Link : “ http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization ”

**» SFSObject/SFSArray: Tuần tự hóa lớp**

Trong bài viết này, chúng ta sẽ khám phá một tính năng nâng cao được cung cấp bởi các lớp **SFSObject** và **SFSArray** . Tính năng này cho phép trao đổi các lớp tùy chỉnh ( **[POJO](http://en.wikipedia.org/wiki/Plain_Old_Java_Object" \t "_blank)** ) giữa máy khách và máy chủ theo cách hoàn toàn minh bạch. Xin lưu ý rằng tính năng này **không được** API JavaScript, Objective-C và C++ của chúng tôi hỗ trợ.

Như chúng ta đã thấy trong phần **[giới thiệu về SFSObject/SFSArray](http://docs2x.smartfoxserver.com/DevelopmentBasics/sfsobject-sfsarray)** , bạn có thể thấy mình sao chép dữ liệu đến và từ các lớp mô hình của bạn vào (các) SFSObject khá thường xuyên trong mã Tiện ích mở rộng của bạn. Với **Tuần tự hóa lớp** , bạn sẽ có thể trực tiếp gửi và nhận các lớp mô hình của mình mà không cần bất kỳ chuyển đổi thủ công nào.

**LƯU Ý**   
Bài viết này đề cập đến các chủ đề nâng cao và yêu cầu kiến thức vững chắc về OOP (bằng cả Java và ngôn ngữ phía máy khách được lựa chọn) và làm quen với các khái niệm về **[Cơ chế tải lớp](http://www.ibm.com/developerworks/java/library/j-dyn0429/" \t "_blank)** và **[Phản chiếu kiểu](http://java.sun.com/developer/technicalArticles/ALT/Reflection/" \t "_blank)** trong JVM.

**» Ví dụ game RPG**

Để chứng minh các tính năng của **Tuần tự hóa lớp** , chúng tôi sẽ sử dụng trường hợp sử dụng thực tế và cung cấp các tệp nguồn để bạn có thể xem lại mã và thử nghiệm mã. Mã phía máy khách cho ví dụ này được cung cấp cả trong C# và ActionScript 3.

Đối với nghiên cứu này, chúng tôi đang làm việc trên một trò chơi RPG. Chúng tôi sẽ tạo một số lớp cho mô hình trò chơi và chúng tôi sẽ bắt đầu với các ký tự, sử dụng các thuộc tính sau:

* name: tên ký tự (Chuỗi)
* loại: như hiệp sĩ, pháp sư, linh mục, v.v. (Chuỗi)
* da: một đối tượng mô tả đồ họa (lớp tùy chỉnh)
* thuộc tính: một từ điển với các thuộc tính và giá trị liên quan như *sức mạnh* , *độ bền* , *sự nhanh nhẹn* , v.v. (Bản đồ)
* kho: một từ điển với các đối tượng vật lý thuộc sở hữu của nhân vật (Bản đồ)
* phép thuật: một từ điển với các phép thuật do nhân vật làm chủ (Bản đồ)

Để lập mô hình lớp **Rpg Character của chúng tôi** cho cả phía máy chủ (Java) và máy khách (C#, ActionScript 3), chúng tôi sẽ phải tuân thủ một loạt các quy ước cho việc tuần tự hóa:

* Triển khai giao diện **SerializableSFSType** . Giao diện này không có phương thức và nó chỉ hoạt động như một "điểm đánh dấu" cho các lớp có thể tuần tự hóa.
* Sử dụng **cùng một gói và tên lớp chính xác** trên cả hai mặt của ứng dụng. Nói cách khác, lớp Rpg Character phải nằm trong cùng một gói ở cả phía máy chủ và máy khách.
* Cung cấp một hàm **tạo trống** (được hệ thống sử dụng để khởi tạo lớp một cách linh hoạt).
* Cung cấp **quyền truy cập công khai vào** tất cả các trường có thể tuần tự hóa: điều này đơn giản có nghĩa là tất cả các thuộc tính mà bạn muốn được vận chuyển phải có thể truy cập trực tiếp hoặc thông qua trình thu thập/thiết lập công khai (đối với C#, hiện chỉ hỗ trợ truy cập trực tiếp). Các trường được bảo vệ, riêng tư, gói-riêng không thể được đánh số thứ tự. Các trường tĩnh cũng bị bỏ qua.
* Đánh dấu là **tạm thời** (Java) hoặc **[NonSerialized()]** (C# - ngoại trừ UWP, chưa được hỗ trợ) bất kỳ trường công khai nào không được đánh dấu theo thứ tự. Vì trong ActionScript 3, trình biến đổi tạm thời không tồn tại nên chúng tôi mô phỏng nó bằng cách sử dụng quy ước đặt tên: các trường công khai không được đánh số tự động phải bắt đầu bằng ký hiệu đô la ($). Ví dụ: $posx, $gameId, $name.
* Trong C#, dòng mã sau đây là bắt buộc để cho phép API truy cập ngữ cảnh ứng dụng:

