A Complete Web Server

This chapter is principally a lengthy illustration of the HTTP chapter, building a complete Web server in Go. It also shows how to use templates in order to use expressions in text files to insert variable values and to generate repeated sections.

Introduction

I am learning Chinese. Rather, after many years of trying I am still *attempting* to learn Chinese. Of course, rather than buckling down and getting on with it, I have tried all sorts of technical aids. I tried DVDs, videos, flashcards and so on. Eventually I realised that there *wasn't a good computer program for Chinese flashcards*, and so in the interests of learning, I needed to build one.

I had found a program in Python to do some of the task. But sad to say it wasn't well written and after a few attempts at turning it upside down and inside out I came to the conclusion that it was better to start from scratch. Of course, a Web solution would be far better than a standalone one, because then all the other people in my Chinese class could share it, as well as any other learners out there. And of course, the server would be written in Go.

The flashcards server is running at cict.bhtafe.edu.au:8000. The front page consists of a list of flashcard sets currently available, how you want a set displayed (random card order, Chinese, English or random), whether to display a set, add to it, etc. I've spent too much time building it - somehow my Chinese hasn't progressed much while I was doing it... It probably won't be too exciting as a program if you don't want to learn Chinese, but let's get into the structure.

Static pages

Some pages will just have static content. These can be managed by a `fileServer`. For simplicity I put all of the static HTML pages and CSS files in the `html` directory and all of the JavaScript files in the `jscript`directory. These are then delivered by the Go code

```
fileServer := http.FileServer("jscript", "/jscript/")
http.Handle("/jscript/", fileServer)

fileServer = http.FileServer("html", "/html/")
http.Handle("/html/", fileServer)
```

Templates

The list of flashcard sets is open ended, depending on the number of files in a directory. These should not be hardcoded into an HTML page, but the content should be generated as needed. This is an obvious candidate for templates.

The list of files in a directory is generated as a list of strings. These can then be displayed in a table using the template

```
<table>
  {{range .}}
  <tr>
    <td>
      {{.}}
    </td>
  </tr>
</table>
```

The Chinese Dictionary

Chinese is a complex language (aren't they all :-( ). The written form is hieroglyphic, that is "pictograms" instead of using an alphabet. But this written form has evolved over time, and even recently split into two forms: "traditional" Chinese as used in Taiwan and Hong Kong, and "simplified" Chinese as used in mainland China. While most of the characters are the same, about 1,000 are different. Thus a Chinese dictionary will often have two written forms of the same character.

Most Westerners like me can't understand these characters. So there is a "Latinised" form called Pinyin which writes the characters in a phonetic alphabet based on the Latin alphabet. It isn't quite the Latin alphabet, because Chinese is a tonal language, and the Pinyin form has to show the tones (much like acccents in French and other European languages). So a typical dictionary has to show four things: the traditional form, the simplified form, the Pinyin and the English. For example,

Traditional	Simplified	Pinyin	English
好	好	hǎo	good

But again there is a little complication. There is a free Chinese/English dictionary and even better, you can download it as a UTF-8 file, which Go is well suited to handle. In this, the Chinese characters are written in Unicode but the Pinyin characters are not: although there are Unicode characters for letters such as 'ǎ', many dictionaries

including this one use the Latin 'a' and place the tone at the end of the word. Here it is the third tone, so "hǎo" is written as "hao3". This makes it easier for those who only have US keyboards and no Unicode editor to still communicate in Pinyin.

This data format mismatch is not a big deal: just that somewhere along the line, between the original text dictionary and the display in the browser, a data massage has to be performed. Go templates allow this to be done by defining a custom template, so I chose that route. Alternatives could have been to do this as the dictionary is read in, or in the Javascript to display the final characters.

