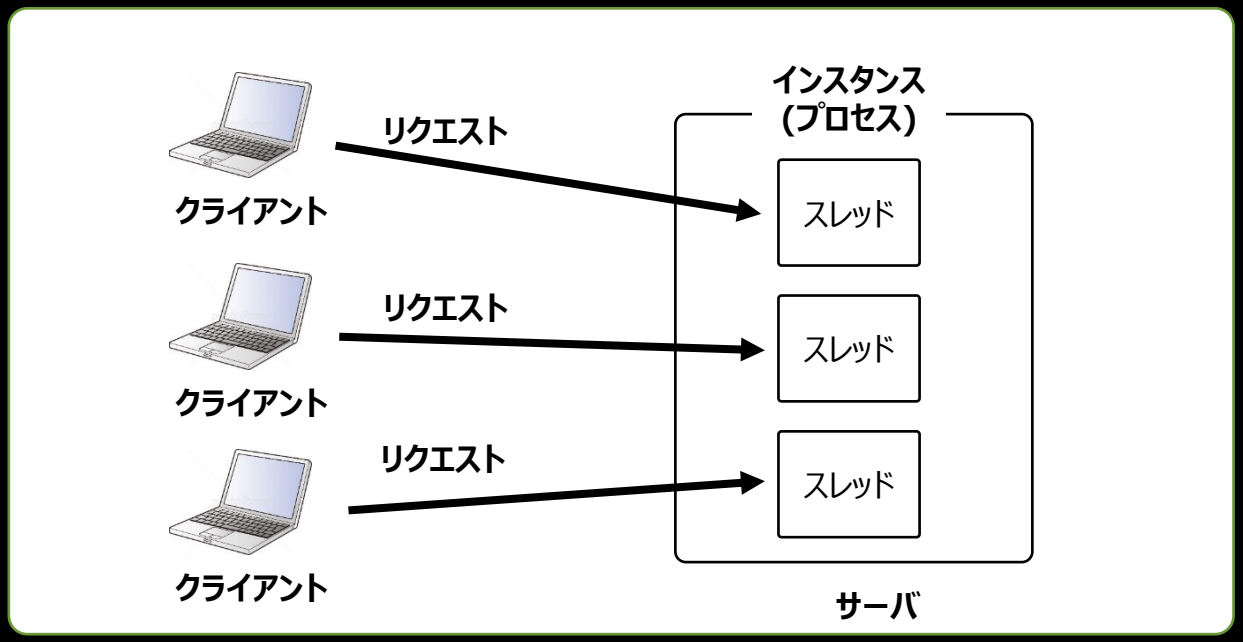
- サーブレットのライフタイム：init()からdestroy()まで
- ライフタイム中は複数のリクエストに対しサービスを提供　(cf.　CGI：リクエストの都度インスタンスが生成⇒消滅)
- サービスは並列実行可能　(要注意)





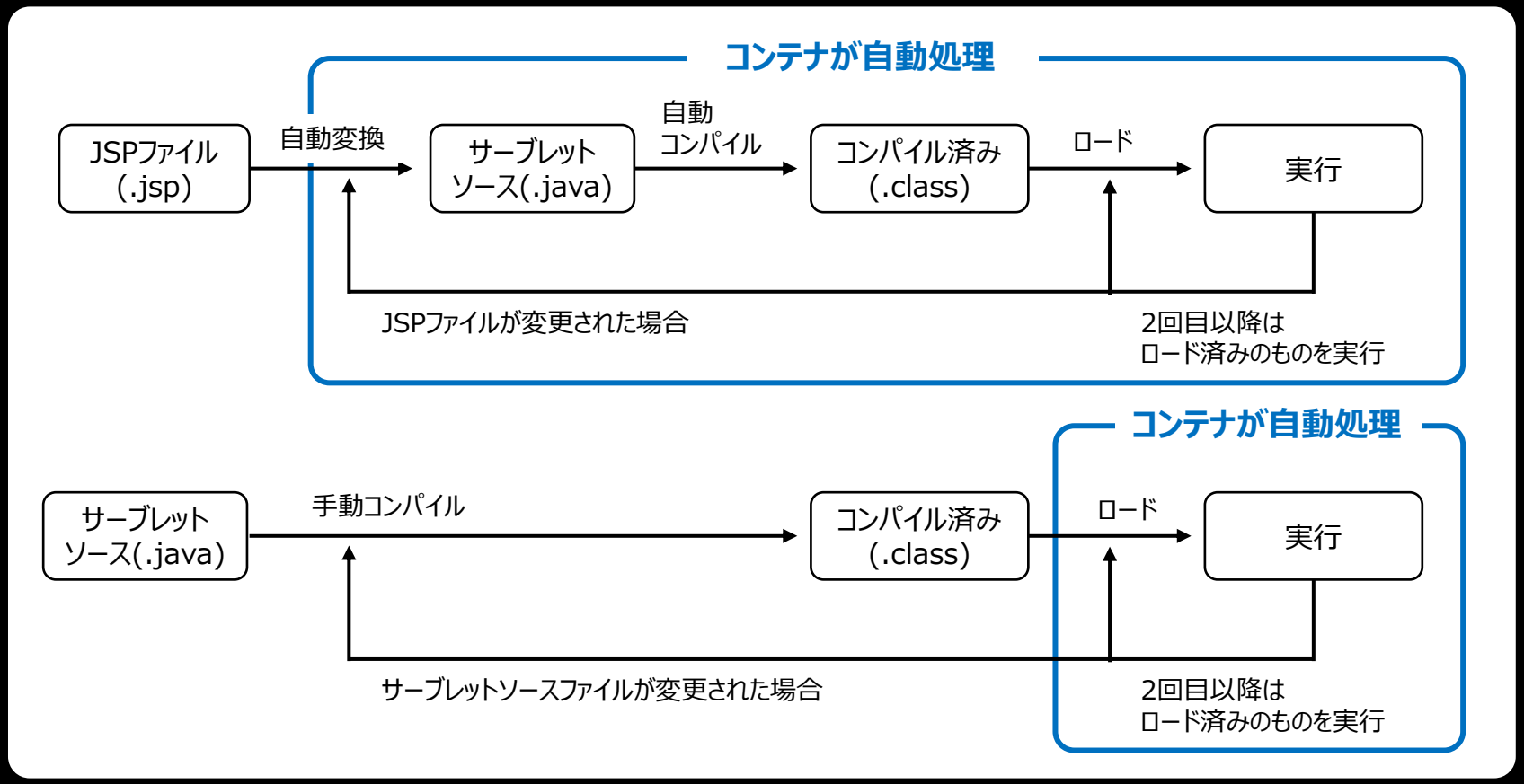
シングルインスタンス・マルチスレッド

- サーブレットは一度インスタンス化されると常駐する　(≠リクエストの度にインスタンス化　)
- 同じサーブレットに対するリクエストは同じインスタンスで処理される。
- 複数の処理がマルチスレッドで実行される。(※スレッド：プロセス内に作られる並列動作が可能な処理の単位)



3．サーブレットとJSP

- Java Servlet
- 動的Webアプリケーションを構築するためのサーバサイド技術
- JSPがHTMLに埋め込むJavaコードであるのに対し、サーブレットはJavaでHTMLを出力



Servlet

Java servlet

1. サーブレットコンテナ上で動作するJavaのプログラムです。
2. コンテナで動作させるには手動でコンパイルする必要があります。
   (が、Eclipseを使用している場合、自動でやってくれます。)
3. サーブレットはロジックの記述が得意
4. サーブレットはHTMLを大量に出力するのが苦手

```
29          /**
30           * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
31           */
32          protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
            IOExcept
33  ion {
34              // TODO Auto-generated method stub
35              //response.getWriter().append("Served at: ").append(request.getContextPath());
36
37              counter++;
38
39              response.setContentType( "text/html; charset=UTF-8" );
40              PrintWriter out = response.getWriter();
41
42              out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
43              out.println( "<title>Counter</title>" );
44              out.println( "</head>" );
45              out.println( "<body>" );
46              out.println( "アクセス数は" + counter + "です。</p>" );
47              out.println( "</body>" );
48              out.println( "</html>" );
49
50              out.close();
51          }
52
```

サーブレットの必要性

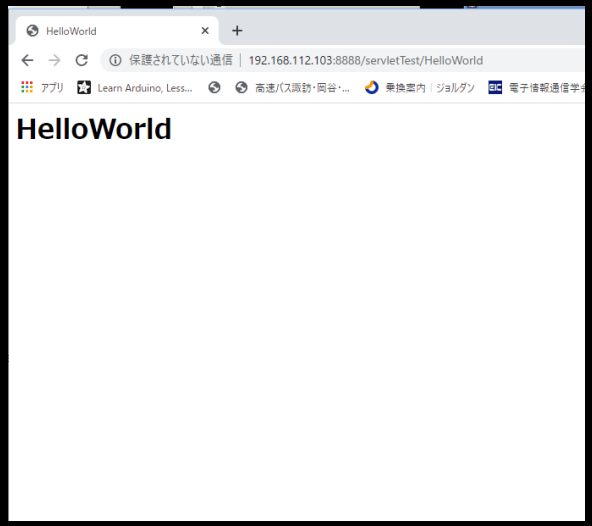
- アプリケーションの構築　⇒　「ただ動けばよい」というものではない。
- メンテナンスのしやすさ、可読性が重要
- JSPで、込み入ったロジックを技術した場合、非常に読みづらい、可読性の低いコードとなる。
- サーブレットとJSPはお互いに補完し合う関係にある
- 役割分担により整理された構成を実現
- 役割分担：サーブレット　⇒　ロジック、　JSP　⇒　HTML

サーブレットの例

- “HelloWorld” と表示するだけのサーブレット
- GETメソッドのハンドラ：doGet()、　POSTメソッドのハンドラ：doPost()

HelloWorld.java

```java
package servletTest;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class HelloWorld
 */
@WebServlet("/HelloWorld")
public class HelloWorld extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public HelloWorld() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                //response.getWriter().append("Served at: ").append(request.getContextPath());

               response.setContentType("text/html; charset=utf-8");
                PrintWriter out = response.getWriter();
                out.println("<html>");
                out.println("<head>");
                out.println("<title>HelloWorld</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>HelloWorld</h1>");
                out.println("</body>");
                out.println("</html>");
                out.close();

        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```



クラスのインポート

- サーブレットの動作には最低６つのクラスが必要
- JSPでは自動的にインポートされていたが、サーブレットでは明にimportが必要(が、Eclipseが自動的にやってくれます。)

```
3   import java.io.IOException;
4
5
6   import javax.servlet.ServletException;
7   import javax.servlet.annotation.WebServlet;
8   import javax.servlet.http.HttpServlet;
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
```

演習11-01

実

- サーブレットでGETメソッドに対し、文字列を表示するだけのサーブレットを作成すること
- サーブレット実行し、出力画面をキャプチャする。
- キャプチャしたファイルをSOLAに提出すること

【手順】

① 動的Webプロジェクトを新規に作成
② プロジェクト内でServletを新規に作成
③ Servletのコードを変更
④ Servletの実行

```
25
26  /**
27   * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
28   */
29  protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletExcepti
30      // TODO Auto-generated method stub
31      //response.getWriter().append("Served at: ").append(request.getContextPath());
32
33      response.setContentType("text/html; charset=utf-8");
34      PrintWriter out = response.getWriter();
35
36      out.println("<html>");
37      out.println("<head><title>Hello,World!</title></head>");
38      out.println("<body><h1>みなさん、こんにちは！</h1><body>");
39      out.println("</html>");
40  }
41
42  /**
43   * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
44   */
45  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletExcept
46      // TODO Auto-generated method stub
47      doGet(request, response);
```

アプリケーションクラスとメソッドのハンドラ

- HttpServletクラスを継承
- doGet(), doPost()をオーバーライド
- HttpServletRequestオブジェクト：クライアントから送信された入力を管理
- HttpServletResponseオブジェクト：クライアントへの出力を管理
- @WebServletアノテーション：サーブレットを呼び出すURLを定義。クラス宣言の直前に記述。ディレクトリ構成と一致不要
- Response.setContentType(“text/html; charset=“utf-8”);　← @page

