Tạp Chí Lập Trình

Coding is cool :-)

Hiệu ứng Shakira

Trang 10

VOL. 1
Tháng 1, 2013


226 tags
17 categories
194 comments
38,304 views
114 posts
http://www.zastavki.com/pictures/1920x1200/2009/New_Year_wallpapers_New_Zeland_New_Year_012879_.jpg

Thư ngỏ của Ban biên tập

Tình trạng thiếu hụt nguồn nhân lực công nghệ thông tin (CNTT) tại Việt Nam trong thời điểm hiện tại và trong những năm tới đã được nhắc đến rất nhiều trong thời gian gần đây. Theo Bộ TT-TT, nhu cầu nhân lực CNTT mỗi năm tăng 13%. Ước tính trong vòng 5 năm tới, các doanh nghiệp trong nước có nhu cầu tuyển dụng 411.000 người có trình độ chuyên môn về CNTT. Bên cạnh đó, các cơ quan nhà nước cũng cần khoảng 15.000 người để tham gia triển khai các dự án. Tuy nhiên, mỗi năm cả nước cũng chỉ đào tạo được khoảng 60.000 nhân lực.

Không những thiếu về số lượng, mà chất lượng của nguồn nhân lực CNTT hiện nay cũng đang ở mức yếu, và một trong những nguyên nhân được chỉ ra đó là do chương trình đào tạo của phần lớn các cơ sở đào tạo CNTT không phù hợp với nhu cầu thực tế của các doanh nghiệp, cho nên có nhiều sinh viên ra trường nhưng vẫn không đáp ứng được các yêu cầu của doanh nghiệp.

Với mong muốn thúc đẩy việc học tập, giảng dạy và thực hành phát triển phần mềm, tháng 7/2012, một nhóm các giảng viên công nghệ phần mềm có tâm huyết đã khởi xướng việc thành lập "Tạp Chí Lập Trình" với mục đích tạo ra một kênh thông tin đa chiều về phát triển phần mềm. Ở đó chúng ta có cơ hội để trao đổi với nhau về tất cả các lĩnh vực, khía cạnh có liên quan đến nghề lập trình.

Với sự ủng hộ của người đọc cùng với sự đóng góp nhiệt tình của nhiều cộng tác viên, Tạp Chí Lập Trình đã bước đầu đạt được những thành công đáng mừng. Đã có hơn 100 bài viết được đăng với tổng cộng gần 40000 lượt truy cập, gần 200 ý kiến phản hồi thuộc nhiều lĩnh vực khác nhau.

Để khích lệ tinh thần của đội ngũ các tác giả và cộng tác viên cũng như là một món quà năm mới mà Tạp Chí Lập Trình muốn gửi tới người đọc, chúng tôi đã chọn lọc và biên soạn lại các bài viết trên Tạp Chí Lập Trình để cho ra đời cuốn tạp chí mà các bạn đang cầm trên tay. Lần xuất bản này đánh dấu một cột mốc mới trong quá trình phát triển của Tạp Chí Lập Trình, và chúng tôi luôn mong muốn nhận được sự ủng hộ cũng như đóng góp của tất cả những ai mang trong mình dòng máu lập trình.

Với trình độ, kinh nghiệm và thời gian còn eo hẹp nên chúng tôi tin rằng sẽ có nhiều thiếu sót trong lần xuất bản này. Chúng tôi mong muốn sẽ nhận được nhiều các ý kiến đóng góp và phê bình của người đọc để trong tương lai chúng tôi sẽ có thể mang đến cho cộng đồng những ấn phẩm có chất lượng tốt hơn.

Xin cảm ơn và chúc các bạn một năm mới tràn đầy năng lượng, thành công và vui với nghề viết mã.

TRONG SỐ NÀY

10 cách để trở thành một lập trình viên giỏi

Đối với một lập trình viên trong thế giới công nghệ, có một thứ mà có thể kéo chúng ta ra khỏi nhà và đến nơi làm việc, đó là niềm vui và đam mê trong việc lập trình. Nhưng để khiến cho công việc thực sự vui vẻ và có thể tạo ra một niềm hứng khởi vĩnh cửu, chúng ta cần phải biết những điều căn bản để giúp trở thành một nhà lập trình viên giỏi.

Tác giả: **Ashish Arya**—Người dịch: **Phạm Thùy Dương**

"Một lập trình viên giỏi là một người luôn nhìn 2 phía khi băng qua đường 1 chiều".

~ Doug Linder~

Trong bài viết này, tôi tập hợp những mẹo mà mình đã học và thực hành trong thực tiễn để có được những kết quả tốt nhất. Không có định nghĩa thế nào là một lập trình viên giỏi, tuy nhiên tôi đang nói tới một nhóm những con người đã tạo ra những giải pháp xuất sắc cho nền công nghiệp IT và giúp cho nền công nghiệp này ngày càng phát triển.

1. Làm việc với Căn bản

Đúng với tất cả các ngành nghề và công việc, hiểu được bản chất là nhân tố quyết định cho thành công. Trừ khi một người có một nền tảng vững chắc, bằng không anh/cô ấy sẽ không bao giờ trở thành một người lập trình giỏi. Việc hiểu được bản chất giúp cho chúng ta có thể thiết kế và thực hiện những giải pháp hay với những cách tốt nhất có thể. Nếu bạn vẫn còn cảm thấy có khoảng trống trong lĩnh vực khoa học máy tính hoặc các khái niệm trong ngôn ngữ lập trình, sẽ không bao giờ là quá muộn để quay đầu và học lại những điều căn bản.