The code for the Pinyin formatter is given below. Please don't bother reading it unless you are *really* interested in knowing the rules for Pinyin formatting.

```
package pinyin

import (
        "io"
        "strings"
)

func PinyinFormatter(w io.Writer, format string, value ...interface{}) {
        line := value[0].(string)
        words := strings.Fields(line)
        for n, word := range words {
                // convert "u:" to "ü" if present
                uColon := strings.Index(word, "u:")
                if uColon != -1 {
                        parts := strings.SplitN(word, "u:", 2)
                        word = parts[0] + "ü" + parts[1]
                }
                println(word)
                // get last character, will be the tone if present
                chars := []rune(word)
                tone := chars[len(chars)-1]
                if tone == '5' {
                        words[n] = string(chars[0 : len(chars)-1])
                        println("lost accent on", words[n])
                        continue
                }
                if tone < '1' || tone > '4' {
                        continue
                }
                words[n] = addAccent(word, int(tone))
        }
        line = strings.Join(words, ` `)
        w.Write([]byte(line))
}

var (
        // maps 'a1' to '\u0101' etc
        aAccent = map[int]rune{
```

```
                '1': '\u0101',
                '2': '\u00e1',
                '3': '\u01ce', // '\u0103',
                '4': '\u00e0'}
        eAccent = map[int]rune{
                '1': '\u0113',
                '2': '\u00e9',
                '3': '\u011b', // '\u0115',
                '4': '\u00e8'}
        iAccent = map[int]rune{
                '1': '\u012b',
                '2': '\u00ed',
                '3': '\u01d0', // '\u012d',
                '4': '\u00ec'}
        oAccent = map[int]rune{
                '1': '\u014d',
                '2': '\u00f3',
                '3': '\u01d2', // '\u014f',
                '4': '\u00f2'}
        uAccent = map[int]rune{
                '1': '\u016b',
                '2': '\u00fa',
                '3': '\u01d4', // '\u016d',
                '4': '\u00f9'}
        üAccent = map[int]rune{
                '1': 'ǖ',
                '2': 'ǘ',
                '3': 'ǚ',
                '4': 'ǜ'}
)

func addAccent(word string, tone int) string {
        /*
         * Based on "Where do the tone marks go?"
         * at http://www.pinyin.info/rules/where.html
         */

        n := strings.Index(word, "a")
        if n != -1 {
                aAcc := aAccent[tone]
                // replace 'a' with its tone version
                word = word[0:n] + string(aAcc) + word[(n+1):len(word)-1]
        } else {
                n := strings.Index(word, "e")
                if n != -1 {
                        eAcc := eAccent[tone]
                        word = word[0:n] + string(eAcc) +
                                word[(n+1):len(word)-1]
                } else {
                        n = strings.Index(word, "ou")
                        if n != -1 {
                                oAcc := oAccent[tone]
                                word = word[0:n] + string(oAcc) + "u" +
                                        word[(n+2):len(word)-1]
                        } else {
                                chars := []rune(word)
                                length := len(chars)
```

```
                        // put tone onthe last vowel
                L:
                        for n, _ := range chars {
                                m := length - n - 1
                                switch chars[m] {
                                case 'i':
                                        chars[m] = iAccent[tone]
                                        break L
                                case 'o':
                                        chars[m] = oAccent[tone]
                                        break L
                                case 'u':
                                        chars[m] = uAccent[tone]
                                        break L
                                case 'ü':
                                        chars[m] = üAccent[tone]
                                        break L
                                default:
                                }
                        }
                        word = string(chars[0 : len(chars)-1])
                }
        }
    }

    return word
}
```

How this is used is illustrated by the function `lookupWord`. This is called in response to an HTML Form request to find the English words in a dictionary.

```
func lookupWord(rw http.ResponseWriter, req *http.Request) {
        word := req.FormValue("word")
        words := d.LookupEnglish(word)

        pinyinMap := template.FormatterMap {"pinyin": pinyin.PinyinFormatter}
        t, err := template.ParseFile("html/DictionaryEntry.html", pinyinMap)
        if err != nil {
                http.Error(rw, err.String(), http.StatusInternalServerError)
                return
        }
        t.Execute(rw, words)
}
```

The HTML code is

```
<html>
  <body>
    <table border="1">
      <tr>
        <th>Word</th>
        <th>Traditional</th>
```

```
        <th>Simplified</th>
        <th>Pinyin</th>
        <th>English</th>
      </tr>
      {{with .Entries}}
      {{range .}}
      {.repeated section Entries}
      <tr>
        <td>{{.Word}}</td>
        <td>{{.Traditional}}</td>
        <td>{{.Simplified}}</td>
        <td>{{.Pinyin|pinyin}}</td>
        <td>
          <pre>
            {.repeated section Translations}
            {@|html}
            {.end}
          </pre>
        </td>
      </tr>
      {.end}
      {{end}}
      {{end}}
    </table>
  </body>
</html>
```

The Dictionary type

The text file containing the dictionary has lines of the form
traditional simplified [pinyin] /translation/translation/.../
For example,
好 好 [hao3] /good/well/proper/good to/easy to/very/so/(suffix indicating completion or readiness)/

We store each line as an `Entry` within the `Dictionary` package:

```
type Entry struct {
     Traditional string
     Simplified string
     Pinyin     string
     Translations []string
}
```

The dictionary itself is just an array of these entries:

```
type Dictionary struct {
      Entries []*Entry
}
```

Building the dictionary is easy enough. Just read each line and break the line into its various bits using simple string methods. Then add the line to the dictionary slice.