```
15  @WebServlet("/HelloWorld")
16  public class HelloWorld extends HttpServlet {
17          private static final long serialVersionUID = 1L;
18
19      /**
20       * @see HttpServlet#HttpServlet()
21       */
22      public HelloWorld() {
23          super();
24          // TODO Auto-generated constructor stub
25      }
26
27          /**
28           * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
29           */
30          protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
31                  // TODO Auto-generated method stub
32                  //response.getWriter().append("Served at: ").append(request.getContextPath());
33
34                 response.setContentType("text/html; charset=utf-8");
35                  PrintWriter out = response.getWriter();
36                  out.println("<html>");
```

HTMLの出力

- JSPの暗黙オブジェクトは存在しない　⇒　インスタンス化が必要
- PrintWriter　out　= reponse.getWriter();
- out.print(), out.println() でHTMLを出力

```
32          //response.getWriter().append("Served at: ").append(request.getContext
33
34          response.setContentType("text/html; charset=utf-8");
35           PrintWriter out = response.getWriter();
36           out.println("<html>");
37           out.println("<head>");
38           out.println("<title>HelloWorld</title>");
39           out.println("</head>");
40           out.println("<body>");
41           out.println("<h1>HelloWorld</h1>");
42           out.println("</body>");
43           out.println("</html>");
44           out.close();
45
```

ServletのTomcatでの実行

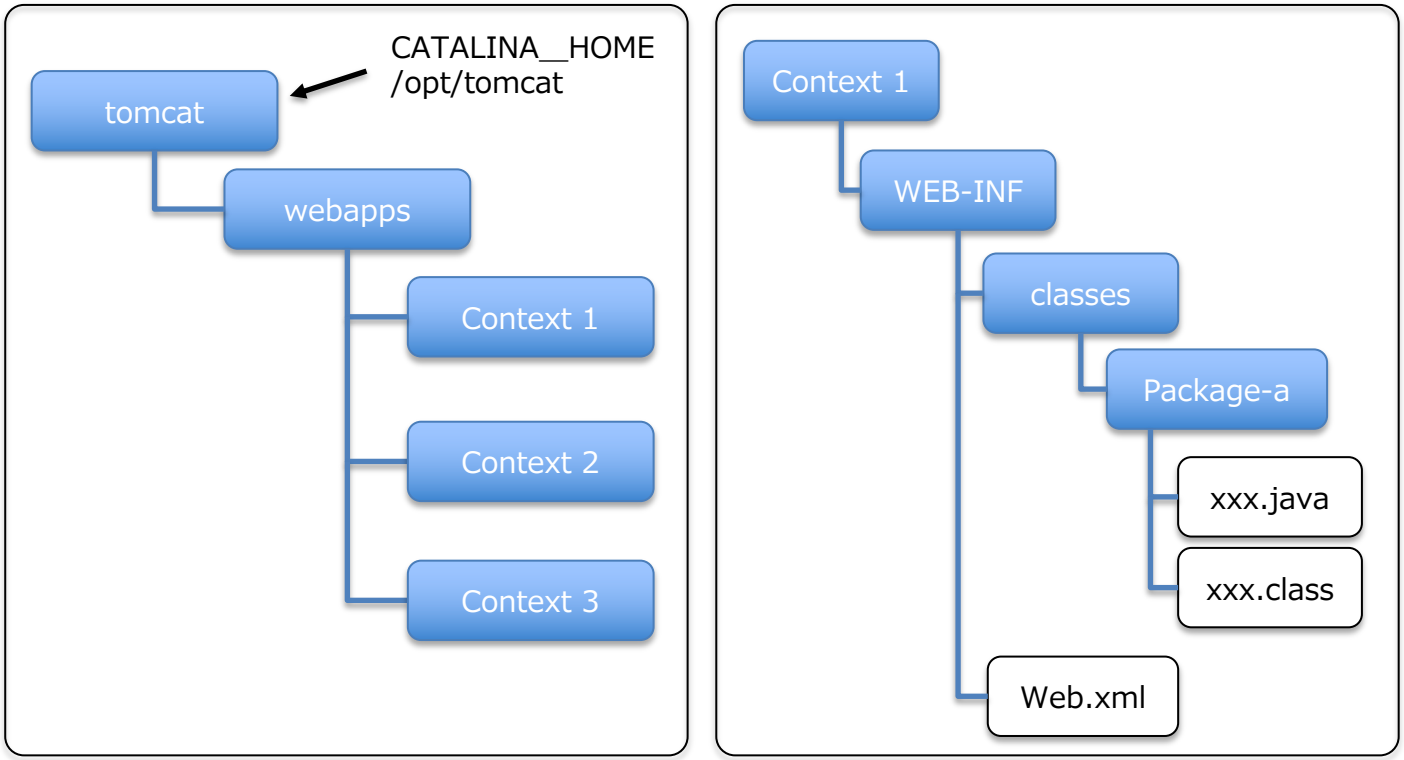
- Eclipse で作成した Servlet を Tomcat で実行するには・・・
- WARファイルの作成　⇒　展開　⇒　Tomcat の(再)起動
- WARファイル　：　Web Application aRchive　Webアプリケーションで必要なファイルをまとめたもの

Tomcatでの実行

1. WARファイルの作成
2. Tomcatのディレクトリにコピー
3. コンテナ(catalina) の起動　　sudo　/opt/tomcat/bin/catalina.sh　start

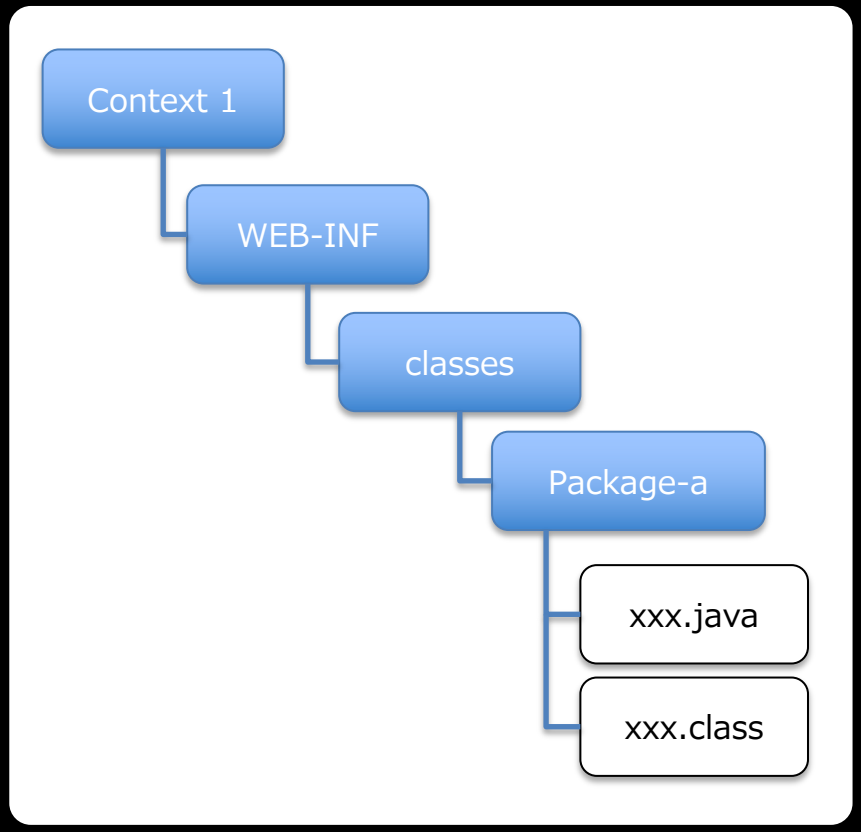
※ WARファイル　：Web Application aRchive 。 Javaベースの
Webアプリケーションやコンポーネントを圧縮し納めたファイル

【注意】EclipseのサーバとTomcatのサーバは両方起動するとコンフリクトを起こす。



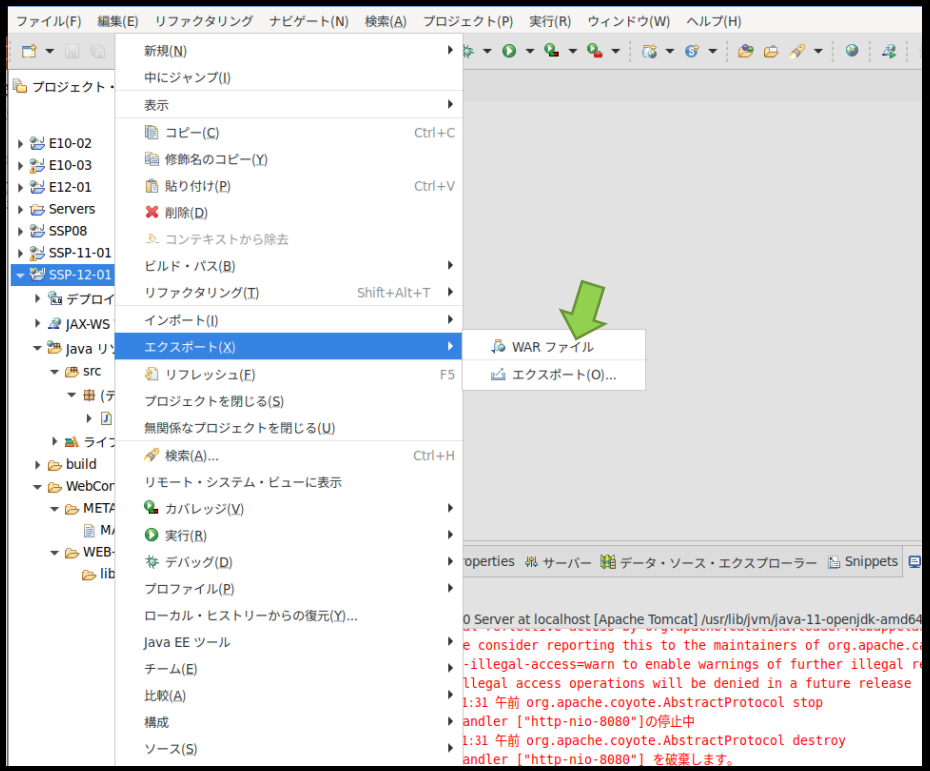
Tomcatでの実行：ファイル構成

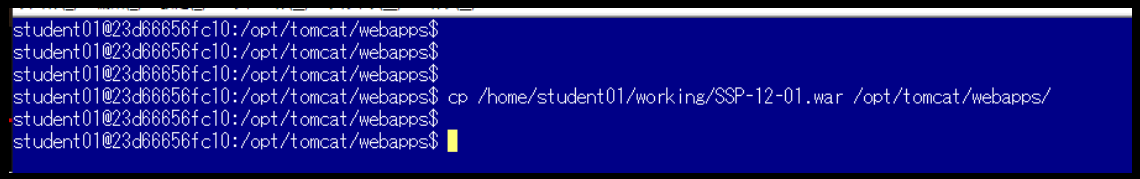
- コンテキスト(アプリケーション)は%CATALINA_HOME%(/opt/comcat)の配下に配置
- サーブレットファイル(XXX.java) はclassesの配下にパッケージに合わせて配置



WARファイルの作成と展開

- Eclipseで作成したServletのWARファイルを作成し、Tomcatで展開
- WARファイル　：　Web Application aRchive ファイル　Webアプリケーションに必要なファイルをまとめたもの
- EclipseでWARファイルを作成し、Tomcatのフォルダに配置する。





Tomcatでの実行：Tomcatの起動

- WARファイルを配置したら、Tomcatを起動

  sudo /opt/tomcat/bin/catalina.sh start

【注意】EclipseのサーバとTomcatのサーバは両方起動するとコンフリクトを起こす。

※ Tomcatの停止： sudo /opt/tomcat/bin/catalina.sh stop

