Dynamic Web Site

Junghoo Cho

cho@cs.ucla.edu

Dynamic Web Site - Junghoo Cho - cho@cs.ucla.edu

Dynamic User Interaction

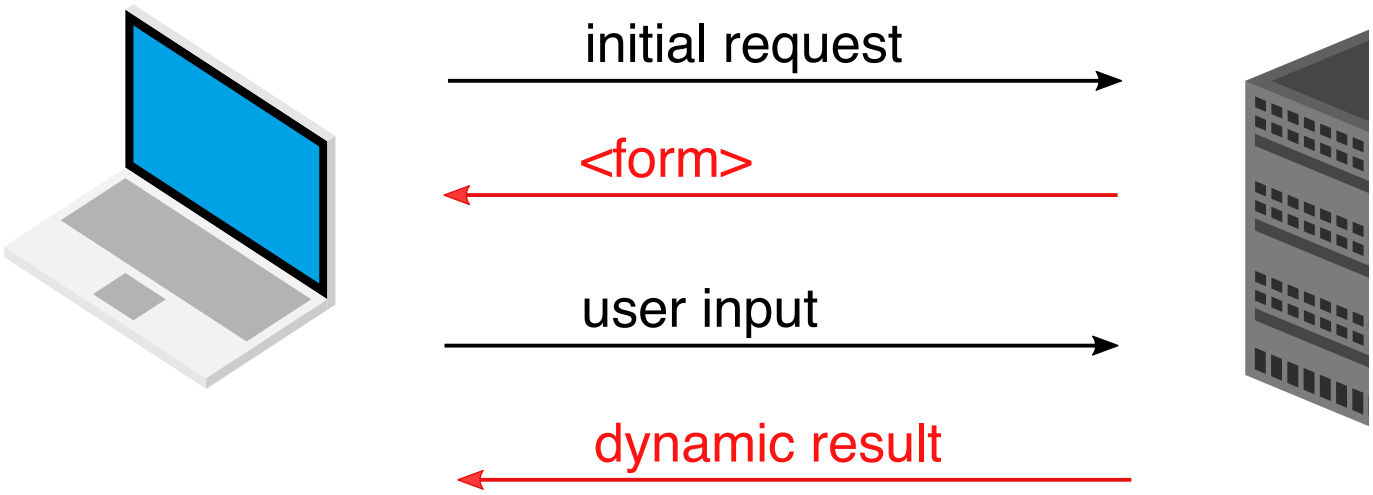
- Many sites generate contents dynamically based on "user in
  - e.g., search by keywords, facebook status update, …
- Q: How can a server obtain user input?

HTML Form

- An intuitive way to obtain user's input
- Example: Simple Google search box

```
<form action="http://www.google.com/search" method="GET">
    <input type="text" name="q">
    <input type="submit">
</form>
```

HTML Form Interaction



HTML Form: <form>

- `<form action="url" method="GET">`
  - Obtain user input and send it to the server
  - `action`: URL to which input is sent (default: .)
  - `method`: HTTP method to use
    - `GET` or `POST` allowed (default: `GET`)
  - Example:

```
<form action="http://www.google.com/search" method="GET">
```

HTML Form: <input>

- `<input type="type" name="name">`
  - Create a "user input" element of *type* with *name*
  - Enclosed inside `<form>`
  - Example:

```
<input type="text" name="q">
```

Input Encoding: GET vs POST

- User-provided input is sent as query "`name=value`" pairs of re
  - GET: query is added to *request path*
    - Example
  - POST: query is added to *request body*
    - `Content-Type: application/x-www-form-urlencoded`
    - Example

Input Types

- Common input types
- `hidden` type
  - Example:

    ```
    <input type="hidden" name="email" value="a@b.com">
    ```
  - Q: Why do we need this?
- HTML5 adds many more input types
  - `date`, `time`, `email`, `color`, `number`, ...

