5-Blocks, Concurrency, Networking

Closures

Una closure è una funzione (o una referense a una funzione) assieme ad un ambiente di riferimento, ovvere assieme ad un area dove le variabili non locali vengono memorizzate e possono essere accedute.

> Nota
> Una funzione è differente da una closure perché la funzione non più accedere a variabili non locali.

Esempio

```
final int rounds = 100;
Runnable runnable = new Runnable() {
        public void run() {
                for (int i = 0; i < rounds; i++) {
                // ...
                }
        }
};
new Thread(runnable).start();
```

Blocchi

I blocchi sono un estensione a livello di linguaggio, di C, Objective-C e C++, introdotti per supportare le closures.

La definizione di blocco inizia con `^{` e termina con `}`:

```
^{
        // block body ...
        NSLog(@"Within block...");
}
```

> Nota
> È possibile assegnare un blocco ad una variabile

Un blocco può essere eseguito, solo se è una funzione C `block();`, e come loro possono ricevere degli argomenti e restituire un valore. La sistanssi di un blocco si presenta `return_type(^block_name)(arguments);`

Block type definition	Input arguments	Return type
`void(^BlockA)(void);`	-	-
`int(^BlockB)(int);`	one `int`	one `int`
`int(^BlockC)(int,int);`	two `int`s	one `int`
`void(^BlockD)(double);`	one `double`	-

Definire un blocco

```
typedef int(^ReturnIntBlock)(int);

ReturnIntBlock square = ^(int val){
        return val * val;
};
int a = square(10);
// ...
ReturnIntBlock cube = ^(int val){
        return val * val * val;
};

int b = cube(10);
```

Esempio

```
int(^max)(int, int) = ^(int a, int b){
        return (a > b ? a : b);
};
int maximum = max(first,second);
```

Se un blocco si trova all'interno di un metodo la visibilità delle varaibili per il blocco si ferma alla solo lettura delle variabili del metodo

```
- (void)aMethod{
        // ...
        int var = 10;
        void(^example)(void) = ^{
                NSLog(@"Accessing variable %d",var);
        };
        // ...
}
```

Le variabili che possono essere modificate nei blocchi devono presentare il flag `__block`, in questo modo la variabile diventa visibile anche tra gli scope di enclosing

```
- (void)aMethod{
        // ...
        __block int var = 10;
        void(^example)(void) = ^{
        NSLog(@"Accessing variable %d",++var);
        };
        // ...
        var = 100;
        example(); // prints 101
}
```

Quindi i blocchi sono oggetti, che pososno essere passati come argomenti a metodi e funzioni:

```
- (void)downloadFromUrl:(NSURL *)url completion:(void(^)(NSData*))callback;
// corrisponde a
typedef void(^DownloadBlock)(NSData*);
- (void)downloadFromUrl:(NSURL *)url completion:(DownloadBlock)callback;
```

L'ultimo elemento di questo metodo è un blocco, che prende un `NSData*` e restituisce nulla

Implementazione del metodo

```
- (void)downloadFromUrl:(NSURL *)url completion:(void(^)(NSData*))callback{
        // download from the URL
        NSData *data = ...;
        // ...
        callback(data);
}
```

Invocazione

```
void(^DownloadCompleteHandler)(NSData *) = ^(NSData *data){
        NSLog(@"Download complete [%d bytes]!", [data length]);
};
// ...
[self downloadFromUrl:url completion:DownloadCompleteHandler];
```

Concurrency

È molto importante essere in grado di eseguire molte task in parallelo, il thread principale è quello dell'applicazione, che è responsabile di:

- gestire gli eventi generati dall'utente
- gestire la UI

Una task che richiede molto tempo o una quantità di tempo non definita non deve mai essere eseguita nel thread principale, perché questo deve rimanere "libero" per riuscire a mantenere l'applicazione responsive.

Per riuscire a fare ciò sono presenti i background threads.

