Il passaggio delle variabili e la gestione dello stato

di Gianni Tomasicchio - 25 giugno 2005

database sicurezza cookie Regex SQL

Quando si realizza un'applicazione per Internet, qualsiasi sia il linguaggio server-side che si andrà ad utilizzare, è estremamente importante affrontare con attenzione e consapevolezza quella che è la più significativa differenza tra la programmazione web e lo sviluppo di un normale software per computer desktop, ovvero la gestione dello stato.

Un comune programma per computer deve interagire direttamente con un unico utente e può contare su un quantitativo di memoria generalmente abbondante e soprattutto sempre disponibile durante tutta la sua esecuzione. La gestione dello stato, ovvero il ricordo di quelle informazioni legate alle precedenti operazioni effettuate dall'utente e necessarie al corretto prosieguo, avviene semplicemente memorizzando tali dati in apposite variabili (variabili di stato), sempre accessibili fino alla chiusura dell'applicazione.

Un'applicazione web invece interagisce con l'utente attraverso Internet. L'utente scrive l'indirizzo di un sito (URL) nel browser. A sua volta il browser, facendo uso del protocollo di comunicazione HTTP, invia una richiesta al server web nel quale risiede la pagina desiderata. Se viene richiesta una pagina PHP, questa viene dapprima eseguita, ovvero vengono eseguite le istruzioni in essa contenute, ed il relativo risultato viene inviato al computer che ne ha fatto richiesta. Quindi uno script PHP rimane in esecuzione esclusivamente per il tempo necessario all'elaborazione di una singola richiesta, ovvero alla realizzazione di un'unica pagina. Ovviamente tutti i dati elaborati durante l'esecuzione andranno persi al termine dello script e questo avviene per ogni pagina richiesta. Inoltre un'applicazione web può essere usata da più utenti contemporaneamente e questo fa sorgere diverse problematiche, tra cui la limitata disponibilità di risorse hardware per utente. L'impressione che un'applicazione web, ad esempio un negozio online, sia a nostra completa disposizione per tutto il tempo della navigazione è quindi solo illusoria.

La soluzione alla mancanza di stato che caratterizza il protocollo HTTP, responsabile dell'interazione tra client (utente) e server web, va cercata quindi non nel PHP in se ma nel protocollo stesso. I realizzatori del PHP hanno infatti sfruttato proprio alcune caratteristiche dell'HTTP per fornire agli sviluppatori diversi strumenti per superare in maniera brillante il problema. Questi metodi sono:

- get
- post
- cookie
- sessioni

Il primi tre metodi si basano esclusivamente su tre relativi metodi dell'HTTP (definiti quando il PHP ancora non esisteva) mentre le sessioni consistono in un abile utilizzo dei metodi get e cookie per la realizzazione di un potente sistema di persistenza dei dati. Nelle prossime pagine vedremo singolarmente l'utilizzo di queste tecniche, esponendo il relativo ambito di pertinenza e proponendo alcuni esempi.

Il metodo GET

Quando scriviamo l'indirizzo di un sito internet nella barra degli indirizzi del nostro browser oppure quando clicchiamo con il mouse su un link contenuto in una pagina web, stiamo utilizzando il metodo GET del protocollo di comunicazione internet HTTP. Questo metodo prevede l'invio da parte del browser di un URL (Uniform Resource Locator), ovvero di una stringa di questo tipo:

<protocollo>://<host>[:<porta>][<percorso>[?<query string >]]

Si tratta di quello che comunemente viene chiamato "indirizzo internet", ad esempio un ipotetico indirizzo potrebbe essere:

http://www.miosito.it/download/scarica.php?file=Mozilla+Firefox&sistema=windows

dove

- protocollo: http
- host: www.miosito.it
- porta: non presente, per default è la 80
- percorso: /download/scarica.php
- query string: file=Mozilla+Firefox&sistema=windows

Ma come possiamo sfruttare questo metodo per risolvere il problema della persistenza delle informazioni tra le pagine di una applicazione PHP? Quello che si fa è sfruttare la query string, ovvero il testo in coda all'URL delimitato dal punto interrogativo "?". In questa stringa è possibile infatti concatenare diverse variabili con i rispettivi valori separate dal carattere "&" nella forma

variabile1=valore1&variabile2=valore2

Nell'esempio precedente la query string contiene due variabili, una di nome "file" il cui valore è "Mozilla Firefox" e l'altra di nome "sistema" che invece vale "windows". Si noti come la stringa "Mozilla Firefox", a causa dello spazio che contiene, deve essere preventivamente codificata per poter essere usata all'interno di un URL. In questo caso lo spazio è stato sostituito da un "+".

