Objective-C Блоки (Block)

Тип и литерал блока

```
typedef int (^MyBlock)(int);

int multiplier = 7;
MyBlock myBlock = ^(int num) {
   return num * multiplier;
};
```

ИЛИ

```
int multiplier = 7;
int (^myBlock)(int) = ^(int num) {
  return num * multiplier;
};
```

Вызов блока

```
{
   ...
   myBlock( 3 );
   //или
   if ( myBlock )
      myBlock( 3 );
}
```

Результат: 21

Контекст блока
1. примитивные типы

```
int multiplier = 7;

int (^myBlock)(int) = ^(int num) {
  return num * multiplier;
};

multiplier = 8;

NSLog( @"%d", myBlock( 3 ) );
```

Печатает: 21

Контекст блока
2. ключевое слово __block

```
__block int multiplier = 7;

int (^myBlock)(int) = ^(int num) {
   return num * multiplier;
};

multiplier = 8;
NSLog( @"%d", myBlock( 3 ) );
```

Печатает: 24

Контекст блока

3. переменные – указатели на объекты с подсчетом ссылок (id, NSObject)

```
NSDate* date = [ [ NSDate alloc ] init ];

void (^printDate)() = ^() {
  NSLog( @"date: %@", date );
};

//копируем блок в кучу
printDate = [ [ printDate copy ] autorelease ];
[ date release ];
printDate();
```

Контекст блока
4a. управление памятью

```
NSDate* date = [ [ NSDate alloc ] init ];

//создаем блок в стеке
void (^printDate)() = ^() {
  NSLog( @"date: %@", date );
};

[ date release ];
//копируем блок в кучу и падаем
printDate = [ [ printDate copy ] autorelease ];
```

Контекст блока
4b. управление памятью

```
__block NSDate* date = [ [ NSDate alloc ] init ];
void (^printDate)() = ^() {
   //здесь падаем при обращении к date
   NSLog( @"date: %@", date );
};
//копируем блок в кучу, для объекта date retain не вызывается
printDate = [ [ printDate copy ] autorelease ];

[ date release ];
printDate();
```

Блоки и управление памятью
1. отложенный вызов

```
void (^printDate)() = ^() {
   NSLog( @"Hello ☺" );
 };
```

```
//добавление в контейнер
printDate = [ [ printDate copy ] autorelease ];
[ NSMutableArray arrayWithObject: printDate ];
```

```
self.simpleBlock = printDate;
//всегда копируем block property
@property ( copy ) JFFSimpleBlock simpleBlock;
```

Блоки и управление памятью

2. block как результат функции

```
-(JFFSimpleBlock)example
{
    return [ [ ^
    {
        NSLog( @"test" );
    } copy ] autorelease ];
}
```

Блоки и управление памятью
3. Виды блоковых объектов

- Глобальные - без состояния
- Локальные - в стеке
- Malloc - Блоки в куче

Ios < 4.0 support:

1. PLBlocks - googlecode
2. ESBlocksRuntime – github

Управление памятью и Блоки

Automatic Reference Counting
No copy, release and autorelease

Блоки
Best practice

1. Работа с контейнерами на примере NSArray
2. Охраняющие выражения - guards
3. Отложенные вызовы:
   a) onDeallocBlock
   b) Scheduled operations
4. Блоки вместо делегатов в UIAlertView

NSArray
concurrent enumerate

```
NSArray* arr_ = [ NSArray arrayWithObjects: @"1"
                , @"2”
                , @"3”
                , nil ];

[arr_ enumerateObjectsWithOptions: NSEnumerationConcurrent
                       usingBlock: ^( id obj_
                                  , NSUInteger idx_
                                  , BOOL* stop_)
{
    NSLog( @"start process: %@", obj_ );
    sleep( 4 );
    NSLog( @"stop process: %@", obj_ );
} ];
```

NSArray
Строгая типизация vs NSPredicate

```
NSArray* array_ = [ NSArray arrayWithObjects: @"1"
                  , @"2"
                  , @"3"
                  , nil ];

[ array_ indexOfObjectPassingTest: ^( id obj_
                                   , NSUInteger idx_
                                   , BOOL* stop_)
{
   NSString* element_ = obj_;
   return [ element_ isEqualToString: @"2" ];
} ];
```