2. Hãy bắt đầu những câu hỏi (làm sao, cái gì) với tất cả nhưng đoạn code bạn viết

Một điều giúp tôi nhận ra sự khác biệt giữa những người lập trình giỏi và những người khác đó chính là sự thôi thúc muốn biết cái gì và làm cách nào điều đó lại xảy ra. Có một nhóm nhỏ những người sẽ không bao giờ rời dòng code của mình cho đến khi biết chắc chắn cái gì đang xảy ra khi dòng code đó được chạy. Đôi khi, với thời gian hạn hẹp, chúng ta không thể lúc nào cũng có được sự tự do để làm điều đó, và đôi khi ta phải bỏ dở việc đào sâu nghiên cứu và chấp nhận rằng dòng code đó có thể làm được việc. Nhưng đối với một người lập trình giỏi, chúng ta luôn có thể cố gắng hết sức để đào vấn đề càng sâu càng tốt. Và hãy tin tôi đi, việc này sẽ trở thành một thói quen và sau đó bạn làm mà không biết rằng mình đang thực hiện điều đó.

3. Bạn học được nhiều hơn nếu giúp đỡ người khác

Hầu hết chúng ta có một xu hướng phổ biến là quay lại với các diễn đàn chỉ khi chúng ta cần sự giúp đỡ. Và một lần nữa, một điều phân biệt giữa người lập trình giỏi và những người khác là họ vào các diễn đàn thường xuyên để giúp đỡ và chia sẻ. Điều này sẽ khiến họ học hỏi thêm được nhiều điều hơn là để vấn đề của mình cho người khác giải quyết. Và tin tôi đi, việc hiểu được vấn đề của người khác trong bối cảnh của họ , điều tra và cung cấp giải pháp sẽ khiến bạn học được nhiều hơn so với trước đây.

4. Viết code đơn giản, dễ hiểu nhưng phải đúng logic

Như trong hầu hết mọi khía cạnh của cuộc sống, công thức KISS (Keep It Short and Simple) cũng được áp dụng với lập trình. Hãy viết nhiều những đoạn code hợp lý và tránh sự rườm rà phức tạp. Nhiều người cố tình viết ra những đoạn mã phức tạp để chứng tỏ khả năng của mình, nhưng theo kinh của mình, tôi thấy những đoạn code đơn giản, logic luôn làm được việc, nó thường sẽ ít có vấn đề và dễ dàng mở rộng. Tôi vẫn còn nhớ một đoạn trích dẫn:

"Những đoạn mã tốt thì bản thân nó đã là một tài liệu tốt nhất. Mỗi khi bạn phải thêm vào 1 dòng chú thích (comment), hãy luôn đặt câu hỏi: Làm cách nào tôi có thể làm sáng tỏ đoạn mã này mà không cần phải chú thích thêm"

~ Steve McConnel ~

5. Dành nhiều thời gian trong việc phân tích vấn đề, bạn sẽ cần ít thời gian hơn để sửa chữa nó

Dành nhiều thời gian trong việc tìm hiểu và phân tích vấn đề và thiết kế giải pháp cho nó sẽ giúp bạn dễ dàng thực hiện được phần còn lại. Thiết kế không phải lúc nào cũng có nghĩa là bằng cách sử dụng ngôn ngữ mô hình hóa và các công bầu trời và suy nghĩ giải pháp trong tâm trí của bạn. Những người có thói quen nhấn bàn phím (để viết code) ngay thời điểm nhận được vấn đề, thường kết thúc với một cái gì đó khác hơn nhiều so với yêu cầu.

"Nếu bạn không thể hình dung ra cấu trúc tổng thể của một chương trình trong khi đang tắm, bạn chưa sẵn sàng để code nó".

~ Richard Pattis ~

6. Hãy là người đầu tiên phân tích và kiểm duyệt mã của bạn

Mặc dù có một chút khó khăn, nhưng hãy cố gắng để khám phá những đoạn mã của bạn trước khi những người khác làm việc đó, và với thời gian, bạn sẽ học được cách để viết ra những đoạn mã mà hầu như sẽ không có lỗi. Luôn luôn xem xét chặt chẽ và không thiên vị với những đoạn mã đó; Cũng không bao giờ ngần ngại để người khác xem nó. Làm việc với các lập trình viên tốt thì các phản hồi từ họ chắc chắn sẽ giúp bạn trở thành một lập trình viên tốt.