```
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$ sudo /opt/tomcat/bin/catalina.sh start
Using CATALINA_BASE:   /opt/tomcat
Using CATALINA_HOME:   /opt/tomcat
Using CATALINA_TMPDIR: /opt/tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /opt/tomcat/bin/bootstrap.jar:/opt/tomcat/bin/tomcat-juli.jar
Tomcat started.
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$
```

Tomcatでの実行：ブラウザからアクセス

- http://localhost:8080/プロジェクト名/サーブレット名　にアクセス
- ポート8080はコンテナの外に出していないのでホストOSからはアクセス不可

演習11-02

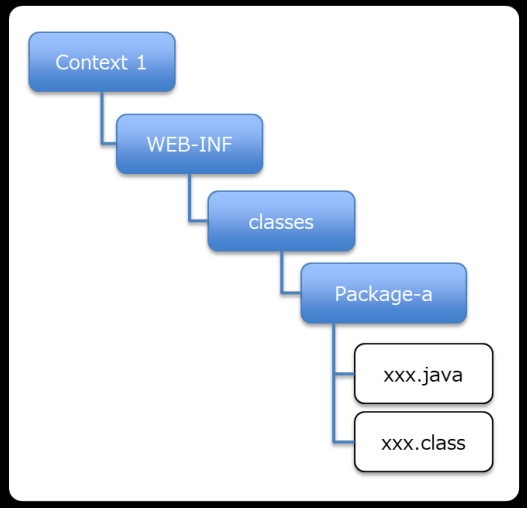
実

- 演習11-01で作成したサーブレットのWARファイルを作成し、Tomcatに展開する。
- Tomcat を起動し、ブラウザからサーブレットにアクセスする。
- 表示画面をキャプチャし、キャプチャファイルをSOLAに提出すること

Servlet の修正とコンパイル

- サーブレットのJavaファイルを修正したら、手動でコンパイルが必要
- コンパイルにはクラスパスを指定

  javac -encoding utf-8 -classpath /opt/apache-tomcat-9.0.36/lib/servlet-api.jar Javaファイル
- classファイルを更新したら、Tomcatを再起動




```
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$ javac -encoding UTF-8 -classpath /opt/apache-tom
cat-9.0.36/lib/servlet-api.jar counter.java
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$ sudo /opt/tomcat/bin/catalina.sh start
Using CATALINA_BASE:   /opt/tomcat
Using CATALINA_HOME:   /opt/tomcat
Using CATALINA_TMPDIR: /opt/tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /opt/tomcat/bin/bootstrap.jar:/opt/tomcat/bin/tomcat-juli.jar
Tomcat started.
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$
```

演習11-03

実

- 演習11-02でWARファイルを展開して作成されたサーブレットのJavaファイルを、Webでの表示が下記の図のようになるよう修正せよ。
- Tomcat を起動し、ブラウザからサーブレットにアクセスする。
- 表示画面をキャプチャし、キャプチャファイルをSOLAに提出すること

※ 展開後のファイルにはstudent01のアクセス権がありません。下記のコマンドを実行し編集できるようにして下さい。

sudo　chown　-R　student01　/opt/tomcat/webapps/プロジェクト名

演習11-04　サーブレットによるテーブル表示

実

- 下記の表を表示するサーブレットを作成してみましょう。
- 表示画面をキャプチャする。
- 作成したjavaファイルとキャプチャファイルをSOLAに提出する。

http://localhost:8080/servletTest/disptable

番号	日本語	英語
1	1月	January
2	2月	February
3	3月	March
4	4月	April
5	5月	May
6	6月	June
7	7月	July
8	8月	August
9	9月	September
10	10月	October
11	11月	November
12	12月	December

ServletのJSPとの連携

- JSP の form タグから Servlet を呼び出す。
- Servlet からフォワードで JSP に制御を移す。

【JSP連携①】JSPからのアクセスとPOSTの処理

- HTMLタグ(フォーム、リンク)で呼び出す。
- パラメータの取得： request.getParameter(“str”)

inputString.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Input string</title>
</head>
<body>

        <form method="POST" action="dispString">
                <label>文字列を入力
                        <input type="text" name="str" />
                </label>
                <input type="submit" value="送信" />
        </form>

</body>
</html>
```

dispString.java

```java
package servletTest;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class dispString
 */
@WebServlet("/dispString")
public class dispString extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public dispString() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                response.getWriter().append("Served at: ").append(request.getContextPath());
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                //doGet(request, response);

                request.setCharacterEncoding("UTF-8");
                response.setContentType( "text/html; charset=UTF-8" );
                PrintWriter out = response.getWriter();

                String msg = request.getParameter("str");

                out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
                out.println( "<title>Display input string</title>" );
                out.println( "</head>" );
                out.println( "<body>" );
                out.println( "<p>input string : " + msg + "</p>" );
                out.println( "</body>" );
                out.println( "<html>" );

                out.close();
        }

}
```

【JSP連携②】サーブレットからJSPへのフォワード

- サーブレットからJSPへ制御を移す。
- getServletContext().getRequestDispathcer(転送パス).forward(HttpServletRequest オブジェクト、 HttpServletResponseオブジェクト)

HelloWorld.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
        <p>Hello, world!</p>
</body>
</html>
```

http://localhost:8080/servletTest/forwardTest

Hello, world!

forwardTest.java

```
package servletTest;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class forwardTest
 */
@WebServlet("/forwardTest")
public class forwardTest extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public forwardTest() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletExce
                // TODO Auto-generated method stub
                //response.getWriter().append("Served at: ").append(request.getContextPath());
                getServletContext().getRequestDispatcher("/HelloWorld.jsp").forward(request,response);
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletExc
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```

Servlet　インスタンス変数とライブラリ

- インスタンス変数　：　インスタンスが消滅するまで値を保持　⇒　リクエストをまたいだ情報の受け渡し
- Javaライブラリ　：　Math等が使用可能

インスタンス変数

実

- インスタンスが消滅するまで保持される。

counter.java

```java
package servletTest;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class counter
 */
@WebServlet("/counter")
public class counter extends HttpServlet {
        private static final long serialVersionUID = 1L;

        private int counter=0;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public counter() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOExcept
ion {
                // TODO Auto-generated method stub
                //response.getWriter().append("Served at: ").append(request.getContextPath());

                counter++;

                response.setContentType( "text/html; charset=UTF-8" );
                PrintWriter out = response.getWriter();

                out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
                out.println( "<title>Counter</title>" );
                out.println( "</head>" );
                out.println( "<body>" );
                out.println( "アクセス数は" + counter + "です。</p>" );
                out.println( "</body>" );
                out.println( "</html>" );

                out.close();
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOExcep
tion {
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```

乱数

実

- ユーティリティクラスMathのメソッドが使えます。

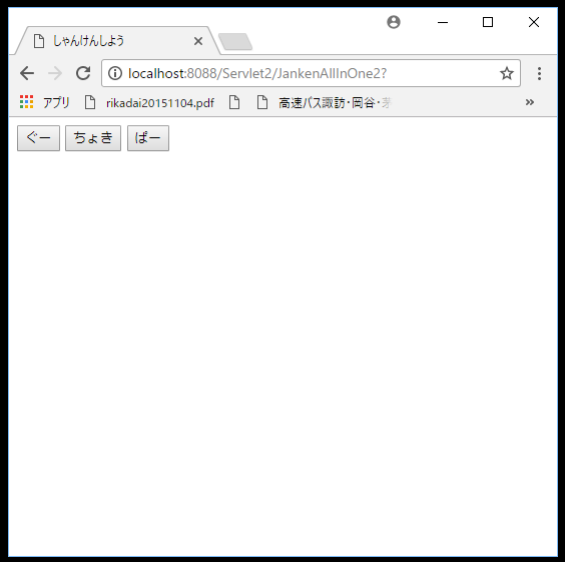
randTest.java

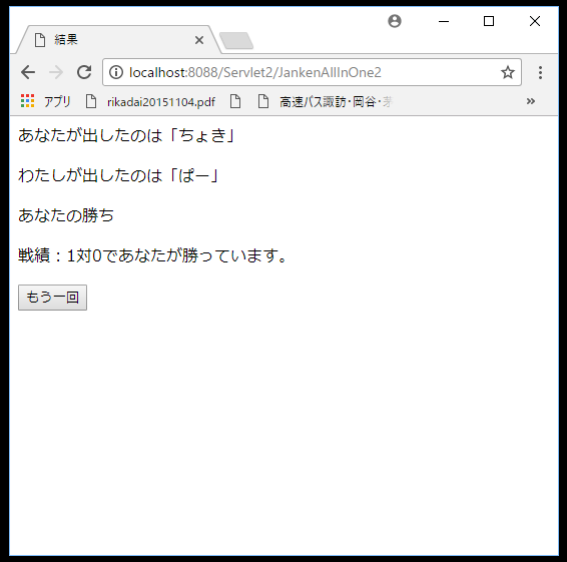
```
27      /**
28       * @see HttpServlet#doGet(HttpServletRequest request, HttpServletRespons
29       */
30      protected void doGet(HttpServletRequest request, HttpServletResponse res
31          // TODO Auto-generated method stub
32          //response.getWriter().append("Served at: ").append(request.getCont
33
34          int r = (int)(Math.random()*3);
35
36          response.setContentType( "text/html; charset=UTF-8" );
37          PrintWriter out = response.getWriter();
38
39          out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
40          out.println( "<title>Random number</title>" );
41          out.println( "</head>" );
42          out.println( "<body>" );
43          out.println( "今回の数は" + r + "です。</p>" );
44          out.println( "</body>" );
45          out.println( "</html>" );
46
47          out.close();
48      }
```

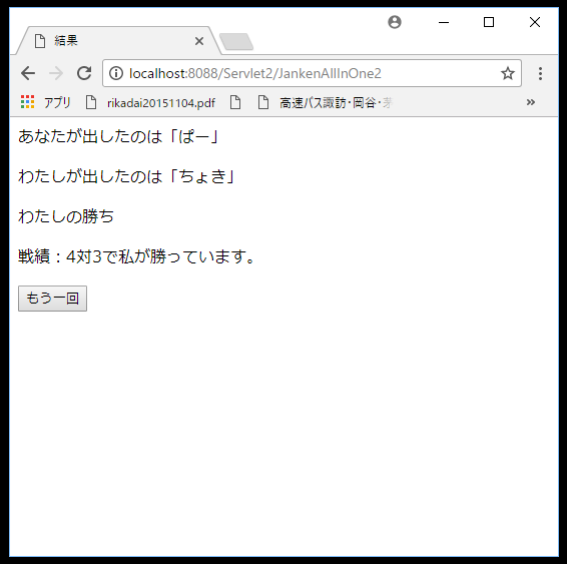
演習11-05　JSPからのサーブレットの呼び出し

実

- 下記の動作をするサーブレットを作成しましょう。
- じゃんけんの相手はランダムで出す手を決める。
- 作成したサーブレットのJavaファイルをSOLAに提出する。







JavaBeans

JavaBeans

- 汎用的に部品化できるよう幾つかの条件を満たしたクラス　⇒部品化のためのクラス
- 条件1：引数のないコンストラクタを持つ。
- 条件2：フィールドを参照・設定できるアクセッサを持つ。
- 条件3：Serializableインタフェースを実装する。　⇒　implements Serializable
- シリアライズ・シリアル化・直列化　⇒　情報をバイト列に変換
- サーブレットからのJavaBeansの仕様：他のクラスと同様、インスタンス化

JavaBeansの例

- お店データベースの情報を管理

StoreInfo.java