JFFLibrirary's NSArray расширения

JFFLibrirary github

```
+(id)arrayWithSize:( NSUInteger )size_
        producer:( ProducerBlock )block_;

-(void)each:( ActionBlock )block_;
-(NSArray*)map:( MappingBlock )block_;
-(NSArray*)select:( PredicateBlock )predicate_;
-(NSArray*)flatten:( FlattenBlock )block_;
-(NSUInteger)count:( PredicateBlock )predicate_;

-(id)firstMatch:( PredicateBlock )predicate_;

-(void)transformWithArray:( NSArray* )other_
              withBlock:( TransformBlock )block_;
```

Охраняющие выражения – guards

```
{
   [ self beginUpdates ];

   //update rows here

   //здесь ошибка если condition_ == true, мы не вызовем    endUpdates
   if ( condition_ )
      return;

   //update rows here

   [ self endUpdates ];
}
```

Охраняющие выражения – guards

```
-(void)withinUpdates:( void (^)( void ) )block_
{
   [ self beginUpdates ];

   @try
   {
      block_();
   }
   @finally
   {
      [ self endUpdates ];
   }
}
```

Охраняющие выражения – guards

```
{
  [ self.tableView withinUpdates: ^( void )
  {
    //update rows here

    if ( condition_ )
      return;

    //update rows here
  } ];
}
```

Отложенные вызовы
onDeallocBlocks

```
-(void)dealloc
{
   [ [ NSNotificationCenter defaultCenter ] removeObserver: self ];
   //release ivars here if NO ARC
   [ super dealloc ];
}
```

или

```
-(void)dealloc
{
   [ self cancelSomeOperations ];
   //release ivars here if NO ARC
   [ super dealloc ];
}
```

Отложенные вызовы
onDeallocBlocks

1. objc_setAssociatedObject(self
, &ownerships_key_
, ownerships_
, RETAIN_NONATOMIC);

2. Class JFFOnDeallocBlockOwner

```
-(void)dealloc
{
   if ( _block )
   {
      _block();
      [ _block release ];
   }

   [ super dealloc ];
}
```

Отложенные вызовы
onDeallocBlocks

```
-(void)addOnDeallocBlock:( void(^)( void ) )block_
{
  JFFOnDeallocBlockOwner* owner_ =

  [ [ JFFOnDeallocBlockOwner alloc ] initWithBlock:
              block_ ];

  [ self.ownerships addObject: owner_ ];

  [ owner_ release ];
}
```

Отложенные вызовы
onDeallocBlocks

```
//лечим циклическую ссылку
__block id self_ = self;
[ self addOnDeallocBlock: ^
{
   [ [ NSNotificationCenter defaultCenter ] removeObserver: self_ ];
} ];
```

Отложенные вызовы

Scheduled operations

```
[ self performSelector: @selector( someMethod )
            withObject: nil
            afterDelay: 20. ];
[ NSObject cancelPreviousPerformRequestsWithTarget: self ]; //отмена
```

или

```
[ NSTimer scheduledTimerWithTimeInterval: 20.
                  target: self
                selector: @selector( someMethod )
                userInfo: nil
                 repeats: YES ];
[ timer_ invalidate ]; //отмена
```

Отложенные вызовы
Scheduled operations

```
__block id self_ = self;

JFFScheduledBlock bk_= ^
{
   [ self_ someMethod ];
}
CancelBlock cancel_ = [ JFFScheduler addBlock: bk_
                              duration: 20. ];

[ self addOnDeallocBlock: cancel_  ];
```

Блоки вместо делегатов в UIAlertView

```
-(void)alertView:( UIAlertView* )alert_view_ clickedButtonAtIndex:( NSInteger )button_index_
{
  NSString* title_ = [ alert_view_ buttonTitleAtIndex: button_index_ ];
  if ( [title_ isEqualToString: cancel_ ] )
        //..
  else if ( [ title_ isEqualToString: button1_ ] )
        //..
  else if ( [ title_ isEqualToString: button2_ ] )
        //..
}
```

Блоки вместо делегатов в UIAlertView

```
JFFAlertButton* bt_ = [ JFFAlertButton alertButton: title_
                                            action: ^
{
   //do some action
} ];

JFFAlertView* alert_view_ =
  [ JFFAlertView alertWithTitle: @"Alert2"
                        message: @"test"
              cancelButtonTitle: @"Cancel"
              otherButtonTitles: bt_, nil ];
```