Looking up entries in this dictionary is straightforward: just search through until we find the appropriate key. There are about 100,000 entries in this dictionary: brute force by a linear search is fast enough. If it were necessary, faster storage and search mechanisms could easily be used.

The original dictionary grows by people on the Web adding in entries as they see fit. Consequently it isn't that well organised and contains repetitions and multiple entries. So looking up any word - either by Pinyin or by English - may return multiple matches. To cater for this, each lookup returns a "mini dictionary", just those lines in the full dictionary that match.

The Dictionary code is

```
package dictionary

import (
        "bufio"
        //"fmt"
        "os"
        "strings"
)

type Entry struct {
        Traditional  string
        Simplified   string
        Pinyin       string
        Translations []string
}

func (de Entry) String() string {
        str := de.Traditional + ` ` + de.Simplified + ` ` + de.Pinyin
        for _, t := range de.Translations {
                str = str + "\n    " + t
        }
        return str
}

type Dictionary struct {
        Entries []*Entry
}

func (d *Dictionary) String() string {
        str := ""
        for n := 0; n < len(d.Entries); n++ {
                de := d.Entries[n]
                str += de.String() + "\n"
        }
```

```
        return str
}

func (d *Dictionary) LookupPinyin(py string) *Dictionary {
        newD := new(Dictionary)
        v := make([]*Entry, 0, 100)
        for n := 0; n < len(d.Entries); n++ {
                de := d.Entries[n]
                if de.Pinyin == py {
                        v = append(v, de)
                }
        }
        newD.Entries = v
        return newD
}

func (d *Dictionary) LookupEnglish(eng string) *Dictionary {
        newD := new(Dictionary)
        v := make([]*Entry, 0, 100)
        for n := 0; n < len(d.Entries); n++ {
                de := d.Entries[n]
                for _, e := range de.Translations {
                        if e == eng {
                                v = append(v, de)
                        }
                }
        }
        newD.Entries = v
        return newD
}

func (d *Dictionary) LookupSimplified(simp string) *Dictionary {
        newD := new(Dictionary)
        v := make([]*Entry, 0, 100)

        for n := 0; n < len(d.Entries); n++ {
                de := d.Entries[n]
                if de.Simplified == simp {
                        v = append(v, de)
                }
        }
        newD.Entries = v
        return newD
}

func (d *Dictionary) Load(path string) {

        f, err := os.Open(path)
        r := bufio.NewReader(f)
        if err != nil {
                println(err.Error())
                os.Exit(1)
        }

        v := make([]*Entry, 0, 100000)
        numEntries := 0
        for {
```

```
                line, err := r.ReadString('\n')
                if err != nil {
                        break
                }
                if line[0] == '#' {
                        continue
                }
                // fmt.Println(line)
                trad, simp, pinyin, translations := parseDictEntry(line)

                de := Entry{
                        Traditional:  trad,
                        Simplified:   simp,
                        Pinyin:       pinyin,
                        Translations: translations}

                v = append(v, &de)
                numEntries++
        }
        // fmt.Printf("Num entries %d\n", numEntries)
        d.Entries = v
}

func parseDictEntry(line string) (string, string, string, []string) {
        // format is
        //    trad simp [pinyin] /trans/trans/.../
        tradEnd := strings.Index(line, " ")
        trad := line[0:tradEnd]
        line = strings.TrimSpace(line[tradEnd:])

        simpEnd := strings.Index(line, " ")
        simp := line[0:simpEnd]
        line = strings.TrimSpace(line[simpEnd:])

        pinyinEnd := strings.Index(line, "]")
        pinyin := line[1:pinyinEnd]
        line = strings.TrimSpace(line[pinyinEnd+1:])

        translations := strings.Split(line, "/")
        // includes empty at start and end, so
        translations = translations[1 : len(translations)-1]

        return trad, simp, pinyin, translations
}
```

Flash cards

Each individual flash card is of the type `Flashcard`

```
type FlashCard struct {
        Simplified string
        English    string
        Dictionary *dictionary.Dictionary
}
```

At present we only store the simplified character and the english translation for that character. We also have a `Dictionary` which will contain only one entry for the entry we will have chosen somewhere.