> Nota
> I thread devono essere coordinati per evitare che il programma raggiunga uno stato di inconsistenza.

iOS fornisce dei meccanismi per lavorare con i thread (UNIX-based), i thread sono features low-level che permettono la concorrenza in un programma. Comunque i thread introducono maggiore complessità all'interno del programma perchè necessitno di:

- singronizzazzione
- scalabilità
- condivisione di strutture dati

Threads in iOS

iOS fornisce diversi metodi per creare e gestire i threads (POSIX, implementazione mediante C).
Il Cocoa framework fornisce una classa Objective-C per i thread, `NSThread`.

```
[NSThread detachNewThreadSelector:@selector(threadMainMethod:) toTarget:self withObject:nil];
```

`NSObject` permette la generazione dei threads `[obj performSelectorInBackground:@selector(aMethod) withObject:nil];`.

> Nota
> L'utilizzo dei thread è scoraggiato, dato che iOS fornisce altri metodiu che possono essere utilizzati per ottenere la concorrenza in un programma.

Alternative ai threads

Invece di utilizzare i thread iOS adotta un approccio asyncrono:

1. Aquisice un background thread
2. Inizia la task sul quel thread
3. Invia una notifica al chiamante quando la task è completa (callback function)

Una funzione asincrona fa alcuni lavori dietro la scena per avviare una task, però ritorna prima the la task termini.

Grand Center Dispatch

Grand Dispatch Central (GCD) è una delle funzioni che iOS fornisce per avviare task in modo a sincrono. GCD è una libreria C che:

- sposta la gestione dei thread a livello di sistema
- si occupa di creare i thread necessari e schedulare i task da eseguire su quei threads
- si basa sulle dispatch queues, che sono basate su meccanismi del C per l'esecuzione di task custom

Dispatch queues

Una dispatch queue esegui i task, sia in modo sincrono sia in modo a sincrono, con un ordine di tipo FIFO.
Queste code possono essere di due tipi:

- seriali, che eseguono un task alla volta, ogni volta prima di avviare una task attendono che la precedente sia terminata
- concorrenti, che eseguono moolte task senza attente il completamento della precedente

> Nota
> Le task inviate alla coda di dispatch devono essere incapsulate dentro a delle funzioni o dentro a dei blocchi oggetto.

Operation queues

Le operation queue in Objective-C sono equivalenti alle concurrent dispatch queue fornite da Cocoa (implementato dalla classe `NSOperationQueue`).

> Non sono hanno necessariamente un ordine di tipo FIFO.

Le operazioni aggiunte ad una operation queue sono instanze della classe `NSOperation`, possono essere configurate per definire dipendenze, così da variare l'ordine delle loro esecuzioni.

Esempio

```
// prendere la coda principale
dispatch_queue_t mainQueue = dispatch_get_main_queue();

// creare una coda
dispatch_queue_t queue = dispatch_queue_create("queue_name", NULL);

// eseguire una task in modo ascincrono su una cosa
dispatch_async(queue, block)

// eseguire una task in modo asincrono sulla coda principale
dispatch_async(dispatch_get_main_queue(), block);

// rilascio della coda se non viene utilizzato l'ARC (Automatic Reference Counting)
dispatch_release(queue);
```

```
dispatch_queue_t queue = dispatch_queue_create("queue_name", NULL);
dispatch_async(queue, ^{
        // long running task...
        dispatch_async(dispatch_get_main_queue(), ^{
                // update UI in main thread...
        });
});
```

Networking

iOS fornisce delle API per il networking su molti livelli che permettono agli sviluppatori di:

- eseguire richieste HTTP/HTTPS (`get/post`)
- stabilire una connessione con un host remoto, con o senza criptaggio o autenticazione
- ascoltare per connessioni in ingresso
- mandare e ricevere dati mediante i protocolli senza connessione
- pubblicare, navigare e risolvere i servizi network con Bonjour

Molte applicazione fanno affidamento su dati e servizi forniti da host remoti, quindi il networking gioca un ruolo centrale nello sviluppo di applicazioni odierno.