Обобщенное асинхронное программирование

1. Асинхронная операция в общем виде
2. Кеширование
3. Порядок выполнения

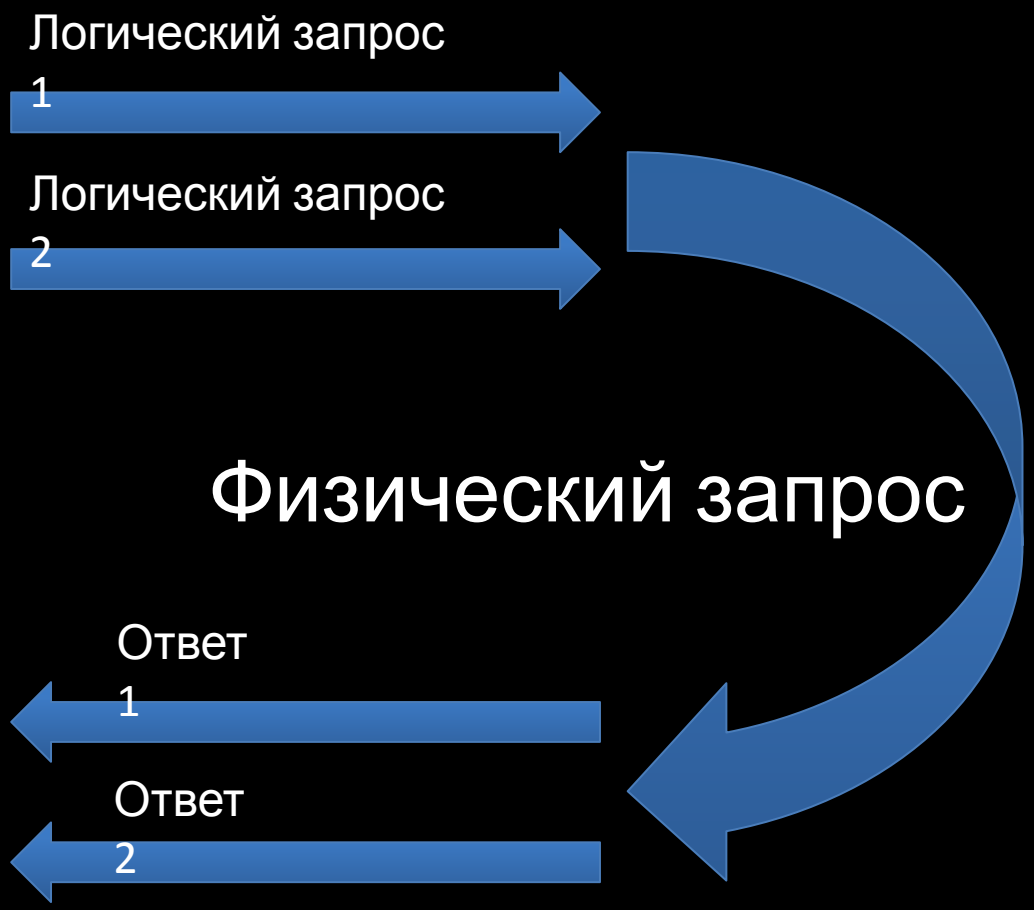
a) Дерево зависимостей, login
b) Lazy load, вычитка страниц

4. Load balancer
5. Асинхронные операции в контексте сессии
6. Асинхронные операции в UI

Асинхронная операция в общем виде

```
CancelBlock (^AsyncOperation)
        ( ProgressHandler
        , CancelHandler
        , FinishHandler ) { … };
```