A set of flash cards is defined by the type

```
type FlashCards struct {
        Name      string
        CardOrder string
        ShowHalf  string
        Cards     []*FlashCard
}
```

where the `CardOrder` will be "random" or "sequential" and the `ShowHalf` will be "RANDOM_HALF" or "ENGLISH_HALF" or "CHINESE_HALF" to determine which half of a new card is shown first.

The code for flash cards has nothing novel in it. We get data from the client browser and use JSON to create an object from the form data, and store the set of flashcards as a JSON string.

The Complete Server

The complete server is

```
/* Server
 */

package main

import (
        "fmt"
        "io/ioutil"
        "net/http"
        "os"
        "regexp"
        "text/template"
)

import (
        "dictionary"
        "flashcards"
        "templatefuncs"
)

var d *dictionary.Dictionary
```

```
func main() {
	if len(os.Args) != 2 {
		fmt.Fprint(os.Stderr, "Usage: ", os.Args[0], ":port\n")
		os.Exit(1)
	}
	port := os.Args[1]

	// dictionaryPath := "/var/www/go/chinese/cedict_ts.u8"
	dictionaryPath := "cedict_ts.u8"
	d = new(dictionary.Dictionary)
	d.Load(dictionaryPath)
	fmt.Println("Loaded dict", len(d.Entries))

	http.HandleFunc("/", listFlashCards)
	//fileServer := http.FileServer("/var/www/go/chinese/jscript",
"/jscript/")
	fileServer := http.StripPrefix("/jscript/",
http.FileServer(http.Dir("jscript")))
	http.Handle("/jscript/", fileServer)
	// fileServer = http.FileServer("/var/www/go/chinese/html", "/html/")
	fileServer = http.StripPrefix("/html/",
http.FileServer(http.Dir("html")))
	http.Handle("/html/", fileServer)

	http.HandleFunc("/wordlook", lookupWord)
	http.HandleFunc("/flashcards.html", listFlashCards)
	http.HandleFunc("/flashcardSets", manageFlashCards)
	http.HandleFunc("/searchWord", searchWord)
	http.HandleFunc("/addWord", addWord)
	http.HandleFunc("/newFlashCardSet", newFlashCardSet)

	// deliver requests to the handlers
	err := http.ListenAndServe(port, nil)
	checkError(err)
	// That's it!
}

func indexPage(rw http.ResponseWriter, req *http.Request) {
	index, _ := ioutil.ReadFile("html/index.html")
	rw.Write([]byte(index))
}

func lookupWord(rw http.ResponseWriter, req *http.Request) {
	word := req.FormValue("word")
	words := d.LookupEnglish(word)

	//t := template.New("PinyinTemplate")
	t := template.New("DictionaryEntry.html")
	t = t.Funcs(template.FuncMap{"pinyin": templatefuncs.PinyinFormatter})
	t, err := t.ParseFiles("html/DictionaryEntry.html")
	if err != nil {
		http.Error(rw, err.Error(), http.StatusInternalServerError)
		return
	}
	t.Execute(rw, words)
}
```

```
type DictPlus struct {
	*dictionary.Dictionary
	Word     string
	CardName string
}

func searchWord(rw http.ResponseWriter, req *http.Request) {
	word := req.FormValue("word")
	searchType := req.FormValue("searchtype")
	cardName := req.FormValue("cardname")

	var words *dictionary.Dictionary
	var dp []DictPlus
	if searchType == "english" {
		words = d.LookupEnglish(word)
		d1 := DictPlus{Dictionary: words, Word: word, CardName:
cardName}
		dp = make([]DictPlus, 1)
		dp[0] = d1
	} else {
		words = d.LookupPinyin(word)
		numTrans := 0
		for _, entry := range words.Entries {
			numTrans += len(entry.Translations)
		}
		dp = make([]DictPlus, numTrans)
		idx := 0
		for _, entry := range words.Entries {
			for _, trans := range entry.Translations {
				dict := new(dictionary.Dictionary)
				dict.Entries = make([]*dictionary.Entry, 1)
				dict.Entries[0] = entry
				dp[idx] = DictPlus{
					Dictionary: dict,
					Word:       trans,
					CardName:   cardName}
				idx++
			}
		}
	}

	//t := template.New("PinyinTemplate")
	t := template.New("ChooseDictionaryEntry.html")
	t = t.Funcs(template.FuncMap{"pinyin": templatefuncs.PinyinFormatter})
	t, err := t.ParseFiles("html/ChooseDictionaryEntry.html")
	if err != nil {
		fmt.Println(err.Error())
		http.Error(rw, err.Error(), http.StatusInternalServerError)
		return
	}
	t.Execute(rw, dp)
}

func newFlashCardSet(rw http.ResponseWriter, req *http.Request) {
	defer http.Redirect(rw, req, "http:/flashcards.html", 200)

	newSet := req.FormValue("NewFlashcard")
```

```
        fmt.Println("New cards", newSet)
        // check against nasties:
        b, err := regexp.Match("[/$~]", []byte(newSet))
        if err != nil {
                return
        }
        if b {
                fmt.Println("No good string")
                return
        }

        flashcards.NewFlashCardSet(newSet)
        return
}

func addWord(rw http.ResponseWriter, req *http.Request) {
        url := req.URL
        fmt.Println("url", url.String())
        fmt.Println("query", url.RawQuery)

        word := req.FormValue("word")
        cardName := req.FormValue("cardname")
        simplified := req.FormValue("simplified")
        pinyin := req.FormValue("pinyin")
        traditional := req.FormValue("traditional")
        translations := req.FormValue("translations")