```java
package nuno.javaAdv.test.first;

import java.io.Serializable;

import java.util.ArrayList;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.sql.DataSource;

public class StoreInfo implements Serializable {

        public StoreInfo() {};

        private int id;
        public void setId(int id ) {this.id=id; }
        public int getId() { return id; }

        private String name;
        public void setName( String name ) { this.name = name; }
        public String getName() { return name; }

        private String add;
        public void setAdd( String add ) { this.add = add; }
        public String getAdd() { return add; }

        public int insertStoreInfo() {

                int result = 0;
                Connection db = null;
                PreparedStatement ps =null;
                try {
                        Context context = new InitialContext();
                        DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
                        db = ds.getConnection();
                        ps = db.prepareStatement("INSERT INTO list_t ( name_c, add_c ) VALUES(? , ? )");
                        ps.setString(1, name );
                        ps.setString(2, add );
                        result = ps.executeUpdate();
                        ps.close();
                        db.close();
                } catch (NamingException e) {
                        e.printStackTrace();
                } catch (SQLException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                if(ps!=null) { ps.close(); }
                                if(db!=null) { db.close(); }
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                return result;
        }
```

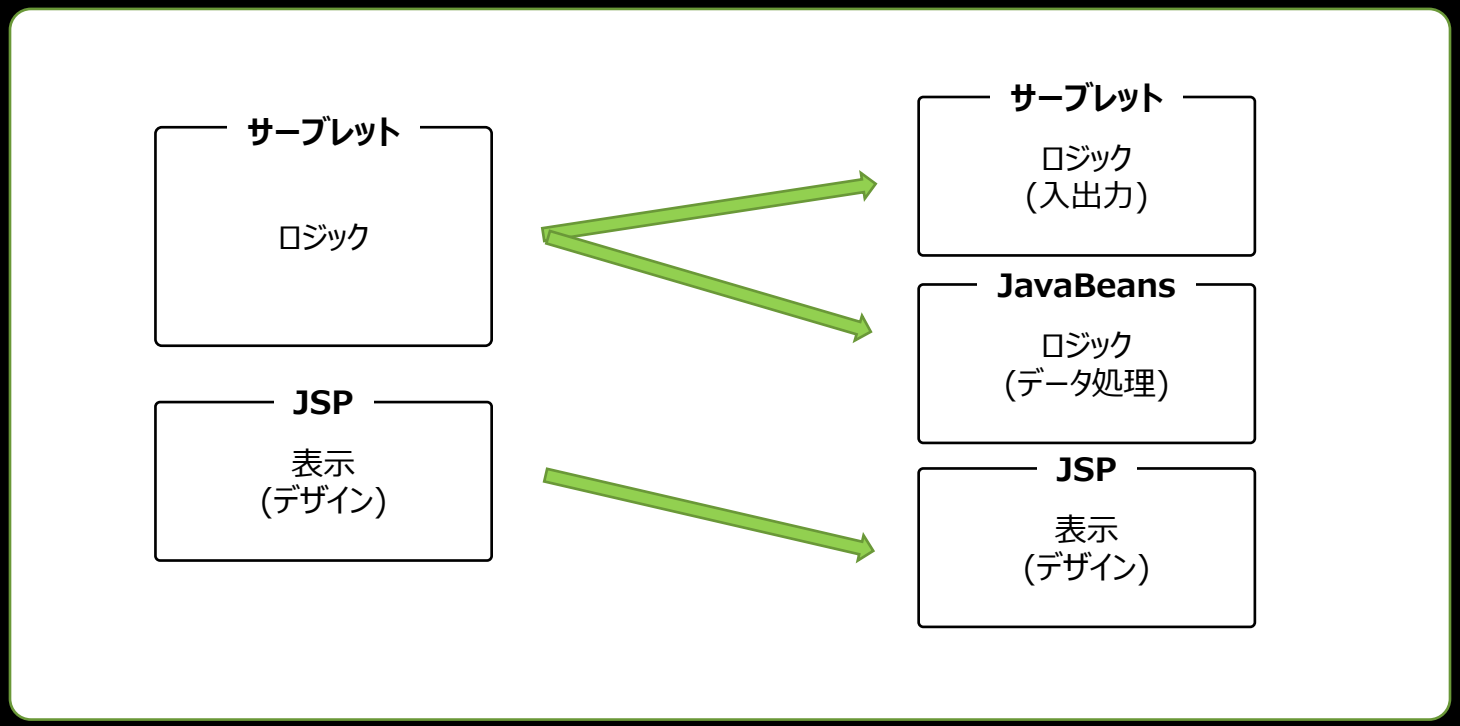
JavaBeansの例 (つづき)

- お店データベースの情報を管理