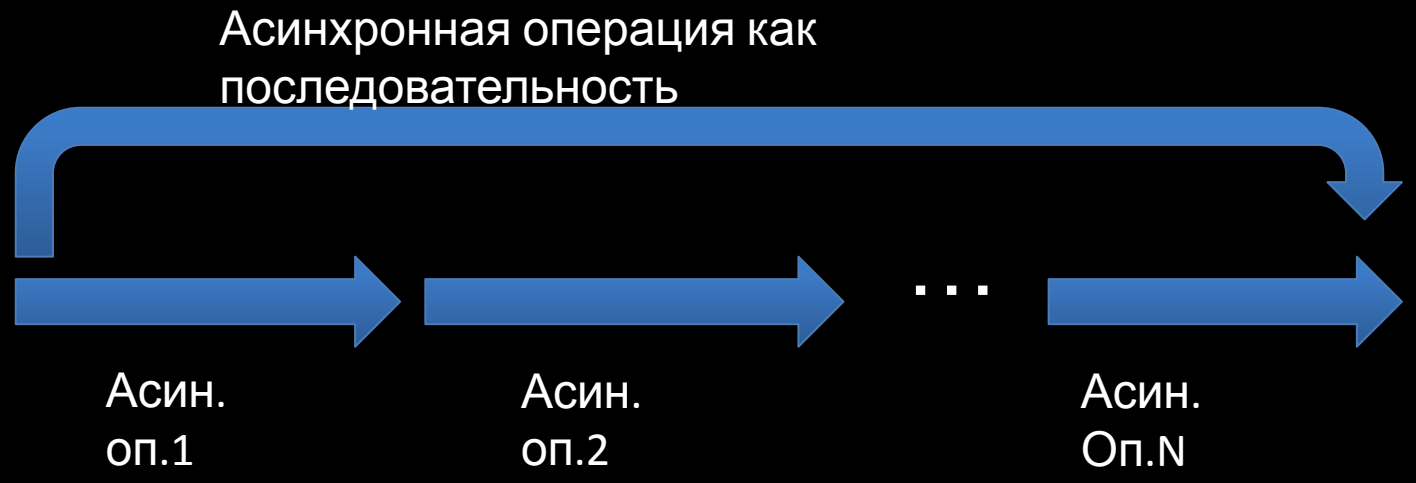
Кеширование



Кэширование, API

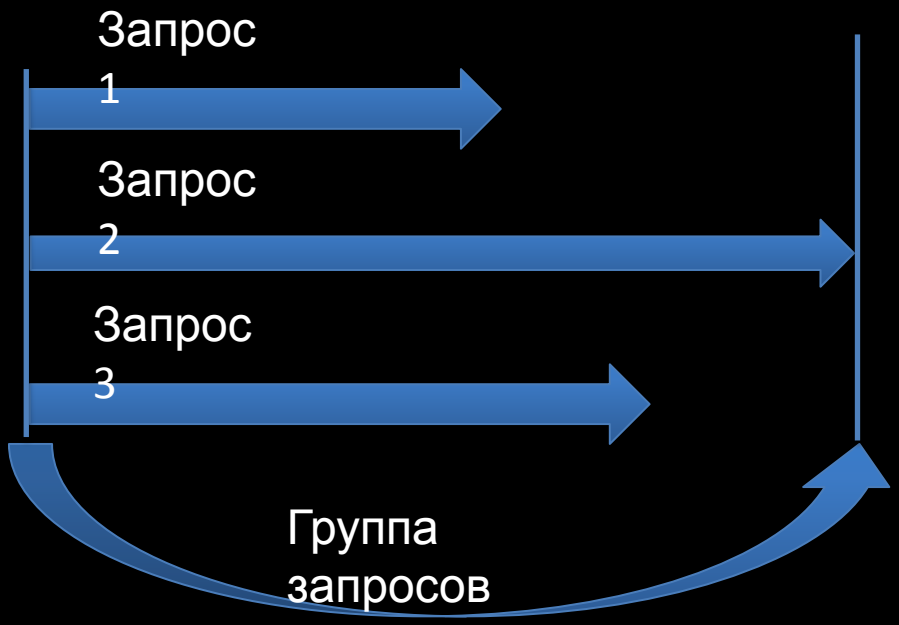
```
//физический запрос
JFFAsyncOperation data_loader_ = ...;

//кэшированный запрос
JFFAsyncOperation cached_loader_ =
[ self asyncOperationForPropertyWithName: @”image”
                          asyncOperation: data_loader_ ];
```

Порядок выполнения - последовательность



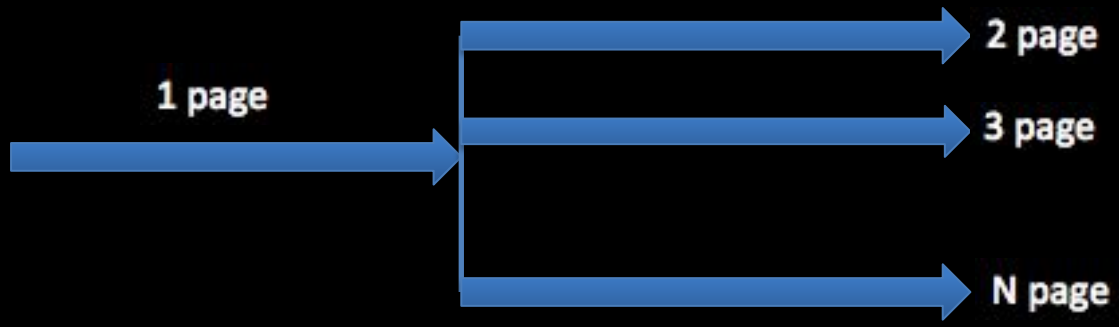

```
sequence_ = sequenceOfAsyncOperations                    (
operation1_
          , operation2_
           , nil );
```