> Come regola generale il networking dovrebbe essere dinamico, asincrono e non sul thread principale.

Lo stato della connettività potrebbe cambiare bruscamente, quindi le applicazioni dovrebbero essere progettate per essere robuste a mantenere l'utilizzabilità e l'esperienza utente. Un progetto che fa assunzioni sulla velocità della connessione è scorretto.

NSURL

Gli oggetti `NSURL` sono utilizzati per manipolare gli URLs e le rispettive risorse, e per far riferimento ad un file.

Creazione di istanze

```
NSURL *aUrl = [NSURL URLWithString:@"http://mobdev.ce.unipr.it"];
NSURL *bUrl = [[NSURL alloc] initWithString:@"http://mobdev.ce.unipr.it"];
NSURL *cUrl = [NSURL URLWithString:@"/path" relativeToURL:aURL];
```

Caricamento di URLs in NSString e NSData

Le istanze di `NSString` ed `NSData` possono essere create dalle informazioni ottente da URL remoti.

```
NSURL *url = ...;
NSError *error;

NSString *str = [NSString stringWithContentsOfURL:url encoding:NSUTF8StringEncoding
error:&error];
NSData *data = [NSData dataWithContentsOfURL:url options:NSDataReadingUncached error:&error];
```

- `stringWithContentsOfURL:encoding:error:` ritorna una stringa create leggendo dati da un URL dato ed interpretato utilizzando una codifica data
- `dataWithContentsOfURL:options:error:` crea e ritorna un oggetto contenente i dati provenienti dalla locazione specificata dal URL dato

Questi metodi sono sincroni, quindi sarebbe meglio eseguirli all'interno di una dispatch queue in background così da mantenere un design asincrono.

NSURLRequest

Gli oggetti `NSURLRequest` rappresentano una richiesta di caricamento URL indipendente dal protocollo e dallo schema URL.

Costruzione con NSURL dato

```
NSURLRequest *req = [NSURLRequest requestWithURL:url];
NSURLRequest *req2 = [[NSURLRequest alloc] initWithURL:url];
```

La sottoclasse mutabile è `NSMutableURLRequest`, una sua istanza può essere configurata per impostare proprietà indipendente dal protocollo e proprietà specifiche di HTTP:

- URL, timeout
- body, headers e metodi HTTP

NSURLConnection

Gli oggetti di `NSURLConnection` forniscono supporto per eseguire il caricamento, sincrono o asincrono, di una richiesta URL.

Sincrono

```
NSError *error;
NSURLResponse *response = nil;
NSData *data = [NSURLConnection sendSynchronousRequest:req
                                returningResponse:&response
                                error:&error];
```

Asincrono

```
[NSURLConnection
        sendAsynchronousRequest:req
        queue:[NSOperationQueue currentQueue]
        completionHandler:^(NSURLResponse *response, NSData *data, NSError *connectionError)
{
        // ...
        }
];
```

NSURLConnectionDelegate

Il protocollo `NSURLConnectionDelegate` fornisce metodi per ricevere callback informativi riguardanti il caricamento asincrono di una richiesta URL. I metodi delegati sono chiamati dal thread che inizia il caricamento asincrono per l'operazione associata all'oggetto `NSURLConnection`.

```
NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:req delegate:self];

// inizio caricamento
[conn start];
// fine caricamento
[conn cancel];
```

I protocolli `NSURLConnectionDelegate` e `NSURLConnectionDataDelegate` definiscono metodi che dovrebbero essere implementati dal delegato per un istanza di `NSURLConnection`.
Per riuscire a tener traccia dei progressi del caricamento dell'URL dovrebbero essere implementati i metodi:

```
- (void)connection:(NSURLConnection *)connection didReceiveData:(NSData *)data{
        NSLog(@"received %ld bytes", [data length]);
}
```

```
- (void)connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse
*)response{
        NSLog(@"received response");
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection{
        NSLog(@"loading finished");
}

- (void)connection:(NSURLConnection *)connection didFailWithError:(NSError *)error{
        NSLog(@"loading failed");
}
```