C#

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

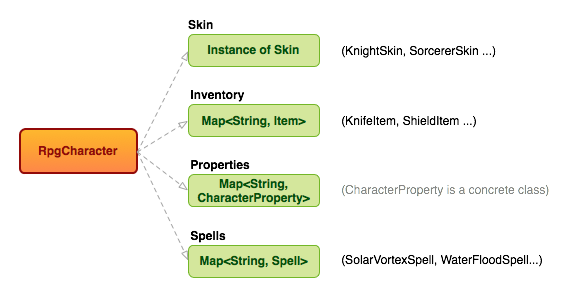
|  |  |
| --- | --- |
| 1 | Mặc địnhSFSDataSerializer.RunningAssembly = Assembly.GetExecutingAssembly(); |

Ngoài ra, chúng ta phải lưu ý rằng không phải tất cả các lớp đều có thể được vận chuyển. Ví dụ: mọi thứ tham chiếu đến tài nguyên cục bộ (tệp, ổ cắm, kết nối cơ sở dữ liệu) đều không thể tuần tự hóa. Đây là danh sách các loại có thể được sử dụng với cơ chế **tuần tự hóa Lớp** và cách chúng được dịch từ ngôn ngữ này sang ngôn ngữ khác:

|  |  |  |
| --- | --- | --- |
| **Java** | **C#** | **Tập lệnh hành động 3** |
| vô giá trị | vô giá trị | vô giá trị |
| boolean | bool | Boolean |
| byte | byte | int |
| ngắn ngủi | ngắn ngủi | int |
| int | int | int |
| Dài | Dài | Con số |
| trôi nổi | trôi nổi | Con số |
| gấp đôi | gấp đôi | Con số |
| Chuỗi | Chuỗi | chuỗi |
| Bộ sưu tập<Đối tượng>  (Danh sách, Tập hợp, Hàng đợi...) | Lập danh sách | Mảng |
| Bản đồ<Chuỗi, Đối tượng> | HashTable | Vật |

**» Các lớp ví dụ**

Sơ đồ sau minh họa rõ hơn lớp **Rpg Character chính của chúng ta** :



Có ba giao diện khác nhau cho từng trường ở trên với việc loại trừ CharacterProperty, bản thân nó là một lớp. Mỗi giao diện có một số triển khai:

* **Làn da**
  + Hiệp sĩDa
  + phù thủyda
* **Mục**
  + Con DaoMục
  + lá chắnmục
  + Búa TạHàng
  + SwordItem
  + MagicWandItem
* **Đánh vần**
  + Cái ChếtTiaChính Tả
  + TăngNgọn LửaChính Tả
  + Năng Lượng Mặt TrờiXoáy
  + NhệnBầy Đàn Chính Tả
  + NướcLũ Chính Tả

Hãy xem cách các giao diện này và các triển khai tương đối được mô hình hóa.   
Đây là giao diện Spell.java:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | **package** sfs2x.extension.test.serialization.model;    **public** **interface** Spell  {      String getId();  **void** setId(String id);    **int** getHitPoints();  **void** setHitPoints(**int** hit);    **int** getCount();  **void** cast();  } |