7. Không chạy theo công nghệ

Trong giai đoạn gần đây của ngành công nghiệp IT, tôi đã gặp rất nhiều người thất vọng bởi công việc của họ, thậm chí bỏ việc để tìm những cơ hội mới. Họ nói rằng họ muốn học và tìm hiểu những công nghệ mới nhất. Những gì chúng ta nghe thấy hàng ngày về "công nghệ mới" có thể hiểu là những công cụ mới, API, framework và nó được phát triển theo từng ngày để khiến cho việc lập trình dễ dàng hơn và nhanh hơn. Điều này dù sao vẫn đang rất phổ biến và sẽ tiếp tục trong thế giới công nghệ. Nhưng những gì cần phải hiểu chính là các công nghệ cốt lõi và cơ bản thì thay đổi rất ít so sánh với các framework, tool và API xung quanh nó. Cũng giống như nước biển, nếu nước ở tầng trên thường nổi sóng cuồn cuộn và chảy đi với tốc độ rất nhanh thì nước ở tầng sâu lại ít di chuyển và đó mới chính là nơi để nguồn thủy sản sinh sôi và phát triển. Vì thế, hãy cảm nhận chính mình trong phần nước sâu, và gần gũi với những công nghệ cốt lõi. Ví dụ, trong Java Enterprise, hiện đang có rất nhiều web frameworks tồn tại và còn nhiều nữa sắp được tung ra. Nhưng những khái niệm cốt lõi của mô hình Client-Server, MVC pattern, filters/servlets/JSP, resource bundling, XML parsing thì vẫn không thay đổi. Vì thế hãy dành nhiều thời gian tìm hiểu về những khái niệm này hơn là lo lắng và chạy theo những framework và tool xung quanh nó. Với nền tảng về các khái niệm cơ bản vững chắc, bạn sẽ luôn luôn thấy việc học công nghệ mới, tool hay các API sẽ là công việc khá đơn giản.

8. Giải pháp tạm thời (work-around solution) không tồn tại được lâu

Rất nhiều lập trình viên sử dụng các giải pháp tạm thời, lý do có thể là do thiếu thời gian, thiếu kinh nghiệm, thiếu sự hiểu biết của vấn đề. Nhưng theo thời gian, những giải pháp đó sẽ gây ra sự hỏng hóc của chương trình hoặc mã, nó khiến ta khó có khả năng mở rộng hoặc bảo trì, dẫn đến việc tốn nhiều thời gian và công sức sau này để sửa chữa nó. Vì thế hãy luôn luôn ghi nhớ là cần phải tìm những giải pháp khi bạn biết rõ đầu vào và đầu ra của nó. Trong nhiều trường hợp, sẽ khó tránh khỏi những việc phải sử dụng work-around, cũng giống như việc người ta hay nói: *"Tôi luôn luôn nói thật, nhưng trong một vài trường hợp tôi phải nói dối"*.

9. Đọc tài liệu

Một trong những thói quen cần thiết của các lập trình viên giỏi là họ đọc rất nhiều tài liệu. Nó có thể là các thông số kỹ thuật, JSR (Java Specification Request), tài liệu, hướng dẫn API. Đọc tài liệu sẽ giúp bạn hình dung ra nền tảng thiết yếu dựa trên đó, bạn có thể lập trình theo cách tốt nhất.

10. Bạn có thể học hỏi từ mã của người khác

Tôi cộng tác với một số lập trình viên xuất sắc, những người lúc nào cũng có những dự án java trong IDE của họ để có thể đọc hoặc tham chiếu đến nó trong công việc hàng ngày. Họ làm điều đó không chỉ để thỏa mãn nhu cầu được hiểu về các vấn đề căn bản mà còn là một cách để viết những chương trình tốt. Đọc và tham khảo những mã nguồn uy tín hoặc từ những đồng nghiệp lâu năm, sẽ giúp cho bạn tạo ra những chương trình tốt hơn.

Và điều cuối cùng, không được liệt kê ở trên: Đừng so sánh mình với những người khác

So sánh của bạn về bản thân với người khác sẽ chỉ dẫn đến những cảm xúc tiêu cực và cạnh tranh không lành mạnh. Mọi người đều có điểm mạnh và điểm yếu riêng. Quan trọng hơn là chúng ta phải hiểu bản thân và làm việc với nó. Tôi đã nhiều lần chứng kiến những người được cho là lập trình viên kiệt xuất lại làm ra những sai lầm ngớ ngẩn. Vì vậy, phân tích bản thân, lập danh sách những thứ bạn cần cải thiện và làm việc với nó. Lập trình là một niềm vui thực sự, hãy tận hưởng nó.

"Bất kỳ kẻ ngốc nào cũng có thể viết mã để một máy tính có thể hiểu được. Các lập trình viên giỏi viết mã để cho những người khác có thể hiểu được".

~ Martin Fowler~

Tự học lập trình trong 10 năm

Peter Norvig

Trong 3 ngày thậm chí bạn còn chưa đủ thời gian viết được một vài chương trình cho ra hồn chứ đừng nói đến việc học từ những thành công và thất bại. Bạn cũng không có cơ hội làm việc với những người có kinh nghiệm và hiểu xem cái gì đang xảy ra xung quanh mình.

Nói tóm lại là bạn chẳng thể nào học cho cặn kẽ được. 3 ngày chỉ đủ để bạn làm quen với phần nổi bề ngoài, đó không phải sự thấu hiểu. Và như Alexander Pope đã từng nói: "Hiểu biết nông cạn còn nguy hiểm hơn kẻ mù chữ" (*A little learning is a dangerous thing*).

Tại sao bây giờ người ta lại vội vàng đến như vậy?