Vediamo ora come poter sfruttare queste variabili all'interno dello script PHP. Continuando con l'esempio precedente, a seguito della richiesta del browser verrà eseguita sul server la pagina scarica.php. Il PHP, in maniera automatica e completamente trasparente, provvederà a leggere la query string, a decodificarla e a renderne disponibile il contenuto allo script sotto forma di array associativo dal nome $_GET. Nel nostro caso $_GET['file'] sarà uguale a "Mozilla Firefox" (notare l'assenza del "+") mentre $_GET['sistema'] varrà "windows".

Uno dei pregi di $_GET è quello di essere un **array superglobale**, ovvero una variabile che non segue le normali regole di visibilità ma è utilizzabile in qualsiasi punto dello script, anche all'interno delle funzioni personalizzate.

Ricapitolando, se una pagina **A** contiene un link ad una pagina **B** e nella query string di questo link ci sono delle variabili concatenate nel modo visto allora queste variabili saranno disponibili nella pagina **B**. Sarà quindi compito della pagina A generare degli URL contenenti le informazioni da far propagare alle pagine che l'utente potrà visitare successivamente.

Vediamo ora quali sono i pro ed i contro di questa tecnica e gli ambiti di utilizzo più appropriati. E' evidente infatti che il metodo GET permette un facile scambio di informazioni tra pagine consecutive, ma risulta laborioso applicarlo per realizzare la persistenza di alcune variabili per un intero sito. Sarebbe necessario infatti accodare ad ogni link presente nelle diverse pagine queste variabili ed i relativi valori. Questa procedura viene adoperata ad esempio da phpMyAdmin per conservare l'informazione della lingua da utilizzare nelle varie pagine.

Se da un lato il metodo GET è semplice da implementare, dall'altro espone tutte le informazioni da propagare direttamente all'utente che potrebbe facilmente alterarle, agendo ad esempio sull'URL presente nella barra degli indirizzi del browser. Uno script PHP che utilizza variabili passate via $_GET dovrà quindi controllare necessariamente la loro validità e gestire anche l'eventuale assenza, parziale o totale, di tali variabili. Un ulteriore limite del metodo GET è il numero massimo dei caratteri che può contenere una querystring: non più di 256.

Si noti infine come questa tecnica non utilizza risorse del server per la memorizzazione delle variabili che, accodate nell'URL, vengono conservate dal client. Questo facilita la scalabilità di una applicazione web ovvero non ne pregiudica il funzionamento all'aumentare dei client poiché all'aumento del numero degli utenti non corrisponde un aumento delle risorse necessarie alla memorizzazione di tali variabili.

Il metodo POST

Anche il metodo POST è una tecnica prevista nel protocollo HTTP. Essa permette al browser di inviare, ad una specifica pagina web sul server, le informazioni inserite dall'utente durante la compilazione di un form (pensate ad esempio a ciò che avviene quando effettuate il log-in in un sito).

Questo metodo viene usato in PHP principalmente per ottenere informazioni dall'utente, al quale abbiamo sottoposto un modulo da compilare, magari per effettuare una registrazione, o per permettergli di lasciare un messaggio (guestbook). Analogamente a quanto avviene per il metodo GET, i dati provenienti dal client con il metodo POST verranno "impacchettati" in un apposito array dal nome $_POST in maniera automatica e trasparente. Ad esempio, se nel nostro form era presente un campo input simile al seguente:

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

allora i dati inseriti ed inviati dall'utente saranno presentati allo script PHP nella variabile $_POST['nome_utente'].