Và đây là một trong những triển khai, WaterFloodSpell.java:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56 | **package** sfs2x.extension.test.serialization.model;    **import** com.smartfoxserver.v2.protocol.serialization.SerializableSFSType;    **public** **class** WaterFloodSpell **implements** Spell, SerializableSFSType  {      String id;  **int** hitPoints;  **int** count = 7;    **public** WaterFloodSpell()      {          // Empty constructor      }    **public** WaterFloodSpell(String id, **int** hitPoints)      {  **this**.id = id;  **this**.hitPoints = hitPoints;      }    **public** String getId()      {  **return** id;      }  **public** **void** setId(String id)      {  **this**.id = id;      }  **public** **int** getHitPoints()      {  **return** hitPoints;      }  **public** **void** setHitPoints(**int** hitPoints)      {  **this**.hitPoints = hitPoints;      }  **public** **int** getCount()      {  **return** count;      }        @Override  **public** **void** cast()      {  **if** (count > 0)          {              System.out.println("CASTING SPELL: " + id);              count--;          }  **else**          {              System.out.println("CAN'T CAST SPELL. BUY MORE!");          }      }  } |

Như bạn có thể nhận thấy lớp thực hiện giao diện **SerializableSFSType** và cung cấp một hàm tạo trống theo yêu cầu của các quy ước đã thảo luận trước đó. Ngoài ra, tất cả các trường lớp cần được hiển thị công khai đều cung cấp quyền truy cập thông qua getters và setters.

Bây giờ chúng ta hãy xem cùng một giao diện và cách triển khai bằng ngôn ngữ phía máy khách:

C# AS3

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | **interface** Spell  {  **string** GetId();  **void** SetId(**string** id);    **int** GetHitPoints();  **void** SetHitPoints(**int** hitPoints);    **int** Count();  **void** Cast();  } |

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | **class** WaterFloodSpell : Spell, SerializableSFSType  {  **public** **string** id;  **public** **int** hitPoints;  **public** **int** count;    **public** **string** GetId() { **return** id; }  **public** **void** SetId(**string** id) { **this**.id = id; }    **public** **int** GetHitPoints() { **return** hitPoints; }  **public** **void** SetHitPoints(**int** hitPoints) { **this**.hitPoints = hitPoints; }    **public** **int** Count() { **return** count; }    **public** WaterFloodSpell() { }  **public** WaterFloodSpell(**string** id = **null**, **int** hitPoints = 70)      {  **this**.id = id;  **this**.hitPoints = hitPoints;      }    **public** **void** Cast()      {          Console.WriteLine("Casting " + id);      }  } |

Bạn có thể nhận thấy rằng cả hai phiên bản đều sử dụng cùng một gói theo yêu cầu của các quy ước **tuần tự hóa Lớp .**

Bây giờ chúng tôi có cả hai lớp ở phía máy khách và máy chủ, chúng tôi sẵn sàng gửi chúng qua mạng một cách rất thuận tiện. Đây là một đoạn ví dụ ngắn, chỉ để cung cấp cho bạn một ý tưởng.

Phía khách hàng:

C# AS3

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | ISFSObject **params** = SFSObject.NewInstance();  **params**.PutBool("IsActive", **true**);  **params**.PutInteger("TheNumber", 42);  **params**.PutClass("spell", **new** WaterFloodSpell()) // This is our custom class instance!    sfs.Send(**new** ExtensionRequest("test", **params**)); |

Server side:

Java

[**?**](http://docs2x.smartfoxserver.com/AdvancedTopics/class-serialization)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | **public** **class** TestRequestHandler **extends** BaseClientRequestHandler  {      @Override  **public** **void** handleClientRequest(User sender, ISFSObject params)      {  **boolean** isActive = params.getBool("IsActive");  **int** theNumber = params.getInt("TheNumber");          WaterFloodSpell spell = (WaterFloodSpell) params.getClass("spell");      }  } |

Trong mã nguồn được cung cấp cùng với bài viết này, chúng tôi sẽ tạo ba phiên bản khác nhau của Rpg Character và điền dữ liệu trò chơi thực tế vào chúng, dữ liệu này sau đó được trao đổi giữa máy khách và máy chủ theo cả hai hướng để minh họa quá trình tuần tự hóa.