Rảo bước quanh các cửa hàng sách, bạn sẽ thấy cái tựa: "Hướng dẫn lập trình Java trong 7 ngày" nằm bên cạnh một dãy dài vô tận những lời "đề nghị dạy học" đại loại như vậy về Visual Basic, Windows hay Internet... chỉ cần vài ngày hay vài giờ ngắn ngủi. Tôi đã thử thực hiện một tìm kiếm nâng cao (advanced search) tại Amazon.com: *"pubdate: after 1992 and title: days and (title: learn or title: teach yourself)"* và nhận được 248 kết quả. 78 kết quả đầu tiên là sách về máy tính (cuốn thứ 79 là *Learn Bengali in 30 days*). Thử thay "days" bằng "hours", những gì thu được cũng hoàn toàn tương tự: 253 kết quả, với 77 kết quả đầu tiên là máy tính (cuốn thứ 78 là *Teach Yourself Grammar and Style in 24 Hours*). Khi vượt ra ngoài top 200, tất cả các cuốn sách đều là sách về máy tính.

Từ đây có thể rút ra kết luận: "Hoặc là bây giờ người ta đổ xô đi tìm hiểu về máy tính, hoặc máy tính không hiểu vì một lí do hoang đường nào đó lại trở nên dễ học hơn những thứ khác". Chẳng có quyển sách nào hướng dẫn học Beethoven, Vật lý Lượng tử hay thậm chí là "chăm sóc cho chó" (Dog Grooming) chỉ trong có vài ngày. Felleisen và các đồng sự đồng tình với xu hướng này trong cuốn How to Design Programs, khi họ cho rằng "lập trình bản là rất dễ, những kẻ ngốc, thậm chí đần độn có thể học nó trong 24 giờ"

Hãy thử xem cái title: Learn C++ in Three Days gợi lên được điều gì:

Learn: Trong 3 ngày thậm chí bạn còn chưa đủ thời gian viết được một vài chương trình cho ra hồn chứ đừng nói đến việc học từ những thành công và thất bại. Bạn cũng không có cơ hội làm việc với những người có kinh nghiệm và hiểu xem cái gì đang xảy ra xung quanh mình. Nói tóm lại là bạn chẳng thể nào học cho cặn kẽ được. 3

ngày chỉ đủ để bạn làm quen với phần nổi bề ngoài, đó không phải sự thấu hiểu. Và như Alexander Pope đã từng nói: "Hiểu biết nông cạn còn nguy hiểm hơn kẻ mù chữ" (*A little learning is a dangerous thing*).

C++: Trong 3 ngày có thể bạn học xong cú pháp của C++ (nếu bạn đã thực sự biết một vài ngôn ngữ lập trình khác), nhưng bạn khó mà học được cách sử dụng chúng. Nói ngắn gọn, nếu bạn là một lập trình viên Basic, bạn có thể viết những chương trình theo **phong cách Basic** sử dụng cú pháp của Pascal, nhưng bạn sẽ chẳng hiểu được những ưu điểm hay nhược điểm của Pascal. Vậy quan điểm ở đây là gì? Alan Perlis (ND: *Giáo sư đại học Yale*), trong *Epigrams on Programming* đã từng nói: "Một ngôn ngữ mà chẳng ảnh hưởng gì đến cách bạn tư duy về lập trình, ngôn ngữ ấy chỉ là đồ bỏ đi, không đáng để học". Có thể chấp nhận được nếu bạn chỉ định học đôi chút về Pascal (hoặc Visual Basic hay JavaScript) vì bạn chỉ cần làm quen với những công cụ có sẵn để làm một việc nào đó. Nhưng đấy không phải là bạn học ngôn ngữ để lập trình, bạn chỉ học để hoàn thành nhiệm vụ cụ thể của mình mà thôi.

Trong 3 ngày: Thật đáng tiếc, điều này là không thể, như tôi sẽ chỉ ra dưới đây.

Tự học lập trình trong 10 năm

Các nhà nghiên cứu (Bloom (1985), **Bryan & Harter (1899), Hayes (1989), Simmon & Chase (1973)) đã chỉ ra rằng cần ít nhất 10 năm để đạt được sự tinh thông trong nhiều lĩnh vực,** từ đánh cờ, sáng tác âm nhạc, hội họa, bơi lội, tennis, hay thu được kết quả trong tâm lí thần kinh hay hình học topo. Điều quan trọng là bàn về phương pháp thực hành: không chỉ là việc lặp đi lặp lại đơn thuần, mà còn thử thách chính mình bằng những nhiệm vụ như vượt qua khả năng hiện tại của bản thân, cố gắng, phân tích hiệu xuất của mình trong và sau quá trình rèn luyện, và sửa chữa bất kỳ sai lầm nào. Cứ như vậy, lặp đi lặp lại. Và lịch sử đã chứng minh không thể có con đường tắt: dù cho đó là Mozart, thiên tài âm nhạc nảy nở từ năm lên 4 tuổi, cũng phải mất 13 năm để cho ra đời tác phẩm nhạc cổ điển đầu tiên. Dù cho đó là Beatles, trước khi

xuất bản #1 đầu tiên vào năm 1964, họ cũng đã phải cặm cụi trong những câu lạc bộ nhỏ tại Liverpool hay Hamburg từ năm 1957, và trong khi họ có sức hấp dẫn đại chúng từ rất sớm, thành công quan trọng đầu tiên của nhóm là *Sgt. Pepper*, album được phát hành năm 1967. Malcolm Gladwell có bài báo nghiên cứu so sánh về các sinh viên tại Học viện âm nhạc Berlin trong ba nhóm tốt, khá và trung bình và hỏi họ đã thực hành chăm chỉ như nào:

Tất cả mọi người, từ tất cả ba nhóm, bắt đầu chơi ở cùng một khoảng thời gian – tầm khoảng năm tuổi. Trong những năm đầu tiên, tất cả mọi người thực hành một lượng như nhau – khoảng hai hoặc ba giờ một tuần. Nhưng tầm tám tuổi sự khác biệt thực sự bắt đầu xuất hiện. Các sinh viên trong nhóm tốt nhất, họ bắt đầu thực hành nhiều hơn những người khác: sáu giờ một tuần lúc chín tuổi, tám tuổi là 12 giờ, 16 giờ một tuần khi 14 tuổi, và tăng lên dần, cho đến tuổi 20 họ đã được tập luyện 30 giờ một tuần. 20 tuổi, nghệ sĩ ưu tú đã có tất cả 10.000 giờ thực hành trong cuộc sống của họ. Các sinh viên chỉ đơn thuần là tốt đã đạt 8.000 giờ, và giáo viên âm nhạc trong tương lai là hơn 4.000 giờ.

Vì vậy, có thể 10.000 giờ, chứ không phải 10 năm, là con số kỳ diệu. (Henri Cartier-Bresson (1908-2004) nói: "10.000 bức ảnh đầu tiên của bạn là tệ nhất", nhưng ông đã chụp nhiều hơn một bức trong một giờ.). Samuel Johnson (1709-1784) nghĩ rằng việc này thậm chí còn lâu hơn: "Sự vượt trội ở bất cứ lĩnh vực nào cũng chỉ có thể đạt được bằng lao động cật lực trong suốt cuộc đời, bạn không thể mua nó bằng một cái giá rẻ hơn". Và Chaucer (1340-1400) phàn nàn rằng: "Cuộc đời quá ngắn ngủi, trong khi những mánh khóe thì lại quá dài để có thể học được". Hippocrates (khoảng 400BC) được biết đến với các trích đoạn "ars longa, vita brevis", là một phần của chích đoạn dài hơn "Ars longa, vita brevis, occasio praeceps, periculosum experimentum, iudicium difficile", với nghĩa tạm dịch là "Cuộc đời là ngắn ngủi, với cơ hội phù du, đầy dãy các kỹ năng, với các thử nghiệm nguy hiểm, và phán quyết khó khăn". Mặc dù trong tiếng Latin, ars có thể có nghĩa là nghệ thuật, hoặc mánh khóe, nếu so từ gốc tiếng Hy Lạp "techne" thì chỉ có thể có nghĩa là "kỹ năng", không phải "nghệ thuật".

Và nếu bạn muốn trở thành lập trình viên.

Đây là công thức cho những thành công của tôi trong lập trình:

Hãy yêu thích nó, bạn làm vì bạn cảm thấy vui vẻ và hào hứng. Hãy chắc chắn rằng bạn luôn như thế trong 10 năm...

Hãy trao đổi với những lập trình viên khác, đọc chương trình của họ viết. Điều này còn quan trọng hơn bất kì quyển sách hay khóa đào tạo nào.

Lập trình: Cách học tốt nhất là học đi đôi với hành (learning by doing). Nói cho rõ ràng hơn thì: "Tầm cao nhất của một lĩnh vực nào đó không thể có được chỉ thông qua sự bổ sung về mặt kinh nghiệm. Nhưng kể cả khi đã có rất nhiều kinh nghiệm, nếu cố gắng và nỗ lực, bạn vẫn có thể tiến xa hơn" và "để học tập một cách hiệu quả nhất, bạn cần phải xác định rõ khả năng hiện tại của mình, thu thập kiến thức từ người khác, và tự tìm lấy những cơ hội để học theo và sửa đổi sai lầm của chính mình". Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life là một cuốn sách rất hữu ích cho quan điểm này.

Nếu bạn muốn, hãy giành 4 năm ở đại học (và nhiều hơn ở trường phổ thông). Bạn sẽ có cơ hội làm những công việc yêu cầu khả năng và hiểu biết chuyên sâu hơn trong một vài lĩnh vực, nhưng nếu bạn không thích trường học, bạn có thể (với một số sự nỗi lực và cống hiến) có được kinh nghiệm tương tự của riêng bạn hoặc trong công việc . Trong bất kì trường hợp nào, chỉ đọc sách thôi là chưa đủ. "Bút vẽ và mực màu không thể biến bạn trở thành họa sĩ, cũng như những bài giảng về khoa học máy tính trên lớp không thể giúp bạn trở thành chuyên gia lập trình", Eric Raymond, tác giả của *The New Hacker Dictionary* đã nói như vậy. Một trong những lập trình viên giỏi nhất mà tôi đã từng thuê chưa từng có bằng đại học, nhưng anh ta đã tham gia viết những phần mềm tuyệt vời (ví dụ), có news group của riêng mình và anh ta thậm chí còn giàu hơn cả tôi.