```
63
64      public int updateStoreInfo() {
65
66              int result = 0;
67              Connection db = null;
68              PreparedStatement ps =null;
69              try {
70                      Context context = new InitialContext();
71                      DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
72                      db = ds.getConnection();
73                      ps = db.prepareStatement("UPDATE list_t SET name_c=?, add_c=? WHERE id_c=?" );
74                      ps.setString(1, name );
75                      ps.setString(2, add );
76                      ps.setInt(3, id );
77                      result = ps.executeUpdate();
78                      ps.close();
79                      db.close();
80              } catch (NamingException e) {
81                      e.printStackTrace();
82              } catch (SQLException e) {
83                      e.printStackTrace();
84              } finally {
85                      try {
86                              if(ps!=null) { ps.close(); }
87                              if(db!=null) { db.close(); }
88                      } catch (SQLException e) {
89                              e.printStackTrace();
90                      }
91              }
92              return result;
93      }
94  }
95
```

役割分担と連携

- ロジック関連処理を①入出力(入力データ受け取り、処理結果受け渡し)と②データ処理に分離
- ①⇒サーブレット、　②⇒JavaBeans
- JSP：ユーザインタフェース
- サーブレット、JavaBeans、JSPが連携して一つのアプリケーションを形成



まとめ

- Javaサーブレットはサーバ上で動作するJavaプログラム。動的Webアプリケーションを構築するためのサーバサイド技術
- Javaサーブレットはサーブレットコンテナ上で実行される
- Javaサーブレットのライフタイムは、最初にリクエストを受けてからコンテナが終了するまで
- Javaサーブレットは「シングルインスタンス・マルチスレッド」で実施される。
- JSPがHTMLに埋め込むJavaコードであるのに対し、JavaサーブレットはJavaでHTMLを出力
- Javaサーブレットはロジックの記述が得意。HTMLを大量に出力するのは苦手
- JavaとJSPで役割分担を行い、メンテナンスしやすいアプリケーションを実現
- JavaサーブレットのアプリケーションクラスはHttpServletを継承
- Httpメソッドのハンドラ：doGet()、doPost()
- Javaサーブレットでは、JSPのような暗黙のオブジェクトは存在しない。⇒インスタンス化が必要
- Tomcatでのファイル構成：Javaサーブレットのファイルは、アプリケーション＞WEB-INF＞classesの配下にパッケージに従って配置
- Javaサーブレットでのパラメータの受け取りはrequest.getParameter("XXX") を使用
- JavaサーブレットからJSPへのフォワード：getServletContext().getRequestDispathcer(転送パス).forward(HttpServletRequestオブジェクト、 HttpServletResponseオブジェクト)
- インスタンス変数は保持される。
- ライブラリが使えます。
- JavaBeans：汎用的に部品化できるよう幾つかの条件を満たしたクラス　⇒部品化のためのクラス
- JSP、Javaサーブレット、JavaBeansで役割分担　⇒メンテナンスしやすいアプリケーション構成

おわり

サーバシステムプログラミング及び演習

– 第12回 Servlet (2) –

今後のスケジュール

第13回	7月03日	Servlet (3)
第14回	7月10日	まとめと課題

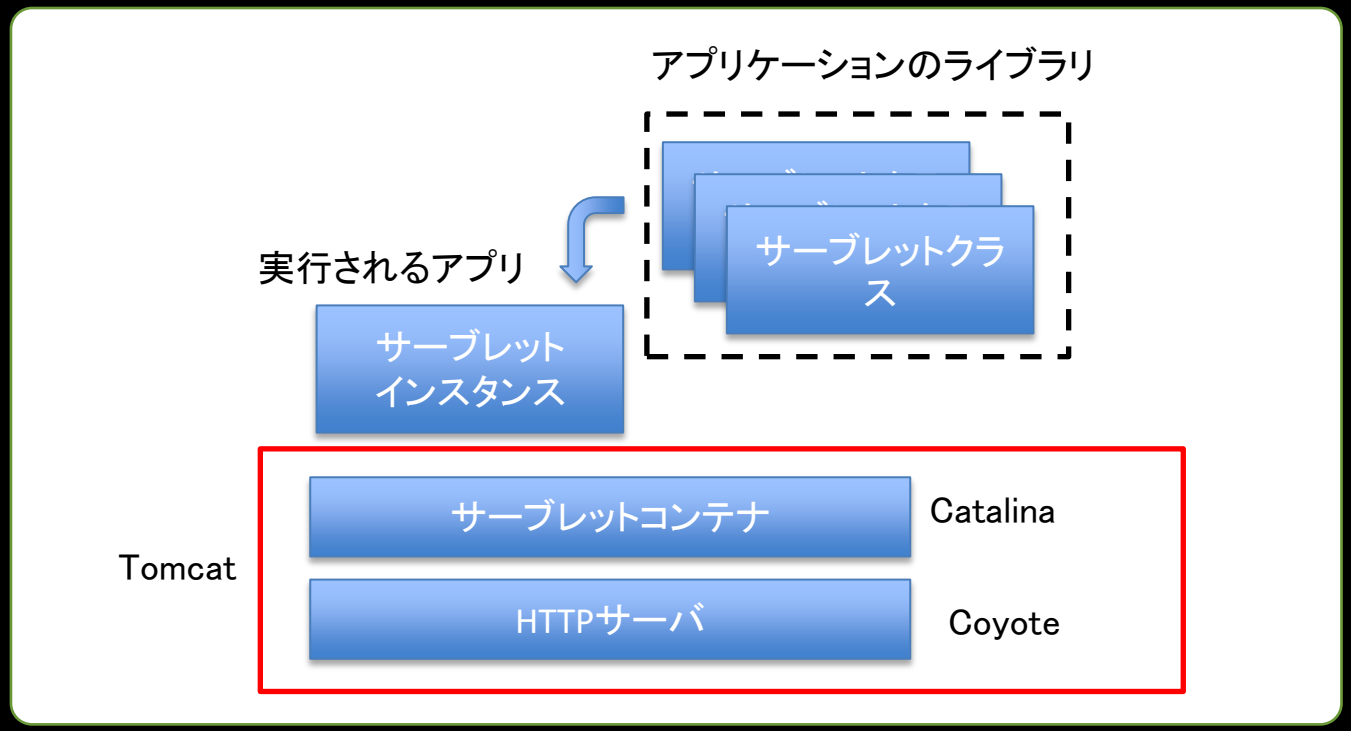
※ 定期試験はありません。

前回の内容

- Javaサーブレットはサーバ上で動作するJavaプログラム。動的Webアプリケーションを構築するためのサーバサイド技術
- Javaサーブレットはサーブレットコンテナ上で実行される
- Javaサーブレットのライフタイムは、最初にリクエストを受けてからコンテナが終了するまで
- Javaサーブレットは「シングルインスタンス・マルチスレッド」で実施される。
- JSPがHTMLに埋め込むJavaコードであるのに対し、JavaサーブレットはJavaでHTMLを出力
- Javaサーブレットはロジックの記述が得意。HTMLを大量に出力するのは苦手
- JavaとJSPで役割分担を行い、メンテナンスしやすいアプリケーションを実現
- JavaサーブレットのアプリケーションクラスはHttpServletを継承
- Httpメソッドのハンドラ：doGet()、doPost()
- Javaサーブレットでは、JSPのような暗黙のオブジェクトは存在しない。⇒インスタンス化が必要
- Tomcatでのファイル構成：Javaサーブレットのファイルは、アプリケーション＞WEB-INF＞classesの配下にパッケージに従って配置
- Javaサーブレットでのパラメータの受け取りはrequest.getParameter("XXX") を使用
- JavaサーブレットからJSPへのフォワード：getServletContext().getRequestDispathcer(転送パス).forward(HttpServletRequestオブジェクト、 HttpServletResponseオブジェクト)
- インスタンス変数は保持される。
- ライブラリが使えます。

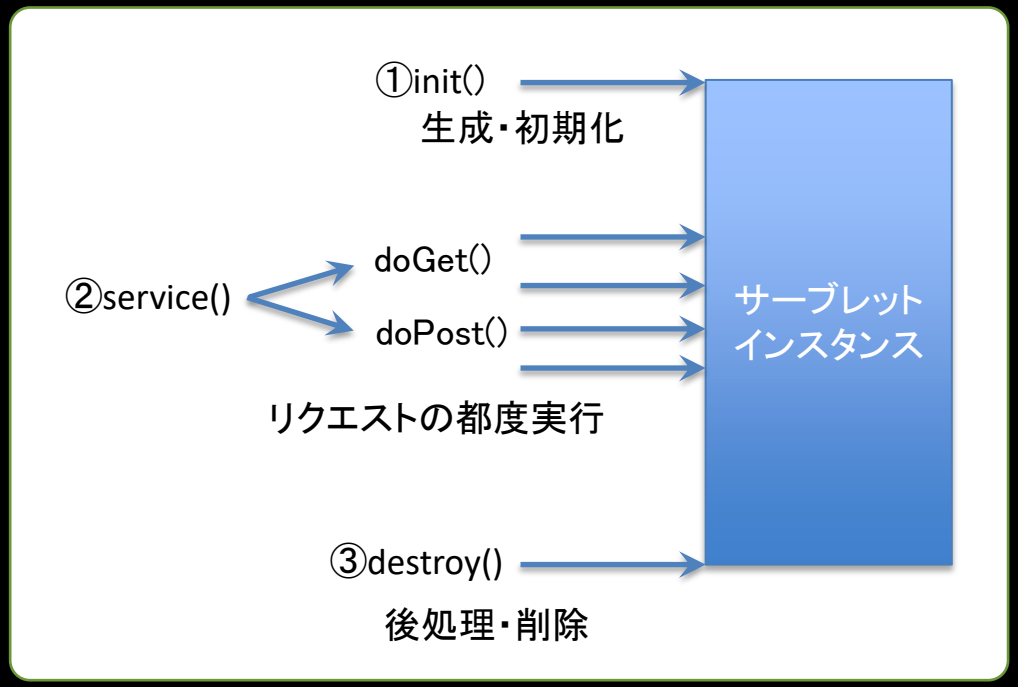
2. サーブレットコンテナ　Apache Tomcat

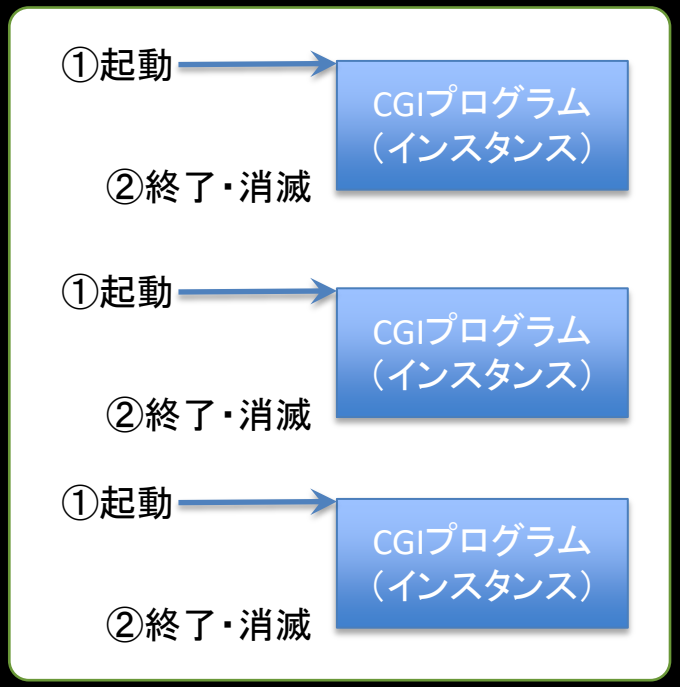
- サーブレットコンテナ：　Javaサーブレットの実行環境
- Apache Tomcat：Apache Software Foundationが開発しているオープンソースのサーブレットコンテナ
- Apache Tomcatはコンテナと HTTPサーバから構成　(コンテナ：Catalina、HTTPサーバ：Coyote　)
- Apache HTTP Serverとも連携可



サーブレットのライフタイム

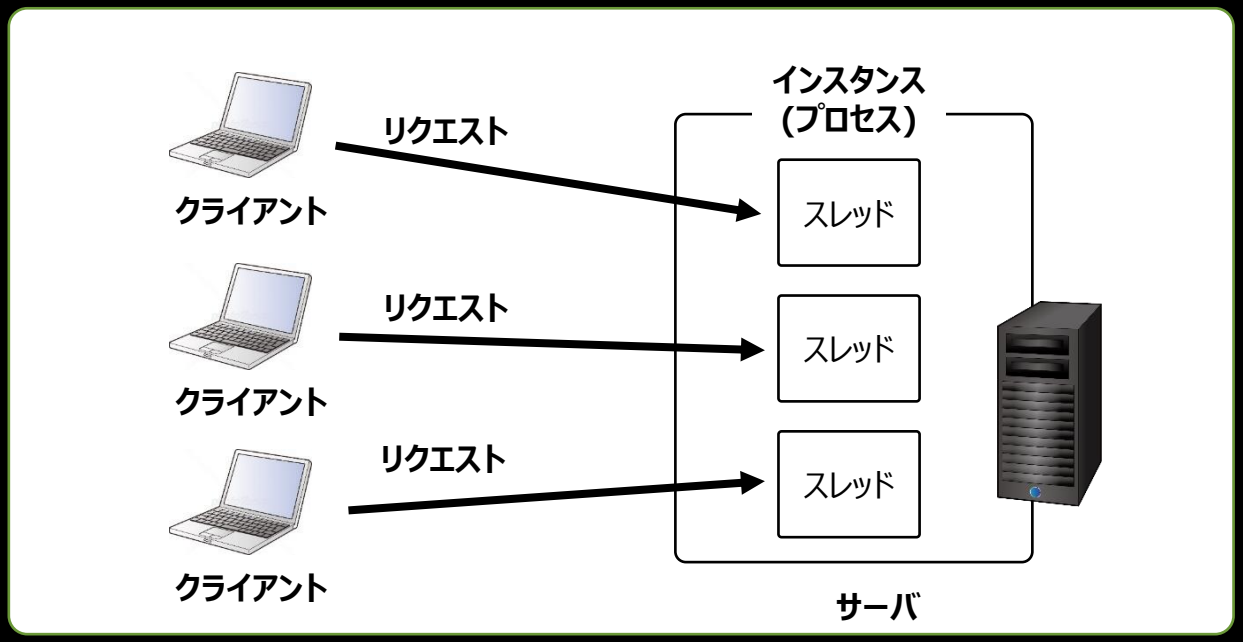
- サーブレットのライフタイム：init()からdestroy()まで
- ライフタイム中は複数のリクエストに対しサービスを提供　(cf.　CGI：リクエストの都度インスタンスが生成⇒消滅)
- サービスは並列実行可能　(要注意)





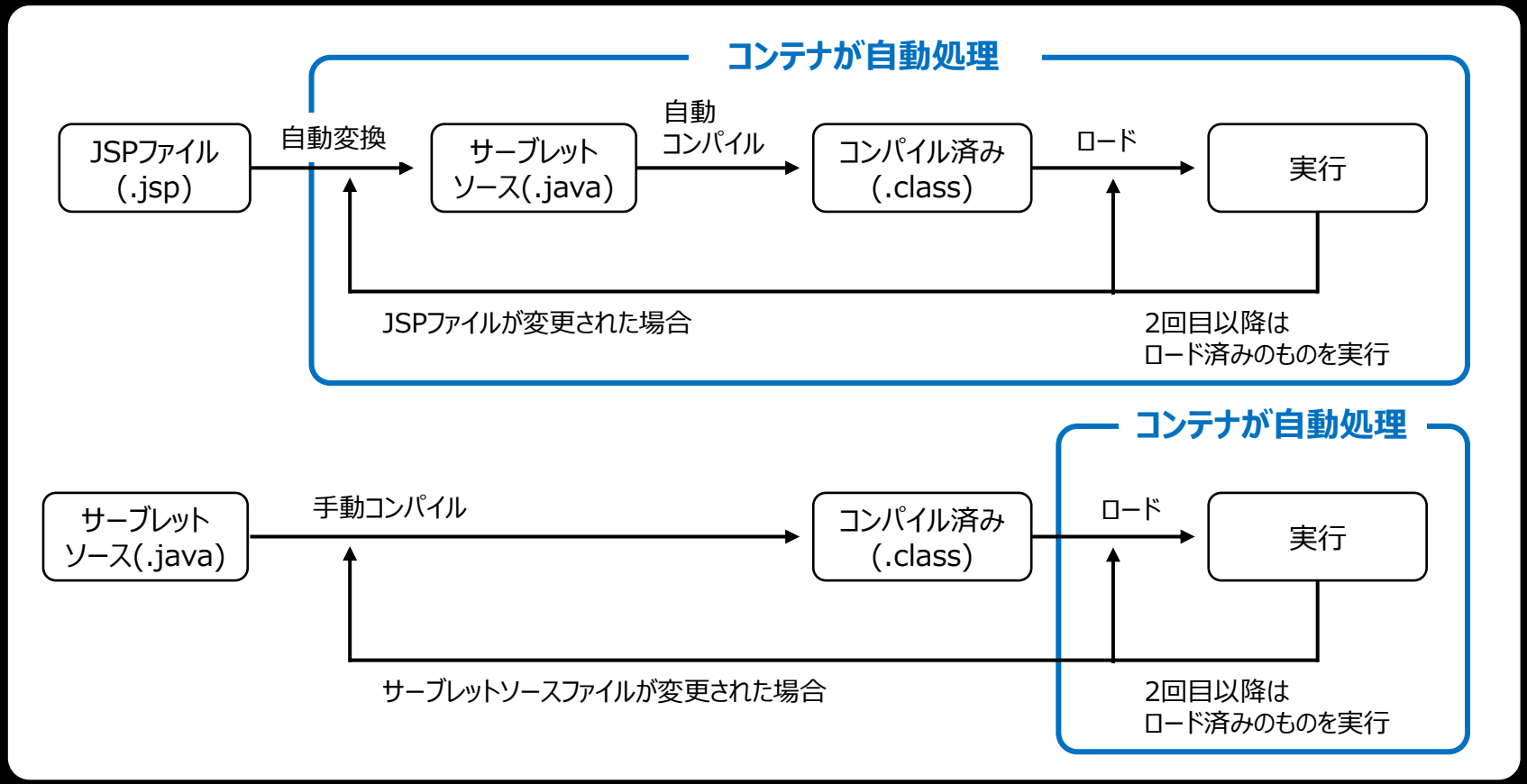
シングルインスタンス・マルチスレッド

- サーブレットは一度インスタンス化されると常駐する　(≠リクエストの度にインスタンス化　)
- 同じサーブレットに対するリクエストは同じインスタンスで処理される。
- 複数の処理がマルチスレッドで実行される。(※スレッド：プロセス内に作られる並列動作が可能な処理の単位)



3．サーブレットとJSP

- Java Servlet
- 動的Webアプリケーションを構築するためのサーバサイド技術
- JSPがHTMLに埋め込むJavaコードであるのに対し、サーブレットはJavaでHTMLを出力



Servlet

Java servlet

1. サーブレットコンテナ上で動作するJavaのプログラムです。
2. コンテナで動作させるには手動でコンパイルする必要があります。
   (が、Eclipseを使用している場合、自動でやってくれます。)
3. サーブレットはロジックの記述が得意
4. サーブレットはHTMLを大量に出力するのが苦手

```
29          /**
30           * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
31           */
32          protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
            IOExcept
33          ion {
34                  // TODO Auto-generated method stub
35                  //response.getWriter().append("Served at: ").append(request.getContextPath());
36
37                  counter++;
38
39                  response.setContentType( "text/html; charset=UTF-8" );
40                  PrintWriter out = response.getWriter();
41
42                  out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
43                  out.println( "<title>Counter</title>" );
44                  out.println( "</head>" );
45                  out.println( "<body>" );
46                  out.println( "アクセス数は" + counter + "です。</p>" );
47                  out.println( "</body>" );
48                  out.println( "</html>" );
49
50                  out.close();
51          }
52
```

サーブレットの必要性

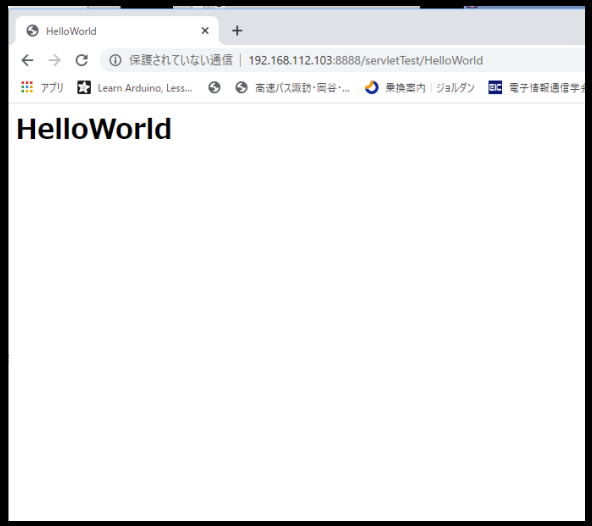
- アプリケーションの構築　⇒　「ただ動けばよい」というものではない。
- メンテナンスのしやすさ、可読性が重要
- JSPで、込み入ったロジックを技術した場合、非常に読みづらい、可読性の低いコードとなる。
- サーブレットとJSPはお互いに補完し合う関係にある
- 役割分担により整理された構成を実現
- 役割分担：サーブレット　⇒　ロジック、　JSP　⇒　HTML

サーブレットの例

- “HelloWorld” と表示するだけのサーブレット
- GETメソッドのハンドラ：doGet()、　POSTメソッドのハンドラ：doPost()

HelloWorld.java