Đầu ra của cấu trúc dữ liệu thu được bởi Tiện ích mở rộng như sau:

+------------------------------------+

RPG Character: Sigfried

+------------------------------------+

Type: knight

Inventory:

longSword -- Price: 500, Active: true

shortSword -- Price: 200, Active: true

Spells:

swarm -- HitPoints: 3, Qty: 10

Properties:

endurance -- value: 70/100

combatSkill -- value: 55/100

strength -- value: 60/100

+------------------------------------+

RPG Character: Tristan

+------------------------------------+

Type: knight

Inventory:

DragonSword -- Price: 1500, Active: true

IceKnife -- Price: 300, Active: true

IronShield -- Price: 150, Active: true

Spells:

Properties:

endurance -- value: 50/100

combatSkill -- value: 75/100

strength -- value: 80/100

+------------------------------------+

RPG Character: Hayden

+------------------------------------+

Type: sorcerer

Inventory:

DiamondKnife -- Price: 600, Active: true

Spells:

vortex -- HitPoints: 30, Qty: 10

flames -- HitPoints: 20, Qty: 5

flood -- HitPoints: 3, Qty: 7

swarm -- HitPoints: 15, Qty: 10

Properties:

endurance -- value: 50/100

combatSkill -- value: 55/100

strength -- value: 50/100

Ngoài ra còn có một cảnh báo máy chủ cụ thể được ghi lại ở phía máy chủ khi cấu trúc dữ liệu được gửi lại từ máy khách đến máy chủ:

No public setter. Serializer skipping private field: count, from class: XYZ...

Đây thực sự không phải là một vấn đề vì nó được thực hiện có mục đích: chúng tôi đã quyết định làm cho thuộc **tính đếm** không thể ghi trong phiên bản phía máy chủ của các lớp để tránh các giá trị máy khách ghi đè lên các giá trị mặc định.

**» Triển khai tiện ích mở rộng**

Tuần tự hóa lớp yêu cầu sự chú ý đặc biệt trong giai đoạn triển khai vì các lớp mô hình cần được SFS2X "nhìn thấy" ở cấp độ của Trình **tải lớp trên cùng** .

Như bạn có thể nhớ lại, mỗi Tiện ích mở rộng chạy trong Trình tải lớp riêng biệt, điều này ngụ ý rằng nếu chúng tôi triển khai các lớp mô hình bên trong tệp .jar của Tiện ích mở rộng, chúng tôi sẽ cô lập các lớp đó trong Trình tải lớp của Tiện ích mở rộng cụ thể. Từ Trình tải lớp chính, máy chủ sẽ không thể biết về các lớp này, do đó sẽ tạo ra lỗi khi cố gắng giải tuần tự hóa dữ liệu (nếu bạn có bất kỳ nghi ngờ nào về cách thức hoạt động của nó trong SFS2X, chúng tôi khuyên bạn nên tham khảo [**Tổng quan về tiện ích mở rộng**](http://docs2x.smartfoxserver.com/ExtensionsJava/overview) ).

Để tránh sự cố này, chúng tôi cần đảm bảo rằng các lớp mô hình được triển khai trong thư mục ***tiện ích mở rộng/\_\_lib\_\_/*** . Vì vậy, đối với ví dụ này, chúng tôi đã chuẩn bị hai tệp, **RpgModel.jar** nằm trong *phần mở rộng/\_\_lib\_\_/* và **RpgExtension.jar** nằm trong *phần mở rộng/rpg/.*

**» Kết luận**

Một từ cuối cùng là hiệu suất của Tuần tự hóa lớp. Nói chung, quá trình này diễn ra nhanh chóng và hiệu quả nhờ có giao thức nén SFS2X. Trong trò chơi Rpg của chúng tôi, kích thước gói kết quả chỉ là **860 byte** (so với 4323 byte không nén).

Một số nhược điểm khi sử dụng Tuần tự hóa lớp là bạn không có quyền kiểm soát trực tiếp trong việc tối ưu hóa các loại số và có thêm chi phí hoạt động do phản ánh loại thời gian chạy. Chúng tôi nghĩ rằng các nhà phát triển có thể tận dụng tốt nhất cả hai thế giới bằng cách tối ưu hóa thông báo quan trọng thông qua SFSObject (những thông báo cần hiệu quả tối đa) và sử dụng Tuần tự hóa lớp khi thích hợp.

Để hoàn thành hướng dẫn, chúng tôi khuyên bạn nên nghiên cứu các tệp nguồn được cung cấp và xem hoạt động của Tuần tự hóa lớp.

[**>> DOWNLOAD các tập tin nguồn liên quan đến hướng dẫn này <<**](http://docs2x.smartfoxserver.com/AdvancedTopics/_files/RpgExample.zip)