Hãy tham gia vào các projects với những lập trình viên khác. Bạn có thể là best programmer trong một projects, nhưng cũng có thể là worst. Nếu là best, hãy thử khả năng làm leader của mình. Còn ngược lại, hãy học hỏi xem người giỏi nhất làm gì, và học xem họ không làm gì (vì họ muốn bạn làm những việc đó).

Trong một project, hãy *bắt tay làm việc sau những lập trình viên khác*, khi bạn đã hiểu chương trình của họ. Hãy tìm xem những gì cần thiết để hiểu và chỉnh sửa khi người viết ra chương trình đó không có mặt ở đây. Thử nghĩ xem làm thế nào để thiết kế những chương trình giúp cho những người sau này có thể dễ dàng hơn trong việc bảo trì nó.

Hãy học ít nhất khoảng nửa tá ngôn ngữ lập trình, bao gồm:

Một ngôn ngữ hỗ trợ việc tạo các lớp trừu tượng (Java hoặc C++)

Một ngôn ngữ lập trình hàm (Lisp hoặc ML)

Một ngôn ngữ cú pháp (Lisp)

Một ngôn ngữ hỗ trợ khai báo định danh (Prolog hoặc C++ templates)

Một ngôn ngữ hỗ trợ coroutine (Icon hay Scheme)

Một ngôn ngữ hỗ trợ song song (Sisal)

Hãy nhớ rằng "máy tính" là một phần của "khoa học máy tính". Nên nắm rõ thời gian để máy tính thực hiện một chỉ thị, lấy một từ trong bộ nhớ (có hay không có cache), đọc những từ liên tiếp trong ổ đĩa, hay tìm kiếm một vùng nhớ cụ thể.

Thời gian cần thiết để thực hiện các phép toán với một PC tốc độ 1GHz:

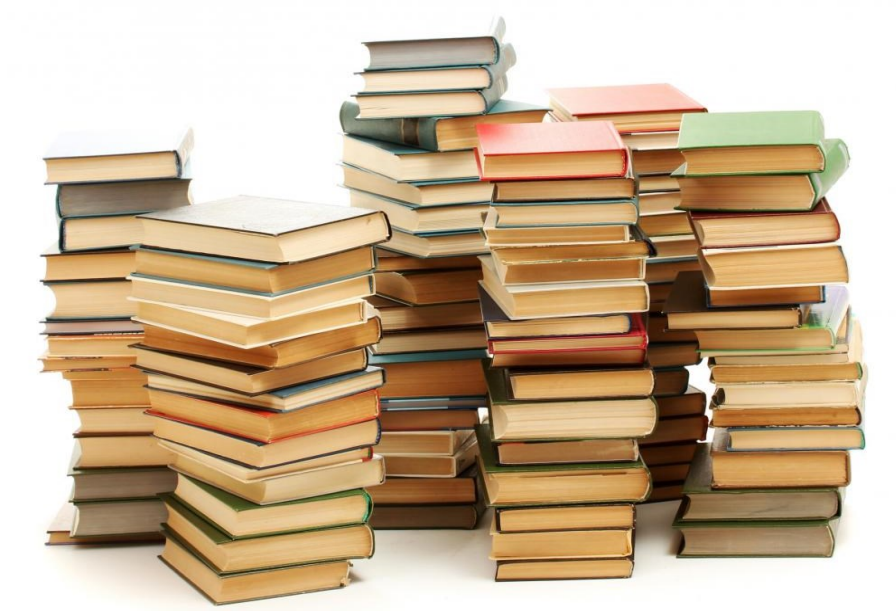
Hãy cố gắng tuân theo những qui định tiêu chuẩn của ngôn ngữ. Có thể đó là của ANSI C++, hoặc đơn giản đó là của công ty bạn. Bạn cũng nên tìm hiểu xem người ta thích gì ở ngôn ngữ đó, họ cảm thấy thế nào, hay vì sao họ thích nó.

Hãy sẵn sàng từ bỏ những tiêu chuẩn của ngôn ngữ càng nhanh càng tốt.

Với những điều tôi đã nói ở trên, bạn có thể băn khoăn rằng bạn sẽ đạt đến mức nào nếu chỉ đọc sách? Trước khi con trai cả của mình ra đời, tôi đã đọc tất cả những cuốn sách *How To* (làm thế nào), và vẫn cảm thấy vô cùng mù mờ, rối rắm. 30 tháng sau, khi sinh đứa thứ hai, phải chăng tôi đã quay lại chúng với cái nhìn hoàn toàn mới mẻ? Không, tôi chỉ dựa vào kinh nghiệm của chính mình.

Chúng có ích và khiến tôi vững tin hơn nhiều so với việc đọc hàng ngàn trang sách của các bậc chuyên gia.

Fred Brooks, trong bài luận nổi tiếng *No Silver Bullets* đã làm rõ ba bước để tìm một nhà thiết kế tốt:



Hãy giải thích một cách có hệ thống cho câu hỏi: "Thế nào là một nhà thiết kế đỉnh?" càng sớm càng tốt.