Come l'array $_GET, anche l'array $_POST è superglobale, ovvero visibile e fruibile in qualsiasi punto dei nostri script.

Vediamo ora come sia possibile utilizzare questo metodo per conservare il valore di una variabile tra due pagine consecutive di una nostra applicazione PHP. Per farlo consideriamo il seguente scenario: vogliamo permettere ad un utente di poter modificare il testo di un messaggio che ha scritto in un forum. L'utente clicca su un link "modifica messaggio" e gli viene proposto un modulo contenente il vecchio testo che può modificare e reinviare premendo un bottone "invia". Ma adesso sorge un problema. Come fa lo script che riceve il nuovo testo a sapere quale vecchio messaggio modificare? Una soluzione potrebbe essere inserire nel form un campo input "nascosto", ad esempio:

```
<input name="id_messaggio" type="hidden" value="123">
```

Questo campo non sarà visibile all'utente che compila il modulo, ma sarà comunque inviato allo script insieme agli altri dati (il nuovo testo del messaggio) e sarà quindi disponibile nella variabile $_POST["id_messaggio"] che varrà appunto "123". La nostra applicazione quindi adesso sa che il messaggio da modificare è il n°123. Ricapitolando: la pagina **A** prepara un form contenente un campo nascosto che conserverà l'informazione che non vogliamo perdere, ovvero l'identificativo del messaggio. L'utente riceve il modulo, lo compila e lo reinvia alla pagina **B**. Questa avrà a disposizione non solo il messaggio modificato ma anche il suo identificativo.

Vediamo ora vantaggi e svantaggi del metodo POST nella gestione dello stato. Come risulta evidente dall'esempio precedente, questo metodo si presta bene al passaggio di variabili solo nel caso in cui la pagina di partenza e quella di arrivo siano collegate dall'invio di un form. Risulta quindi praticamente impossibile far propagare diverse variabili all'interno di una intera applicazione web (cosa possibile con il metodo GET, con i cookie e con le sessioni).

Rispetto al metodo GET ha comunque il vantaggio di poter trattare variabili più grandi di 256 caratteri e di non mostrarle all'utente (cosa impossibile con GET). Ciò non significa che queste non possano essere manipolate ad arte da utenti "intraprendenti", esse infatti transiteranno comunque nel browser.

Anche il metodo POST non richiede la memorizzazione sul server delle "variabili di stato" e quindi, come il metodo GET, non pone vincoli alla scalabilità dell'applicazione.

I cookies

Un cookie (dall'inglese: biscottino) è un piccolo insieme di dati, di dimensioni strettamente limitate (massimo 4 kilobyte), che può essere inviato insieme ad una pagina web. Queste informazioni vengono salvate nel browser, spesso sotto forma di file di testo, e conserva per un determinato intervallo di tempo durante il quale verranno rese disponibili al server, attraverso il loro continuo ed automatico reinvio, ad opera del browser, ad ogni nuova pagina web richiesta.

Ovviamente questi cookies sono legati al nome del dominio del server che gli ha inviati e sono quindi visibili solo alle pagine web appartenenti a tale dominio. Quindi i cookies inviati da una pagina del sito www.miosito.it non saranno fruibili dal sito www.tuodominio.it

Similmente a ciò che accade ai dati ricevuti dal server attraverso i metodi GET e POST, le informazioni presenti in un cookie vengono lette, decodificate ed inserite in un array superglobale di nome $_COOKIE. Per inviare un cookie invece occorre utilizzare la funzione setcookie(), che permette di specificare anche l'intervallo temporale di validità del cookie. Questi possono essere cancellati alla fine della sessione di navigazione (chiamati magic cookie e memorizzati nella memoria volatile del client) o possono essere conservati per un tempo ben determinato (cookie persistenti, salvati su disco rigido).

E' evidente quindi come i cookie siano uno strumento privilegiato nella gestione dello stato di un'applicazione web. Contrariamente a quanto accade con i metodi GET e POST infatti, non è necessario effettuare un palleggiamento di informazioni tra client e server, ma basta inviare una sola volta le informazioni che vogliamo rendere persistenti durante tutta la navigazione dell'utente, che queste saranno disponibili in ogni pagina del sito, semplicemente accedendo all'array $_COOKIE.