Uploading a File: `file` Type

- `<input type="file">`
- Encosing `<form>` should use:
  - `method="POST"`
  - `enctype="multipart/form-data"`
- Example

```
<form action="submit.php" method="POST" enctype="multipart/for
    Name: <input type="text" name="name">
    File: <input type="file" name="myfile">
    <input type="submit">
</form>
```

MIME: `multipart/form-data`

- Way to include multiple "objects" in a single message
- `boundary` attribute of `Content-Type` header
  - Specifyies "object boundary"
  - Example

```
Content-Type: multipart/form-data; boundary=--EndOfFile
```

- See HTTP request

Coding Dynamic Web Server

- Simple example: Hello, John!
- Q: How does it work? What happens at the server?
- Q: How should we write code to generate dynamic content a server?
- Two approaches
  - Programmatic vs Template
  - "Write a program!" vs "Write a Web page!"

Programmatic Approach

- Write a program that prints out the HTML page!
- Example: Java Servlet for "Hello, John!"

```
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException
{
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>Hello</title></head>");
    out.println("<body>Hello, " + request.getParameter("fname"
    out.println("</body>");
    out.println("</html>");
    out.close();
}
```

Template Approach

- Write an “HTML page” extended with “variable substitution”!
- Example: Java ServerPages (JSP)

```
<html>
<head><title>Hello</title></head>
<body>Hello, <%= request.getParameter("first_name") %>!
</body>
</html>
```

- Q: What do you like better?

Separating Code from Page

- Even with the template approach, the page gets messy quic complex application logic is added
- Q: Can we separate “code” from “page”?
  - Code “ownership”
    - Often, page design is done by designers, while app coding is done by develop
    - Who “owns” the above pages?
    - When multiple people “own” the same page, “conflicts” arise

Model-View-Controller (MVC) Pattern

- Most programs (including web sites) have to deal with
  - data
  - application logic, and
  - final result presentation

MVC Pattern: Data

- Data may be stored
  - in a file or database engine, and
  - locally or remotely
- Where and how data is stored and managed change over tin
- Let us create an abstract “data layer” and make it separate fr
  - Detailed storage mechanism is hidden from other layers
  - Changes in data layer do not affect other layers

MVC Pattern: Presentation

- The same "data" may be presented in many different ways
  - Depending on user, device, location, …
- Let us make "presentation layer" separate from others
  - Presentation changes do not affect other layers

MVC Pattern

- Split the code into three modular components!
  - *Model*: deals with data storage and access
  - *View*: deals with result presentation
  - *Controller*: deals with "application logic"
- Each component may be "owned" by different people
  - e.g., model: DB engineer, controller: app developer, view: UI designe

MVC Example: Controller

```
/*  CONTROLLER */
protected void doGet(HttpServletRequest request,
                     HttpServletResponse response)
    throws ServletException, IOException
{
    // retrive data
    user = getUser('id_34');

    // complex application logic here

    // dispatch data model to the view
    request.setAttribute("user_name", user.name);
    request.getRequestDispatcher("/index.jsp").forward(request, r
}
```

MVC Example: Model

```
/*  MODEL  */
User getUser(String userid)
{
    // retrieve and return the user data
}
```

MVC Example: View

```
/* VIEW */
<html>
<head><title>Demo</title></head>
<body>Hello, <%= request.getAttribute("user_name") %></body>
</html>
```

Special Tag Libraries

- Example: Java Standard Tag Libraries (JSTL)

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
<title>Contacts</title>
<body>
    <table>
        <c:forEach var="contact" items="${contacts}">
            <tr>
                <td>${contact.name}</td>
                <td>${contact.email}</td>
            </tr>
        <c:forEach>
    </table>
</body>
</html>
```

- Custom tags to enable looping, conditions, etc.

Four Layers of Web site

- Encryption layer: encrypt transport
- HTTP layer: interpret request and serve response
- Application layer: generate dynamic content
- Storage/Data layer: store and retrieve data

What We Learned

- HTML `form` and `input` elements
- Dynamic site: programmatic vs template approach
- MVC (Model-View-Controller) pattern
- Web server architecture