```java
package servletTest;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class HelloWorld
 */
@WebServlet("/HelloWorld")
public class HelloWorld extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public HelloWorld() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                //response.getWriter().append("Served at: ").append(request.getContextPath());

               response.setContentType("text/html; charset=utf-8");
                PrintWriter out = response.getWriter();
                out.println("<html>");
                out.println("<head>");
                out.println("<title>HelloWorld</title>");
                out.println("</head>");
                out.println("<body>");
                out.println("<h1>HelloWorld</h1>");
                out.println("</body>");
                out.println("</html>");
                out.close();

        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```



クラスのインポート

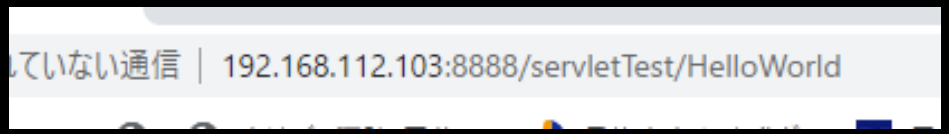
- サーブレットの動作には最低６つのクラスが必要
- JSPでは自動的にインポートされていたが、サーブレットでは明にimportが必要(が、Eclipseが自動的にやってくれます。)

```
3   import java.io.IOException;
4
5
6   import javax.servlet.ServletException;
7   import javax.servlet.annotation.WebServlet;
8   import javax.servlet.http.HttpServlet;
9   import javax.servlet.http.HttpServletRequest;
10  import javax.servlet.http.HttpServletResponse;
```

アプリケーションクラスとメソッドのハンドラ

- HttpServletクラスを継承
- doGet(), doPost()をオーバーライド
- HttpServletRequestオブジェクト：クライアントから送信された入力を管理
- HttpServletResponseオブジェクト：クライアントへの出力を管理
- @WebServletアノテーション：サーブレットを呼び出すURLを定義。クラス宣言の直前に記述。ディレクトリ構成と一致不要
- Response.setContentType(“text/html; charset=“utf-8”);　← @page

```
15  @WebServlet("/HelloWorld")
16  public class HelloWorld extends HttpServlet {
17          private static final long serialVersionUID = 1L;
18
19      /**
20       * @see HttpServlet#HttpServlet()
21       */
22      public HelloWorld() {
23          super();
24          // TODO Auto-generated constructor stub
25      }
26
27          /**
28           * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
29           */
30          protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
31                  // TODO Auto-generated method stub
32                  //response.getWriter().append("Served at: ").append(request.getContextPath());
33
34                 response.setContentType("text/html; charset=utf-8");
35                  PrintWriter out = response.getWriter();
36                  out.println("<html>");
```



HTMLの出力

- JSPの暗黙オブジェクトは存在しない　⇒　インスタンス化が必要
- PrintWriter　out　= reponse.getWriter();
- out.print(), out.println() でHTMLを出力

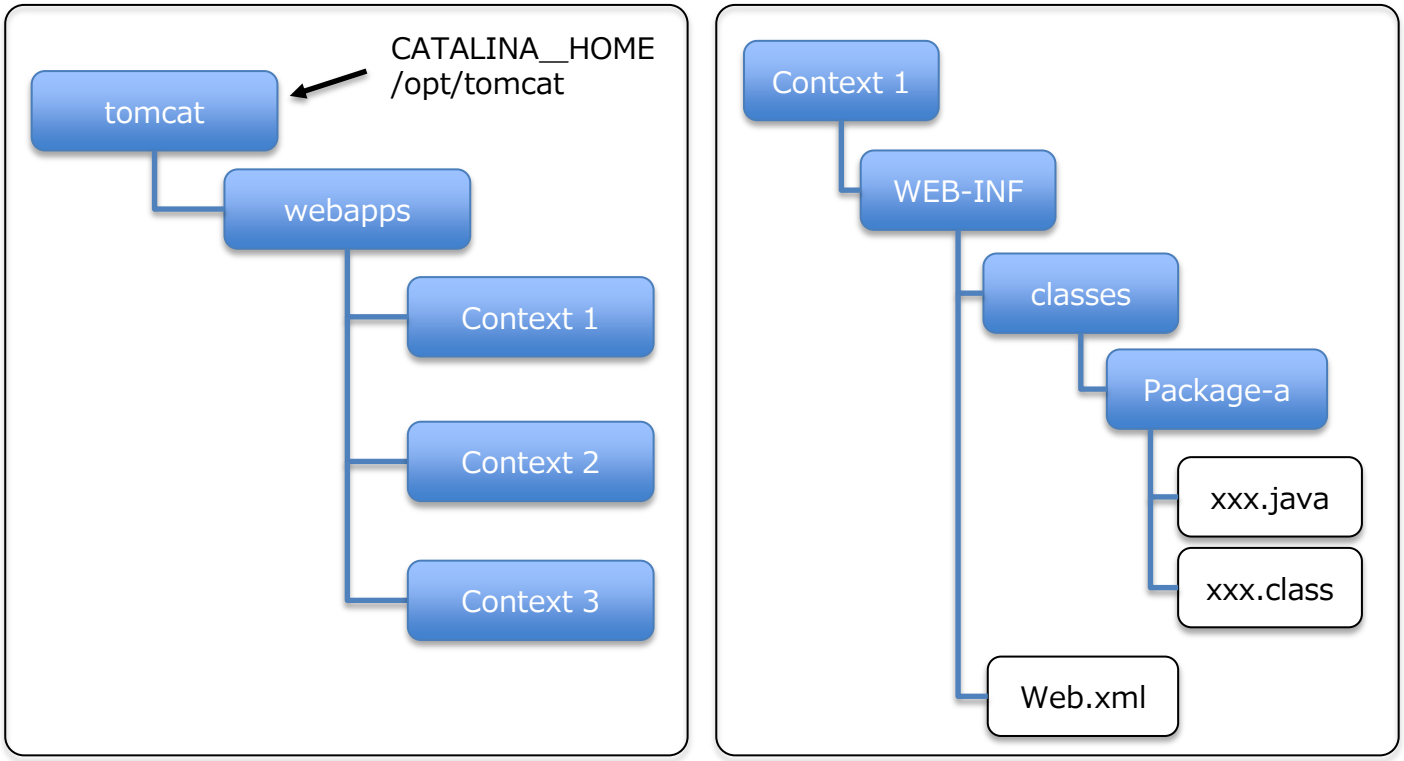
```
32          //response.getWriter().append("Served at: ").append(request.getContext
33
34          response.setContentType("text/html; charset=utf-8");
35           PrintWriter out = response.getWriter();
36           out.println("<html>");
37           out.println("<head>");
38           out.println("<title>HelloWorld</title>");
39           out.println("</head>");
40           out.println("<body>");
41           out.println("<h1>HelloWorld</h1>");
42           out.println("</body>");
43           out.println("</html>");
44           out.close();
45
```

Tomcatでの実行

1. WARファイルの作成
2. Tomcatのディレクトリにコピー
3. コンテナ(catalina) の起動　　sudo　/opt/tomcat/bin/catalina.sh　start

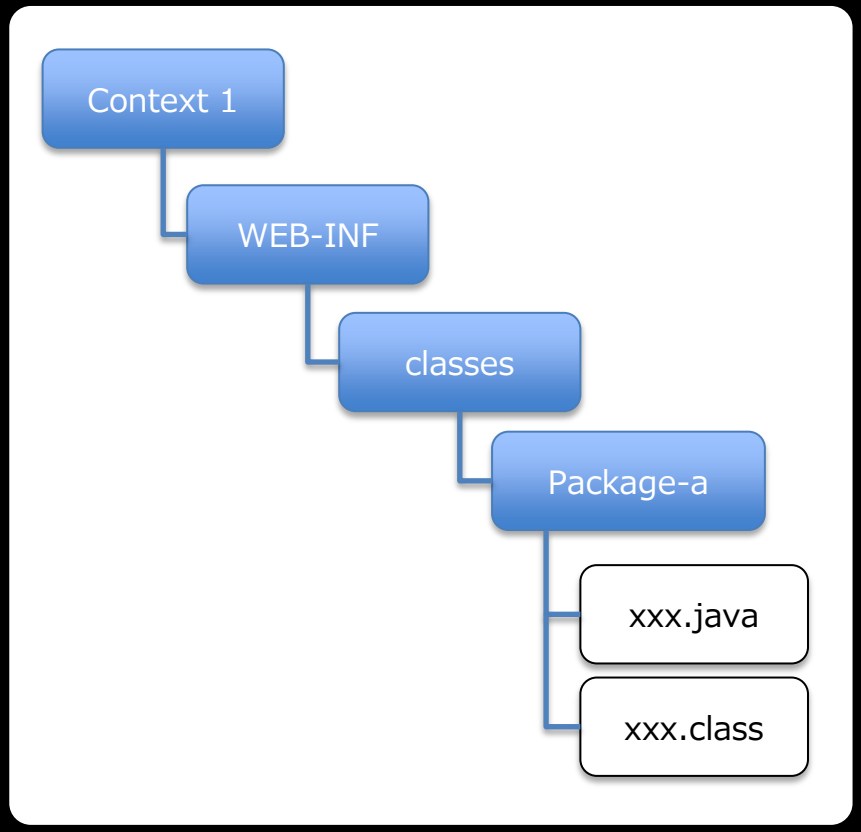
※ WARファイル　：Web Application aRchive 。 Javaベースの
Webアプリケーションやコンポーネントを圧縮し納めたファイル

【注意】EclipseのサーバとTomcatのサーバは両方起動するとコンフリクトを起こす。



Tomcatでの実行：ファイル構成

- コンテキスト(アプリケーション)は%CATALINA_HOME%(/opt/comcat)の配下に配置
- サーブレットファイル(XXX.java) はclassesの配下にパッケージに合わせて配置



Tomcatでの実行：コンパイルと再起動

実

- サーブレットのJavaファイルを作成したら、手動でコンパイル
- classファイルを更新したら、Tomcatを再起動

```
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$ javac -encoding UTF-8 -classpath /opt/apache-tom
cat-9.0.36/lib/servlet-api.jar counter.java
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$ sudo /opt/tomcat/bin/catalina.sh start
Using CATALINA_BASE:   /opt/tomcat
Using CATALINA_HOME:   /opt/tomcat
Using CATALINA_TMPDIR: /opt/tomcat/temp
Using JRE_HOME:        /usr
Using CLASSPATH:       /opt/tomcat/bin/bootstrap.jar:/opt/tomcat/bin/tomcat-juli.jar
Tomcat started.
student01@e7ffae141a9e:/opt/tomcat/webapps/servletTest2/WEB-INF/classes/javaAdv$
```

演習12-01　サーブレットによるテーブル表示

実

- 下記の表を表示するサーブレットを作成してみましょう。
- 表示画面をキャプチャし、キャプチャファイルをSOLAに提出する。

http://localhost:8080/servletTest/disptable

番号	日本語	英語
1	1月	January
2	2月	February
3	3月	March
4	4月	April
5	5月	May
6	6月	June
7	7月	July
8	8月	August
9	9月	September
10	10月	October
11	11月	November
12	12月	December

ServletのJSPとの連携

- JSP の form タグから Servlet を呼び出す。
- Servlet からフォワードで JSP に制御を移す。

【JSP連携】JSPからのアクセスとPOSTの処理

- HTMLタグ(フォーム、リンク)で呼び出す。
- パラメータの取得：　request.getParameter(“str”)

inputString.jsp

```jsp
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Input string</title>
</head>
<body>

        <form method="POST" action="dispString">
                <label>文字列を入力
                        <input type="text" name="str" />
                </label>
                <input type="submit" value="送信" />
        </form>

</body>
</html>
```

dispString.java

```java
package servletTest;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class dispString
 */
@WebServlet("/dispString")
public class dispString extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public dispString() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                response.getWriter().append("Served at: ").append(request.getContextPath());
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
                // TODO Auto-generated method stub
                //doGet(request, response);

                request.setCharacterEncoding("UTF-8");
                response.setContentType( "text/html; charset=UTF-8" );
                PrintWriter out = response.getWriter();

                String msg = request.getParameter("str");

                out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
                out.println( "<title>Display input string</title>" );
                out.println( "</head>" );
                out.println( "<body>" );
                out.println( "<p>input string : " + msg + "</p>" );
                out.println( "</body>" );
                out.println( "<html>" );

                out.close();
        }

}
```