NSURLSession

La classe `NSURLSession` e le rispettive, forniscono un API per scaricare contenuti mediante HTTP. Questa API fornisce metodi per:

- supportare l'autenticazione
- dare all'applicazione l'abilità di eseguire download in background, quando non è in esecuzione o è in sospensione
- sono altamente asincrone
- richiedono un blocco del gestore di completamento (che restituisce i dati all'app quando un trasferimento termina correttamente o con un errore)
- forniscono le proprietà di stato e di avanzamento, oltre a fornire queste informazioni ai delegati
- supportano la cancellazione, il riavvio e la sospensione della richiesta, forniscono anche la possibilità di riavviare le sospensioni, cancellazioni o i download falliti da dove erano stati lasciati

> Il comportamento di un `NSURLSession` dipende dal tipo di sessione e dalla task

Tipi di sessione

Esistono tre tipi di sessioni, in base al tipo di oggetto di configurazione utilizzato per creare la sessione:

1. **Default session**, utilizza una cache persistente basata su disco e memorizzale credenziali nel portachiavi dell'utente.
2. **Ephemeral sessions**, non memorizza nessun dato nel disco; tutte le cache,le credenziali memorizzate, e molto altro sono memorizzate in ram e legate alla sessione, se l'applicazione invalida la sessione queste sono eliminate automaticamente
3. **Background sessions**, sono simili alle default sessions, ma un processo separato gestisce il trasferimento dei dati

> Esiste un altro tipo di sessione, la **shared session** è un unico `NSURLSession` che non può essere configurato e ca può essere usato per le richieste base. Può essere richiamato mediante il metodo `[NSURLSession sharedSession]`

Tipi di task

Esistono tre tipi di task supportati da una sessione:

1. **Data tasks,** manda e riceve dati utilizzando oggetti `NSData`
   - Le attività di dati sono destinate a richieste brevi, spesso interattive, dalla tua applicazione a un server
   - Le attività di dati possono restituire i dati alla tua applicazione un pezzo alla volta dopo che ogni pezzo di dati è stato ricevuto o tutti in una volta tramite un gestore di completamento
   - Le attività di dati non archiviano i dati in un file, quindi non sono supportate nelle sessioni in background, le attività dati non memorizzano i dati in un file, quindi non sono supportate nelle sessioni in background
2. **Download tasks**, reperisce dati sotto forma di un file, supporta i download in background mentre l'applicazione non è in esecuzione
3. **Upload tasks**, manda dati(solitamente sotto forma di file), e supporta i caricamenti in background mentre l'applicazione non è in esecuzione

Esempi di `Shared NSURLSession`

Data task

```
NSURLSessionTask *task = [[NSURLSession sharedSession]
                dataTaskWithRequest:req
                completionHandler:^(NSData * data, NSURLResponse * response, NSError * error)
{
                        NSLog(@"%@", [NSString stringWithUTF8String:[data bytes]]);
                }
];

[task resume];
```

Download task

```
NSURLSessionTask *task = [[NSURLSession sharedSession]
                downloadTaskWithRequest:req
                completionHandler:^(NSURL *location, NSURLResponse *response, NSError *error)
{
                        NSLog(@"%@", location.absoluteURL);
                }
];
[task resume];
```

Lavorare con `NSURLSession`

Le istanze di `NSURLSession` possono essere create per eseguire il suo blocco gestore di completamento nella coda principale o in un'altra coda.
Esecuzione nella main queue

```
NSURLSessionConfiguration *conf = [NSURLSessionConfiguration defaultSessionConfiguration];
NSURLSession *session = [NSURLSession
                sessionWithConfiguration:conf
                delegate:nil
                delegateQueue:[NSOperationQueue mainQueue] // impostazione della main queue
come delegato dell'esecuzione del blocco gestore
];

NSURLSessionDownloadTask *task;
task = [session
        downloadTaskWithRequest:request
        completionHandler:^(NSURL *localfile, NSURLResponse *response, NSError *error) {
        /* this block is executed on the main queue */
        /* UI-related code ok, but avoid long-running operations */
        }
];

[task resume];
```