        fmt.Println("word is ", word, " card is ", cardName,
                " simplified is ", simplified, " pinyin is ", pinyin,
                " trad is ", traditional, " trans is ", translations)
        flashcards.AddFlashEntry(cardName, word, pinyin, simplified,
                traditional, translations)
        // add another card?
        addFlashCards(rw, cardName)
}

func listFlashCards(rw http.ResponseWriter, req *http.Request) {

        flashCardsNames := flashcards.ListFlashCardsNames()
        t, err := template.ParseFiles("html/ListFlashcards.html")
        if err != nil {
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }
        t.Execute(rw, flashCardsNames)
}

/*
 * Called from ListFlashcards.html on form submission
 */
func manageFlashCards(rw http.ResponseWriter, req *http.Request) {

        set := req.FormValue("flashcardSets")
        order := req.FormValue("order")
        action := req.FormValue("submit")
        half := req.FormValue("half")
        fmt.Println("set chosen is", set)
```

```
        fmt.Println("order is", order)
        fmt.Println("action is", action)

        cardname := "flashcardSets/" + set

        //components := strings.Split(req.URL.Path[1:], "/", -1)
        //cardname := components[1]
        //action := components[2]
        fmt.Println("cardname", cardname, "action", action)
        if action == "Show cards in set" {
                showFlashCards(rw, cardname, order, half)
        } else if action == "List words in set" {
                listWords(rw, cardname)
        } else if action == "Add cards to set" {
                addFlashCards(rw, set)
        }
}

func showFlashCards(rw http.ResponseWriter, cardname, order, half string) {
        fmt.Println("Loading card name", cardname)
        cards := new(flashcards.FlashCards)
        //cards.Load(cardname, d)
        //flashcards.SaveJSON(cardname + ".json", cards)
        flashcards.LoadJSON(cardname, &cards)
        if order == "Sequential" {
                cards.CardOrder = "SEQUENTIAL"
        } else {
                cards.CardOrder = "RANDOM"
        }
        fmt.Println("half is", half)
        if half == "Random" {
                cards.ShowHalf = "RANDOM_HALF"
        } else if half == "English" {
                cards.ShowHalf = "ENGLISH_HALF"
        } else {
                cards.ShowHalf = "CHINESE_HALF"
        }
        fmt.Println("loaded cards", len(cards.Cards))
        fmt.Println("Card name", cards.Name)

        //t := template.New("PinyinTemplate")
        t := template.New("ShowFlashcards.html")
        t = t.Funcs(template.FuncMap{"pinyin": templatefuncs.PinyinFormatter})
        t, err := t.ParseFiles("html/ShowFlashcards.html")
        if err != nil {
                fmt.Println(err.Error())
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }
        err = t.Execute(rw, cards)
        if err != nil {
                fmt.Println("Execute error " + err.Error())
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }
}
```

```
func listWords(rw http.ResponseWriter, cardname string) {
        fmt.Println("Loading card name", cardname)
        cards := new(flashcards.FlashCards)
        //cards.Load(cardname, d)
        flashcards.LoadJSON(cardname, cards)
        fmt.Println("loaded cards", len(cards.Cards))
        fmt.Println("Card name", cards.Name)

        //t := template.New("PinyinTemplate")
        t := template.New("ListWords.html")
        if t.Tree == nil || t.Root == nil {
                fmt.Println("New t is an incomplete or empty template")
        }
        t = t.Funcs(template.FuncMap{"pinyin": templatefuncs.PinyinFormatter})
        t, err := t.ParseFiles("html/ListWords.html")
        if t.Tree == nil || t.Root == nil {
                fmt.Println("Parsed t is an incomplete or empty template")
        }

        if err != nil {
                fmt.Println("Parse error " + err.Error())
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }
        err = t.Execute(rw, cards)
        if err != nil {
                fmt.Println("Execute error " + err.Error())
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }
        fmt.Println("No error ")
}

func addFlashCards(rw http.ResponseWriter, cardname string) {
        t, err := template.ParseFiles("html/AddWordToSet.html")
        if err != nil {
                fmt.Println("Parse error " + err.Error())
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }
        cards := flashcards.GetFlashCardsByName(cardname, d)
        t.Execute(rw, cards)
        if err != nil {
                fmt.Println("Execute error " + err.Error())
                http.Error(rw, err.Error(), http.StatusInternalServerError)
                return
        }

}

func checkError(err error) {
        if err != nil {
                fmt.Println("Fatal error ", err.Error())
                os.Exit(1)
        }
}
```