Hãy giao triển vọng và tương lai phát triển của công ty cho những người thông thái, có nhiều kinh nghiệm và tìm cách giữ chân họ cẩn thận.

Trao cơ hội cho những nhà thiết kế đang trong giai đoạn trưởng thành để họ có thể tiếp xúc, học hỏi, nâng cao trình độ.

Điều này coi như giả định là một số người có đầy đủ những phẩm chất cần thiết để trở thành một nhà thiết kế lớn, và công việc đủ để dụ họ theo. Alan Perlis đã từng nói: "Mọi người đều có thể được dạy điêu khắc, Michealangelo thì không. Và ông đã tự trở thành nhà điêu khắc vĩ đại. Những lập trình viên siêu hạng cũng nên thế". Perlis đã từng nói rằng những người vĩ đại sẵn có một nội lực vượt qua cả sự đào tạo. Nhưng nội lực đó đến từ đâu? Bẩm sinh ư? Hay chúng được phát triển thông qua sự tích cực? Như Auguste Gusteau (đầu bếp huyền thoại của Ratatouille) đã nói: "bất cứ ai cũng có thể nấu ăn, nhưng chỉ những người không biết sợ mới trở nên vĩ đại". Tôi nghĩ về điều này nhiều đến mức cứ như là sẵn sàng cống hiến phần lớn thời gian trong đời của ai đó để luận bàn. Nhưng *không hề sợ hãi* có lẽ là cách để tóm lại tất cả những điều đó. Hoặc, như nhà phê bình Gusteau, Anton Ego, nói: "Không phải tất cả mọi người đều có thể trở thành một nghệ sĩ lớn, nhưng một nghệ sĩ lớn có thể đến từ bất cứ nơi nào."

Vì vậy, nếu mua quyển sách Java/ Ruby/Javascript/ PHP nói trên, bạn có thể sẽ thu được một vài điều có ích. Nhưng nó không thể thay đổi cuộc đời của bạn, hoặc đưa bạn đến sự thành thạo hay tinh thông, chỉ trong 24 giờ, vài ngày hay thậm chí là vài tháng.

Người dịch: **Nguyễn Ngọc Tú**

Tôi nên lựa chọn ngôn ngữ lập trình nào để bắt đầu?

Hãy nhìn vào bạn bè mình. Khi hỏi: "Tôi nên sử dụng hệ điều hành nào đây, Windows, Unix hay Mac? tôi thường đáp rằng: "Hãy nhìn vào bạn của mình". Lợi ích thu được từ việc học hỏi bạn bè sẽ bù đắp cho những khác biệt căn bản giữa các hệ điều hành hay ngôn ngữ lập trình. Thêm nữa, hãy nhìn những người bạn "sắp quen": đó là tập hợp các programmer mà bạn sẽ song hành cùng họ nếu bạn vẫn tiếp tục hành trình. Ngôn ngữ bạn chọn có một cộng đồng phát triển rộng lớn hay chỉ một vài người tham gia? Có những tài liệu, websites hay forums nào mà bạn sẽ nhận được lời giải đáp cho thắc mắc của mình? Bạn có thích những người ở đó không?

Hãy làm thật đơn giản. Những ngôn ngữ như C++ và Java được thiết kế cho các ứng dụng chuyên nghiệp thực hiển bởi đội ngũ đông đảo các lập trình viên giàu kinh nghiệm, những người vốn rất quan tâm đến hiệu suất những dòng mã của họ. Vì thế, chúng thường bao gồm nhiều thành phần rất phức tạp. Nếu mới bắt đầu, bạn không cần đến sự phức tạp. Bạn hãy học những ngôn ngữ có cấu trúc đơn giản, dễ học.

Chơi. Bạn thích học piano theo cách nào hơn: kiểu tương tác thông thường, tức là bạn nghe mỗi nốt nhạc ngay khi nhấn một phím đàn hay cách thức theo "lô", tức bạn chỉ nghe các nốt sau khi đã hoàn thành cả nhạc phẩm? Rõ ràng, cách thứ nhất giúp bạn dễ dàng học hơn phải không? Lập trình cũng như vậy. Hãy chọn cho mình một kiểu tương tác và sử dụng nó

Theo tiêu chuẩn của riêng mình, tôi khuyên bạn nên bắt đầu với Python hoặc Scheme. Nhưng mỗi người có những hoàn cảnh khác nhau, và có thể có những lựa chọn tốt hơn. Nếu tuổi của bạn mới chỉ là số có một chữ số, theo tôi bạn nên chọn Alice hoặc Squeak (người lớn cũng có thể thích chúng). Tuy nhiên, đó không phải là điều quan trọng. Quan trọng là, hãy chọn đi và bắt đầu ngay lập tức.