E' altrettanto vero però che i cookies, poiché risiedono sul client, possono essere cancellati o manipolati dall'utente. Raramente può anche capitare che le impostazioni di sicurezza del browser non permettano la ricezione dei cookie. Conservano quindi quell'intrinseca insicurezza che caratterizzava le informazioni propagate via GET e POST.

Vi siete mai chiesti come facciano certi siti, forum, programmi di posta, e-commerce a riconoscervi ad ogni vostro accesso, senza che dobbiate ripetere il log-in? E' tutto merito dei cookies.

Infine si noti come anche l'utilizzo dei cookies non richiede la memorizzazione sul server delle "variabili di stato" e quindi, come il metodo GET e POST, non pone vincoli alla scalabilità dell'applicazione.

Le sessioni

Le sessioni sono un potente strumento messo a disposizione da PHP per la gestione dello stato di una applicazione web. La principale differenza con i metodi precedentemente visti consiste nella conservazione delle variabili di stato, ovvero delle informazioni che vogliamo rendere persistenti, direttamente sul server. Le sessioni infatti utilizzano il metodo GET e i cookies solo per riconoscere l'utente, che sarà identificato da un codice (session id). Ed è proprio questo session id ad essere palleggiato con il browser, non i dati ad esso associati che verranno salvati sul server, normalmente sotto forma di files di testo.

Poiché le sessioni vengono realizzate in maniera trasparente dal PHP, non dovremo preoccuparci di nulla. Per memorizzare e prelevare le informazioni useremo direttamente l'array superglobale $_SESSION. Sarà il PHP, attivato dalla funzione session_start, a preoccuparsi di caricare in esso i dati di sessione e, al termine dello script, a salvarli sul server.

Per approfondire il funzionamento e l'utilizzo delle sessioni rimandiamo a questo articolo. Ciò che è importante sottolineare in questo contesto è che le sessioni trasferiscono il carico di memorizzazione sul server che dovrà conservare le variabili di stato di tutti gli utenti connessi. E questo, se da un lato potrebbe pregiudicare, o quantomeno complicare, la scalabilità dell'applicazione, dall'altro fa si che le sessioni siano lo strumento più sicuro tra quelli visti.

La potenza delle sessioni è dimostrata anche dalla possibilità di poter rendere persistenti non solo semplici numeri, stringhe o array ma anche interi oggetti semplicemente assegnandoli all'array $_SESSION. E' possibile inoltre personalizzare le procedure che si occupano della memorizzazione e del recupero delle variabili di sessione, che potranno così essere conservate ad esempio in un database.

Confronto tra le tecniche

Per concludere questa carrellata sulle tecniche per la propagazione delle variabili e la gestione dello stato riportiamo di seguito uno schema che riassume i punti di forza e debolezza di tali metodi. Con "persistenza a corto e lungo raggio" si è indicata la minore o maggiore difficoltà nel riuscire a far propagare i dati tra pagine consecutive (corto raggio) o pagine visualizzate dopo lunghi percorsi di navigazione.

Come si evince dalla tabella, non esiste la tecnica perfetta, che risolve tutti i problemi di persistenza dei dati. Ed infatti la maggior parte delle applicazioni PHP fa uso di più metodi contemporaneamente.

	GET	**POST**	**Cookie**	**Sessioni**
persistenza a corto raggio	facile	facile	medio	facile
persistenza a lungo raggio	medio-difficile	impossibile	facile	facile
capacità di memorizzazione	bassa	media	media	alta
carico sul server	nessuno	nessuno	nessuno	proporzionale agli utenti
sicurezza	bassissima	bassa	bassa	alta

register_globals e magic_quotes_gpc

E' importante tenere presente che le impostazioni presenti nel php.ini possono alterare il comportamento delle tecniche precedentemente viste per la comunicazione tra pagine PHP. Le principali direttive da considerare sono register_globals e magic_quotes_gpc.