Порядок выполнения - группа




```
group_ = groupOfAsyncOperations                    ( operation1_
                , operation2_
                 , nil );
```

Порядок выполнения – граф ленивые вычисления

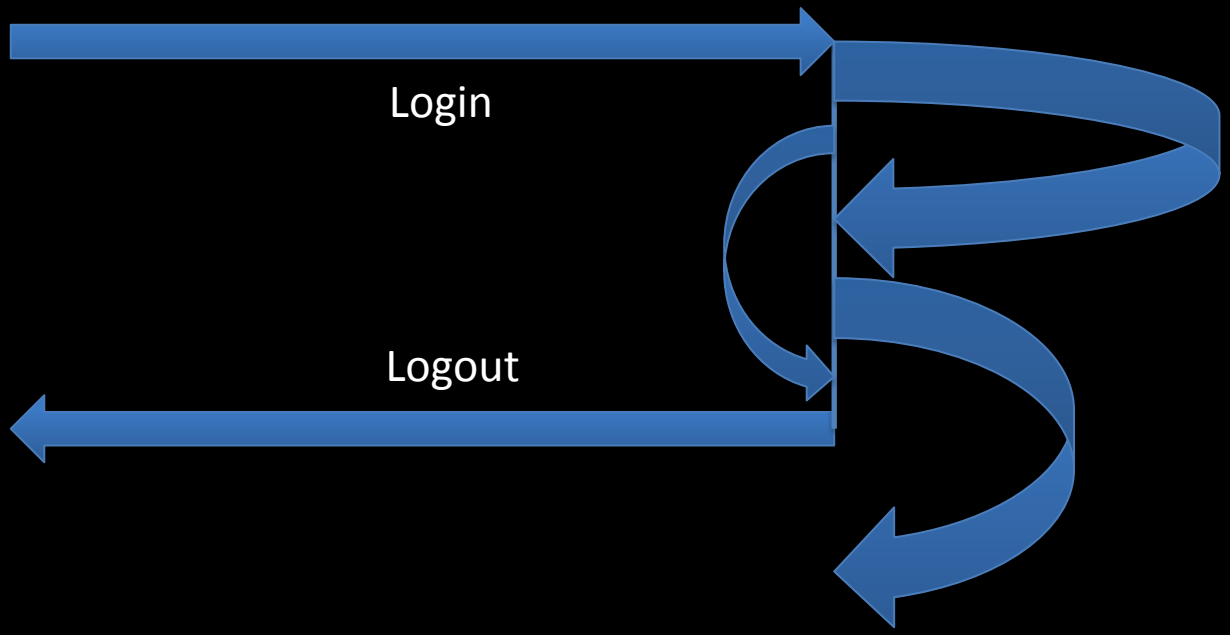



```
JFFAsyncOperation other_pages_ = ^( callbacks_ )
{
    NSArray* loaders_ = …;
    result_ = groupOfAsyncOp( loaders_ );
    return result_( callbacks_ );
};

sequenceOfAsyncOperations( first_page_
                         , other_pages_
                         , nil );
```

Load balancer


```
//имя текущего контекста
void setBalancerActiveContextName( NSString* name_ );

//сбалансированная асинхронная операция
balanced_loader_ = balancedAsyncOperation( loader_ );
```

Запросы и сессия




```
safe_loader_ = checkSessionForLoaderBlock( loader_ )
```

Легкий делегат

```
{
   [ self.clip asyncImageWithWeakDelegate: self ];
}

#pragma mark ClipDelegate

 -(void)clip:( Clip* )clip_
didLoadImage:( UIImage* )image_
       error:( NSError* )error_
{
   if ( self.clip != clip_ )
      return;

   self.imageView.image = image_;
}
```

Легкий делегат

```
JFFAsyncOperation loader_ = …;
__block id weak_delegate_ = delegate_;
[ weak_delegate_ weakAsyncOperation: loader_ ]
      ( nil, nil, ^( id image_, NSError* error_ )
{
   [ weak_delegate_ clip: self
            didLoadImage: image_
                   error: error_ ];
} );
```

Легкий делегат ARC

```
JFFAsyncOperation loader_ = …;
weak id weak_delegate_ = delegate_;
loader_( nil
       , nil
       , ^( id image_
       , NSError* error_ )
{
    [ weak_delegate_ clip: self
             didLoadImage: image_
                    error: error_ ];
} );
```

Всем спасибо !!!

Email: gorbenko.vova@gmail.com

Skype: vova.gorbenko.mac