【JSP連携】サーブレットからJSPへのフォワード

- サーブレットからJSPへ制御を移す。
- getServletContext().getRequestDispathcer(転送パス).forward(HttpServletRequest オブジェクト、 HttpServletResponseオブジェクト)

HelloWorld.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
        <p>Hello, world!</p>
</body>
</html>
```

http://localhost:8080/servletTest/forwardTest

Hello, world!

forwardTest.java

```
package servletTest;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class forwardTest
 */
@WebServlet("/forwardTest")
public class forwardTest extends HttpServlet {
        private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public forwardTest() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletExce
                // TODO Auto-generated method stub
                //response.getWriter().append("Served at: ").append(request.getContextPath());
                getServletContext().getRequestDispatcher("/HelloWorld.jsp").forward(request,response);
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletExc
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```

インスタンス変数

実

- インスタンスが消滅するまで保持される。

counter.java

```java
package servletTest;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class counter
 */
@WebServlet("/counter")
public class counter extends HttpServlet {
        private static final long serialVersionUID = 1L;

        private int counter=0;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public counter() {
        super();
        // TODO Auto-generated constructor stub
    }

        /**
         * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOExcept
ion {
                // TODO Auto-generated method stub
                //response.getWriter().append("Served at: ").append(request.getContextPath());

                counter++;

                response.setContentType( "text/html; charset=UTF-8" );
                PrintWriter out = response.getWriter();

                out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
                out.println( "<title>Counter</title>" );
                out.println( "</head>" );
                out.println( "<body>" );
                out.println( "アクセス数は" + counter + "です。</p>" );
                out.println( "</body>" );
                out.println( "</html>" );

                out.close();
        }

        /**
         * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)
         */
        protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        IOExcep
tion {
                // TODO Auto-generated method stub
                doGet(request, response);
        }

}
```

乱数

実

- ユーティリティクラスMathのメソッドが使えます。

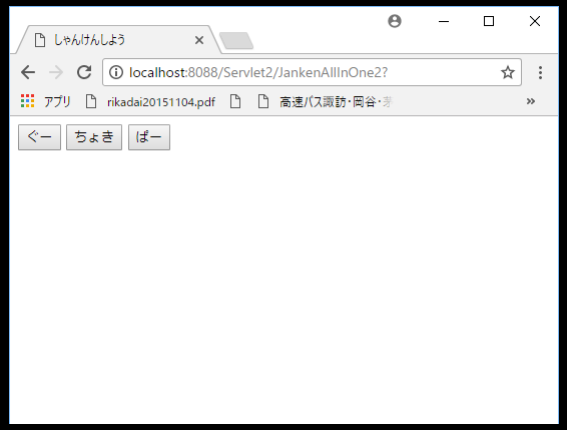
randTest.java

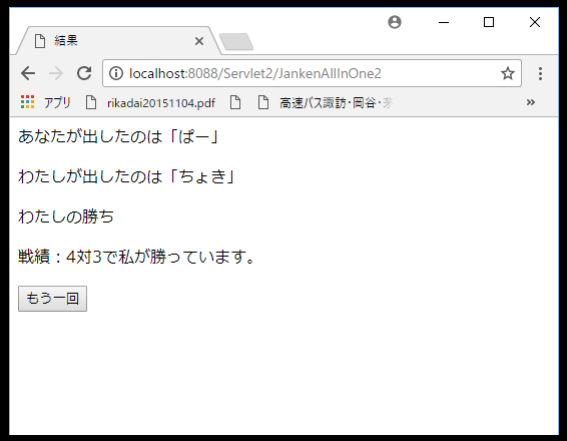
```
27      /**
28       * @see HttpServlet#doGet(HttpServletRequest request, HttpServletRespons
29       */
30      protected void doGet(HttpServletRequest request, HttpServletResponse res
31          // TODO Auto-generated method stub
32          //response.getWriter().append("Served at: ").append(request.getCont
33
34          int r = (int)(Math.random()*3);
35
36          response.setContentType( "text/html; charset=UTF-8" );
37          PrintWriter out = response.getWriter();
38
39          out.println( "<!DOCTYPE html><head><meta charset='UTF-8' />" );
40          out.println( "<title>Random number</title>" );
41          out.println( "</head>" );
42          out.println( "<body>" );
43          out.println( "今回の数は" + r + "です。</p>" );
44          out.println( "</body>" );
45          out.println( "</html>" );
46
47          out.close();
48      }
```

演習11-05　JSPからのサーブレットの呼び出し

実

- 下記の動作をするJSPとサーブレットを作成しましょう。
- じゃんけんの相手はランダムで出す手を決める。
- 戦績を表示する。
- 作成したJSPファイル、JavaファイルをSOLAに提出





演習11-06

実

- 演習11-05をサーブレットだけで実現しましょう。
- 作成したJavaファイルをSOLAに提出

  ※GETとPOSTごとに表示を変えることで一つのJavaファイルで実現

本日の内容

本日の内容

- JavaBeans：汎用的に部品化できるよう幾つかの条件を満たしたクラス　⇒部品化のためのクラス
- JavaBeansのスコープ：　page, request, session, application
- ServletからJSPへのJavaBeansの受け渡し：　setAttribute(), forward()
- JSPでのJavaBeansの受け取り：　アクションタグ <jsp:useBean>
- MVCモデル：　Model →JavaBeans、View→JSP、Controller→Javaサーブレット　で役割分担
- MVCモデルのメリット：独立性の高いモジュール構成。開発分担が容易。コード煩雑化を抑制。コード再利用可　⇒メンテナンスしやすいアプリケーション構成

JavaBeans

JavaBeans

- 汎用的に部品化できるよう幾つかの条件を満たしたクラス　⇒部品化のためのクラス
- 条件1：引数のないコンストラクタを持つ。
- 条件2：フィールドを参照・設定できるアクセッサを持つ。
- 条件3：Serializableインタフェースを実装する。　⇒　implements Serializable
- インターフェースSerializable：マーカーインターフェース　→中身なし
- シリアライズ・シリアル化・直列化　⇒　情報をバイト列に変換
- サーブレットからのJavaBeansの仕様：他のクラスと同様、インスタンス化

- JavaBeans を使って、JavaサーブレットとJSPとの間で情報のやり取りを実現

JavaBeans

- 条件1：引数のないコンストラクタを持つ。
- 条件2：フィールドを参照・設定できるアクセッサを持つ。
- 条件3：Serializableインタフェースを実装する。　⇒　implements Serializable

```
17
18  public class StoreInfo implements Serializable {
19
20          public StoreInfo() {};
21
22          private int id;
23          public void setId(int id ) {this.id=id; }
24          public int getId() { return id; }
25
26          private String name;
27          public void setName( String name ) { this.name = name; }
28          public String getName() { return name; }
29
30          private String add;
31          public void setAdd( String add ) { this.add = add; }
32          public String getAdd() { return add; }
33
```

アクセッサ
フィールド　xxxx　⇒　setXxxx(),　getXxxx()

JavaBeansの例

- お店データベースの情報を管理

StoreInfo.java

```java
package nuno.javaAdv.test.first;

import java.io.Serializable;

import java.util.ArrayList;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;

import javax.sql.DataSource;

public class StoreInfo implements Serializable {

        public StoreInfo() {};

        private int id;
        public void setId(int id ) {this.id=id; }
        public int getId() { return id; }

        private String name;
        public void setName( String name ) { this.name = name; }
        public String getName() { return name; }

        private String add;
        public void setAdd( String add ) { this.add = add; }
        public String getAdd() { return add; }

        public int insertStoreInfo() {

                int result = 0;
                Connection db = null;
                PreparedStatement ps =null;
                try {
                        Context context = new InitialContext();
                        DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
                        db = ds.getConnection();
                        ps = db.prepareStatement("INSERT INTO list_t ( name_c, add_c ) VALUES(? , ? )");
                        ps.setString(1, name );
                        ps.setString(2, add );
                        result = ps.executeUpdate();
                        ps.close();
                        db.close();
                } catch (NamingException e) {
                        e.printStackTrace();
                } catch (SQLException e) {
                        e.printStackTrace();
                } finally {
                        try {
                                if(ps!=null) { ps.close(); }
                                if(db!=null) { db.close(); }
                        } catch (SQLException e) {
                                e.printStackTrace();
                        }
                }
                return result;
        }
```

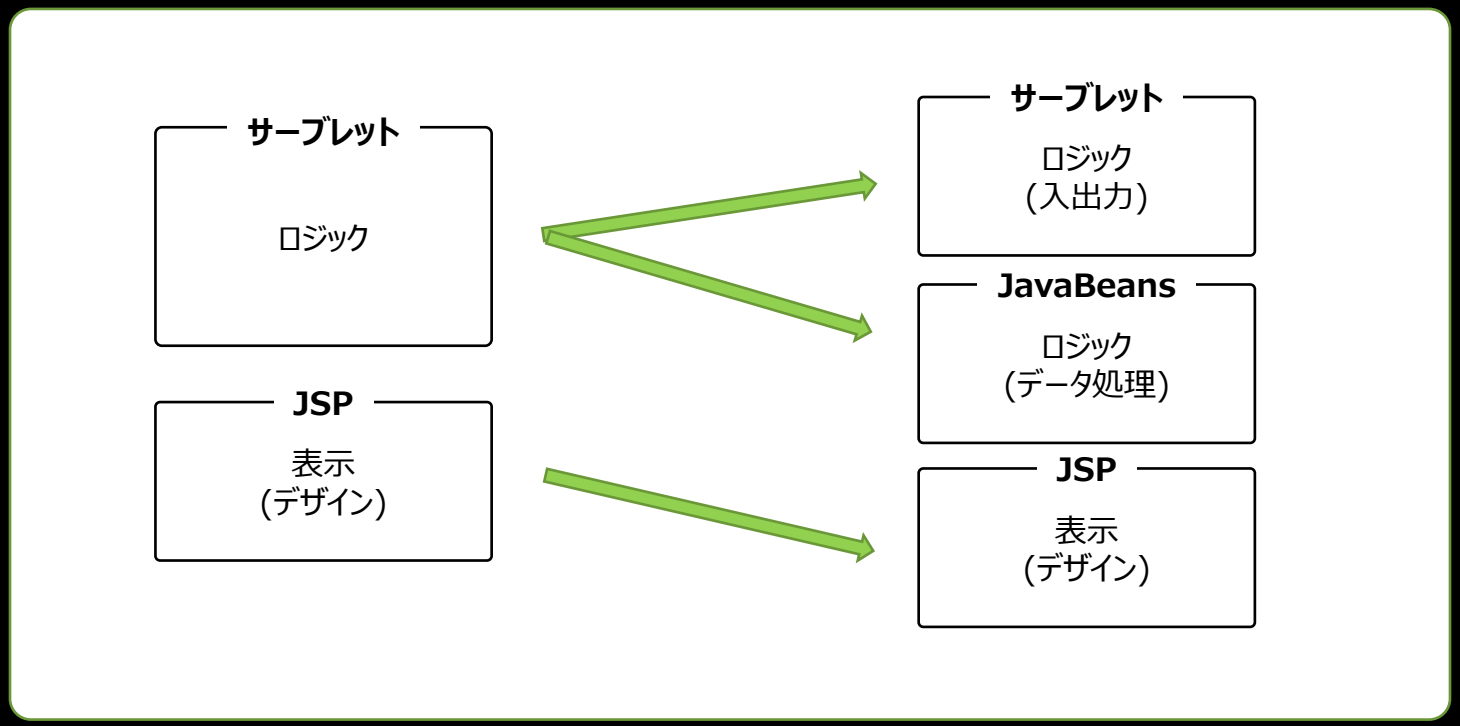
JavaBeansの例 (つづき)

- お店データベースの情報を管理

```
63
64      public int updateStoreInfo() {
65
66              int result = 0;
67              Connection db = null;
68              PreparedStatement ps =null;
69              try {
70                      Context context = new InitialContext();
71                      DataSource ds = (DataSource)context.lookup("java:comp/env/jdbc/shop_db");
72                      db = ds.getConnection();
73                      ps = db.prepareStatement("UPDATE list_t SET name_c=?, add_c=? WHERE id_c=?" );
74                      ps.setString(1, name );
75                      ps.setString(2, add );
76                      ps.setInt(3, id );
77                      result = ps.executeUpdate();
78                      ps.close();
79                      db.close();
80              } catch (NamingException e) {
81                      e.printStackTrace();
82              } catch (SQLException e) {
83                      e.printStackTrace();
84              } finally {
85                      try {
86                              if(ps!=null) { ps.close(); }
87                              if(db!=null) { db.close(); }
88                      } catch (SQLException e) {
89                              e.printStackTrace();
90                      }
91              }
92              return result;
93      }
94  }
95
```