Esecuzione in un altra coda

```
NSURLSessionConfiguration *conf = [NSURLSessionConfiguration defaultSessionConfiguration];
NSURLSession *session = [NSURLSession sessionWithConfiguration:conf]; // nessuna coda
delegata

NSURLSessionDownloadTask *task;
task = [session
        downloadTaskWithRequest:request
        completionHandler:^(NSURL *localfile, NSURLResponse *response, NSError *error) {
                /* non-UI-related code here (not in the main thread) */
                dispatch_async(dispatch_get_main_queue(), ^{
                        /* UI-related code (in the main thread) */
                });
        }
];
```

Socket e streams

Per rappresentare i flussi è presente la classe astratta `NSStream`, i sui oggetti forniscono modi semplici per leggere e scrivere dati da/in:

- memoria
- file
- network (sockets)

Gli oggetti `NSInputStream` sono usati per leggere byte da un flusso, mentre quelli `NSOutputStream` sono usati per scrivere byte su un flusso.

> Con gli stream è possibile scrivere codice di rete che funzioni a livello TCP, invece che a livello di applicazione (come il caricamento di URL HTTP/HTTPS).

Network Activity Indicator

Quando si eseguono task relative al network, come il caricamento o il download di contenuti, dovrebbe essere fornito un feedback all'utente. Per fare ciò si sfrutta un **network activity indicator** nella barra di stato, controllato mediante il metodo `networkActivityIndicatorVisible`.

```
[UIApplication sharedApplication].networkActivityIndicatorVisible = YES;
```

UIActivityIndicatorView

Un **activity indicator** è una ruota che gira che indica che un'attività è in esecuzione, dovrebbe essere mostrato per fornire un feedback all'utente per notificare che l'applicazione non è bloccata.

La classe che fornisce l'activity indicator view è `UIActivityIndicatorView`, un indicatore può:

- essere fatto partire/fermare con i metodi `startAnimating`/`stopAnimating`
- essere nascosto quando viene fermato, impostando la proprietà `hidesWhenStopped`
- essere personalizzato mediante le proprietà `activityIndicatorViewStyle` e `color`

UIImage

La classe `UIImage` è utilizzata per mostrare i dati di tipo immagine, provenienti da file o network.

```
UIImage *imgA = [UIImage imageNamed:@"image.png"];
NSData *data = ...;
UIImage *imgB = [UIImage imageWithData:data];
UIImage *imgC = [UIImage imageWithContentsOfFile:@"/path/to/file"];
```

> Gli oggetti di tipo immagine sono immutabili.

I tipi supportati sono:

- TIFF
- JPEG
- PNG
- GIF

Le proprietà sono:

- `imageOrientation`
- `scale`
- `size`

UIImageView

Un `UIImageView` mostra un immagine o un animata sequenza di immagini. Per mostrare una singola immagine è necessario configurare la "image view" con un `UIImage` appropriata. Mentre per molteplici immagini con le rispettive animazioni tra queste, deve essere configurato un oggetto `UIImageView` che può essere trascinato in una vista utilizzando un oggetto ‘palette’

L’immagine che deve essere mostrata può essere configurata nella storyboard, e le sue proprietà possono essere impostate programmaticamente.

Gli ”Image views” possono eseguire operazioni costose, che possono influire le performance dell’applicazione, come ad esempio cambiare le dimensioni di un immagine (scaling) od impostare la trasparenza per permetterne il mix con dei layer sottostanti.

Alcune possibili contromisure che possono avere un impatto positivo sulle performance sono:

- fornire un immagine già ridimensionata, se e dove possibile
- limitare le dimensioni delle immagini
- disabilitare l’alpha blending dove non necessario