Other Bits: JavaScript and CSS

On request, a set of flashcards will be loaded into the browser. A much abbreviated set is shown below. The display of these cards is controlled by JavaScript and CSS files. These aren't relevant to the Go server so are omitted. Those interested can download the code.

```
<html>
  <head>
    <title>
      Flashcards for Common Words
    </title>

    <link type="text/css" rel="stylesheet"
          href="/html/CardStylesheet.css">
    </link>

    <script type="text/javascript"
            language="JavaScript1.2" src="/jscript/jquery.js">
      <!-- empty -->
    </script>

    <script type="text/javascript"
            language="JavaScript1.2" src="/jscript/slideviewer.js">
      <!-- empty -->
    </script>

    <script type="text/javascript"
            language="JavaScript1.2">
      cardOrder = RANDOM;
      showHalfCard = RANDOM_HALF;
    </script>
  </head>
  <body onload="showSlides();">
    <h1>
      Flashcards for Common Words
    </h1>
    <p>
        <div class="card">

          <div class="english">
            <div class="vcenter">
              hello
            </div>
          </div>

              <div class="pinyin">
                <div class="vcenter">
                  nǐ hǎo
                </div>

              </div>
```

```
            <div class="traditional">
              <div class="vcenter">
                你好
              </div>
            </div>

            <div class="simplified">
              <div class="vcenter">
                你好
              </div>

            </div>

            <div class ="translations">
              <div class="vcenter">
                hello <br />
                hi <br />
                how are you? <br />
              </div>

           </div>
        </div>
        <div class="card">
          <div class="english">
            <div class="vcenter">
              hello (interj., esp. on telephone)
            </div>
          </div>

            <div class="pinyin">

              <div class="vcenter">
                wèi
              </div>
            </div>

            <div class="traditional">
              <div class="vcenter">
                喂
              </div>
            </div>

            <div class="simplified">

              <div class="vcenter">
                喂
              </div>
            </div>

            <div class ="translations">
              <div class="vcenter">
                hello (interj., esp. on telephone) <br />
                hey <br />

                to feed (sb or some animal) <br />
              </div>
```

```
            </div>
        </div>
    </p>

    <p class ="return">
      Press &lt;Space&gt; to continue
        <br/>
      <a href="http:/flashcards.html"> Return to Flash Cards list</a>
    </p>
  </body>
</html>
```

Source: http://jan.newmarch.name/go/chinese/chapter-chinese.html