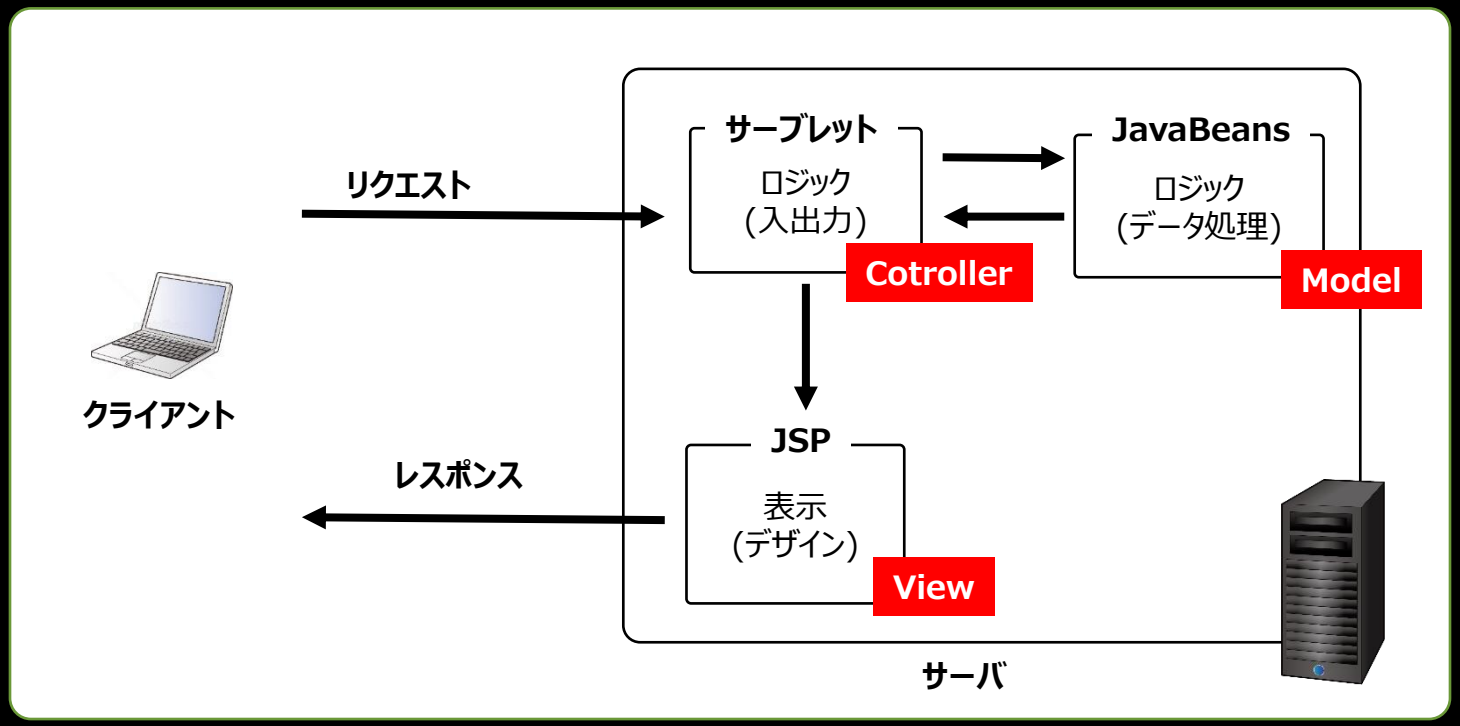
役割分担と連携

- ロジック関連処理を①入出力(入力データ受け取り、処理結果受け渡し)と②データ処理に分離
- ①⇒サーブレット、　②⇒JavaBeans
- JSP：ユーザインタフェース
- サーブレット、JavaBeans、JSPが連携して一つのアプリケーションを形成



役割分担と連携　クライアントとのやり取り

- MVCモデル
- Controller　→　サーブレット：　リクエスト受け取り、制御
- Model　→　JavaBeans：　データの処理
- Veiw　→　JSP：　サーブレットからの情報でHTMLを作成・出力
- メリット：独立性が高い。開発分担が容易。コードの煩雑化抑制。コード再利用可



JavaBeans の受け渡し（Servlet)

- ServletからJSPにJavaBeansを用いて情報を渡す場合、requestオブジェクトのsetAttributeメソッドを用いて渡したい変数をリクエスト属性に登録し、requestオブジェクトを介して渡す。

HttpServletRequestオブジェクト.setAttribute(属性名, 変数)

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletExceptio

        request.setCharacterEncoding("UTF-8");
        String idStr= request.getParameter("id");

        StoreInfo info = StoreInfo.getInfo( idStr );

        request.setAttribute("attr_info",info);

        getServletContext().getRequestDispatcher(/disp.jsp).forward(request,response);
    }
```

JavaBeans の受け渡し（JSP(2)）

- JavaBeansを用いてServletから情報をJSPで受け取る場合、アクションタグの一つである <jsp:useBean>要素を利用する。

```
<jsp:useBean  id="オブジェクトを格納する変数"
   class="取得するクラスの完全限定名"  scope="JavaBeansの有効範囲" />
```

```
1  <%@ page language="java" contentType="text/html; charset=UTF-8"
2      pageEncoding="UTF-8"%>
3   <jsp:useBean id="attr_info" class="javaAdv.StoreInfo" scope="request" />
4
5  <!DOCTYPE html>
6  <html>
7  <head>
8  <meta charset="UTF-8">
9  <title>info for id</title>
10 </head>
11 <body>
12 <p>ID: <%=attr_info.getId() %></p>
13 <p>name: <%=attr_info.getName() %></p>
14 <p>add: <%=attr_info.getAdd() %></p>
15 </body>
16 </html>
```

JavaBeans のスコープ

- JavaBeansのスコープ：JavaBeansが有効である範囲
- page, request, session, application の４つのスコープがある。

スコープ	概要
page	現在のページが処理されている間のみ有効。他のページと共有しない。
request	現在のページからフォワード、インクルードされるページで有効
session	セッションの範囲内でBeansを保持。ユーザが現在のアプリケーション内を行き来する間、有効
application	現在のアプリケーションにアクセスしている全ユーザの間で共有

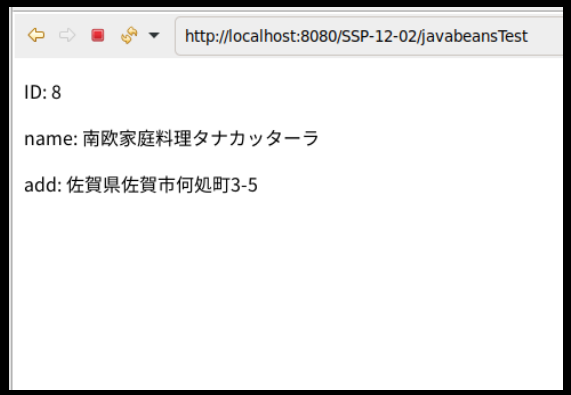
- <jsp:useBean>要素でJavaBeansを呼び出す際のコンテナ(Tomcat)の動作

① id属性とscope属性が一致するJavaBeansが既に存在する場合　⇒　引き続き、そのJavaBeansを使用

② 一致するJavaBeansが存在しない場合　⇒　新たなインスタンスを生成

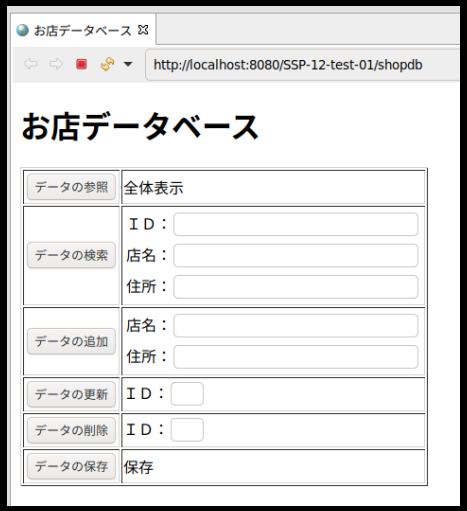
演習12-01　JavaBeansの基礎

- MySQLのレコードを読み出す。
- SOLAからファイルをダウンロード　(演習12-01.zip)
- 手順
  1. パッケージの作成
  2. javaファイル　→　パッケージ
  3. context.xml　→　META-INF
  4. JSPファイル　→　WebContent
- レコードの情報を表示し、表示画面をキャプチャする。
- キャプチャファイルをSOLAに提出



演習12-02　JSP&Servletを用いたお店データベース

- お店の店名、住所を管理するデータベース
- SOLAからファイルをダウンロード　(演習12-02.zip)
- 手順
  1. 動的Webプロジェクトを新規に作成　プロジェクト名：E13-02
  2. パッケージの作成　パッケージ名：shopDB
  3. javaファイル　→　src/shopDB
  4. context.xml　→　META-INF
  5. JSPファイル　→　WebContent
- 「データの参照」ボタンを押し、データベースの内容を表示
- 表示画面をキャプチャーしキャプチャーファイルをSOLAに提出



演習12-03　JSP&Servletを用いたお店データベース

- データを変更する shopdb-update.java を作成する。
- 変更画面を表示し、表示画面をキャプチャする。
- 「データの参照」ボタンを押し、データベースの内容を表示し、表示画面をキャプチャする。
- キャプチャーファイルをSOLAに提出

まとめ

- JavaBeans：汎用的に部品化できるよう幾つかの条件を満たしたクラス　⇒部品化のためのクラス
- JavaBeansのスコープ：　page, request, session, application
- ServletからJSPへのJavaBeansの受け渡し：　setAttribute(), forward()
- JSPでのJavaBeansの受け取り：　アクションタグ <jsp:useBean>
- MVCモデル：　Model →JavaBeans、View→JSP、Controller→Javaサーブレット　で役割分担
- MVCモデルのメリット：独立性の高いモジュール構成。開発分担が容易。コード煩雑化を抑制。コード再利用可　⇒メンテナンスしやすいアプリケーション構成

今後のスケジュール

第13回	7月12日	Servlet (3)
第14回	7月19日	まとめと課題

※ 定期試験はありません。

おわり

サーバシステムプログラミング及び演習

― 第13回 まとめと復習 ―

講義概要

◆Webアプリケーションの概念形成と，実践的なシステム構築スキルの養成

◆複数の言語，複数の部品を組み合わせ，統合する能力の習得

◆実践的な応用力を養成

【到達目標】

◆各講義で学習した技術をベースとして，これらを統合した実践的なweb開発の工程を習得すること

◆システム開発の考え方，進め方を知ること

◆システムエンジニアの仕事の工程を体験的に知り，即戦力の情報技術者となる基本能力を養成すること

授業内容

第1回　授業の概要と進め方

第2回　Linux

- 環境構築
- Linux概要
- Shellとコマンド
- ファイルシステム

第3回　LAMP

- LAMP概要　（Linux, Apache, MySql, PHP ）
- Webサーバ構成
- 通信プロトコル (IP, HTTP)

第4回　JavaScript

- クライアントサイド技術
- 文法基礎
- オブジェクト

第5回　PHP

- スクリプトの作成・実行方法
- 変数
- 文字列の連結
- 連想配列
- 関数の定義

第6回　HTTP

- リクエストとレスポンスの構成
- メソド (GET, POST, PUT,・・・)
- URLパラメータ

第7回　MySQL

- 実行方法
- データベース操作
- テーブル操作
- レコード操作
- PHPからのデータベースアクセス

授業内容

第８回　JSP (1)

- Javaサーブレット
- サーブレットコンテナ
- JSPの基本
- ディレクティブ
- スクリプレット

第９回　JSP (2)

- 宣言部
- 式 (Expression)
- アクションタグ

第１０回　JSP (3)

- JDBC (Java Database Connectivity)
- コネクションプーリング
- コンテキスト定義
- データベース操作

第11回　サーブレット (1)

- サーブレットとJSP
- アプリケーションクラス
- メソッドハンドラ
- インスタンス変数
- JSP連携
- JavaBeans

第１２　サーブレット (2)

- JavaBeansのスコープ
- JavaBeansの受け渡し
- MVCモデル

第13回　まとめと復習