La direttiva register_globals, settata ad On, comporta la creazione automatica, da parte di PHP, di tutte le variabili provenienti da GET, POST, Cookie (ed altre) . Facciamo un esempio: se lanciamo uno script con delle variabili nell'url come il seguente:

mia_pagina.php?variabile_cattiva=valore_pericoloso

nello script mia_pagina.php il PHP creerà automaticamente, oltre alla variabile $_GET['variabile_cattiva'], anche la variabile $variabile_cattiva. Questa tecnica viene usata per "avvelenare" gli script con delle variabili create ad arte ed è per questo motivo che è più sicuro impostare la direttiva register_globals ad off. Dalla versione 4.2 comunque le distribuzioni di PHP hanno questa impostazione ad off di default.

magic_quotes_gpc è una direttiva del php.ini che se impostata su On fa si che il PHP aggiunga, all'interno dei valori provenienti da GET, POST e dai cookie, dei backslash prima degli apici singoli (') e doppi ("). Se ad esempio attraverso un campo input di un form viene inviato il valore *l'università* questo verrà modificato in *l\'università*.

Il carattere di backslash inserito automaticamente prima dell'apostrofo permette di evitare diversi problemi di sicurezza, ad esempio nell'interazione con un db. La famigerata SQL Injection, ovvero quella tecnica di hacking per l'alterazione di una query, si basa proprio su questo principio. Anche se il dibattito sul valore appropriato da assegnare alla direttiva magic_quotes_gpc non ha portato ancora ad una conclusione condivisa da tutti, è buona norma, sopratutto per i meno esperti, lasciare questo parametro settato a On.

Sicurezza delle variabili di stato

Oltre ad impostare correttamente i parametri del php.ini visti nel precedente paragrafo, occorre realizzare degli accorgimenti per poter garantire la massima sicurezza delle variabili di stato, quando queste vendono conservate sul browser dell'utente.

Per prima cosa, in fase di progettazione di un'applicazione PHP, è necessario evidenziare quali siano le variabili che possiamo far propagare via GET, POST o cookies, senza compromettere la sicurezza dell'intero sistema. Un numero di carta di credito non rientrerà ovviamente in questo insieme.

Inoltre, nella realizzazione degli script dovremo sempre prevedere e gestire due situazioni:

- assenza delle variabili che ci aspettavamo
- manomissione di tali variabili da parte dell'utente

La prima evenienza è facilmente gestibile con la funzione isset. Ad esempio, se ci aspettiamo che via POST ci venga fornito il valore di un "id" potremo comportarci nel seguente modo:

```
<?php
if(isset($_POST['id']))
{
        // uso l'id passato via POST
}
else
{
        // gestisco la mancanza della variabile
}
?>
```

La manomissione è invece più difficile da rilevare e le tecniche che si usano si basano sul controllo del tipo di dato atteso. Ad esempio se la variabile $_POST['id'] dovrebbe contenere un numero si potrebbe fare un controllo del tipo:

```
if(is_numeric($_POST['id']))
```

Le funzioni che PHP mette a disposizione per effettuare questi controlli sono numerose. Da non sottovalutare anche l'uso delle espressioni regolari. Più sarà raffinato il controllo sulla tipologia del dato ricevuto, più il nostro script sarà sicuro. In generale, quando ci si accorge che i controlli da fare sono numerosi e complessi converrebbe usare le sessioni.

Conclusioni

Come si è visto, la gestione dello stato di una applicazione web è una questione complessa. Nonostante lo sviluppatore PHP abbia a disposizione diverse tecniche per poter realizzare la persistenza delle informazioni, nessuna di queste è fruibile a costo zero.

A complicare la situazione intervengono i diversi problemi di sicurezza che possono insorgere a causa dell'inevitabile interazione con l'utente. Non si tratta di debolezze del PHP ma di inadeguatezza del protocollo HTTP che sottende a tutta la comunicazione client-server.

Per tutti questi motivi è buona norma, in una applicazione web, ridurre al minimo le esigenze di persistenza dei dati, cercando però di non compromettere quell'accattivante illusione di navigazione interattiva creata dalla gestione dello stato.