Hiệu ứng Shakira

Nguyễn Khắc Nhật

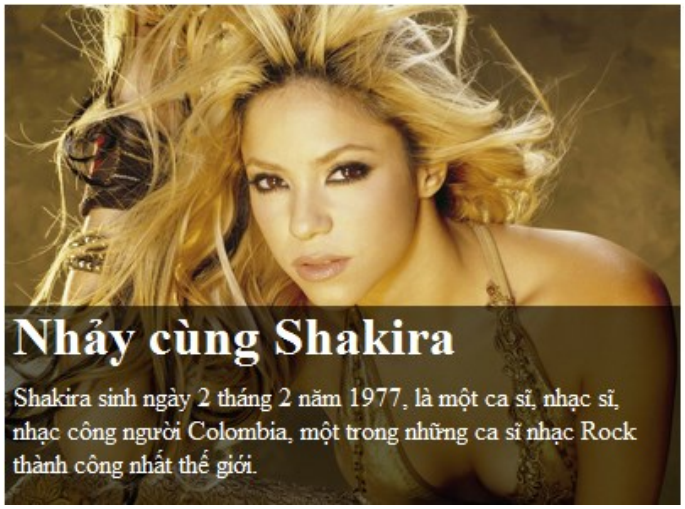
Hiệu ứng Shakira?

Chào các bạn, "Coding is cool" là câu khẩu hiệu của Tạp Chí Lập Trình, tuy nhiên, tôi biết rằng việc Lập trình có đôi khi là nhàm chán, nhất là đối với những người vẫn còn khó khăn trong việc tiếp cận những dòng code mới. Nhằm tạo một chút vui vẻ và cũng để mang lại cho các bạn chút hứng thú để "play" với những dòng code của mình, hôm nay tôi giới thiệu tới các bạn cách dùng CSS để tạo một hiệu ứng có tên là "Hiệu ứng Shakira".

Chắc chắn bạn đã gặp hiệu ứng này nhiều lần nhưng chỉ là bạn không biết đến cái tên mà tôi dùng để gọi nó mà thôi, (thú thực là cách đây khoảng 30 phút về trước nếu có ai hỏi tôi về cái hiệu ứng có tên như vậy thì tôi cũng chịu, đơn giản là vì tôi vừa mới nghĩ ra :)). Hiệu ứng này được mô tả như sau: Chúng ta có một bức ảnh, trên bức ảnh đó có một tiêu đề của bài viết, khi ta di chuột lên trên bức ảnh thì một đoạn tóm tắt nội dung bài viết sẽ được hiện ra. Đơn giản là thế.

Tóm lại, ban đầu ta có cái này:

Khi di chuột lên bức ảnh ta nhận được kết quả sau:



HTML Code

Chúng ta bắt đầu nhé, mã HTML của chúng ta là như sau:

```
<div class="shakira-box">
   <img src="shakira-1.jpg"/>
   <div class="shakira-content">
      <h2 class="shakira-header">Nhảy cùng
Shakira</h2>
      <p class="shakira-body">Shakira sinh
ngày 2 tháng 2 năm 1977, là một ca sĩ, nhạc
sĩ, nhạc công người Colombia, một trong những
ca sĩ nhạc Rock thành công nhất thế giới.</p>
   </div>
</div>
```

Nhìn vào đó bạn sẽ dễ dàng nhận thấy rằng tôi có một thẻ div ngoài cùng với class là "shakira-box", đây chính là thẻ div chứa tất cả các thành phần con bên trong như là: một bức ảnh (thẻ <img>), một thẻ với nội dung bên trong là một header (thẻ <h2>) và một đoạn tóm tắt bài viết (thẻ <p>). Để dễ bề điều khiển các thuộc tính của thẻ thì tôi đã thêm vào đó các class khác nhau. Bây giờ đến lượt thêm các dòng code css.

CSS Code

```
.shakira-box{
   width:400px;
   height:300px;
   overflow:hidden;
}
```

Đối với thẻ ngoài cùng thì tôi sẽ gắn cho nó một chiều cao, chiều rộng nhất định; thêm nữa tôi gắn giá trị 'hidden' cho thuộc tính 'overflow', điều này khiến cho cái 'box' này chỉ hiển thị những nội dung nằm trong khoảng không gian của nó, những phần nằm ở ngoài thì sẽ bị ẩn đi.

```
.shakira-box img{
   width:100%;
   height:100%;
}
```

Đoạn code ở trên thì quá đơn giản phải không nào, chắc không cần phải giải thích gì thêm.

```
.shakira-box .shakira-content{
   width:100%;
   height:120px;
   top:-40px;
   position:relative;
   background-color:rgba(0,0,0,0.6);
   color:#FFF;
}
```

Đoạn code trên chính là nơi quan trọng nhất của hiệu ứng này, ta sẽ đi phân tích từng dòng một. Thuộc tính 'width' quy định rằng thẻ này sẽ có độ rộng bằng chính thẻ bao ở ngoài (thẻ có class là shakira-box); thuộc tính height quy định chiều cao của thẻ*content* (thẻ có class là shakira-content), trong trường hợp này thì thẻ *content* của tôi có chiều cao là 120px. Thuộc tính *position* có giá trị là '*relative*', thuộc tính '*top*' có giá trị là'*-40px*', điều này làm cho thẻ của chúng ta được dịch lên phía trên một đoạn là *40px*so với vị trí bình thường của nó, tôi làm như vậy với mục đích là để phần header của chúng ta nằm gọn bên trong '*box*' và sẽ được nhìn thấy, bạn nên xem để ý đoạn code dưới đây và bạn sẽ thấy là phần header sẽ có chiều cao là *40px*:

```
.shakira-box .shakira-header{
   height:40px;
   margin:0;
   font-size:24pt;
   padding: 0 5px;
}
```

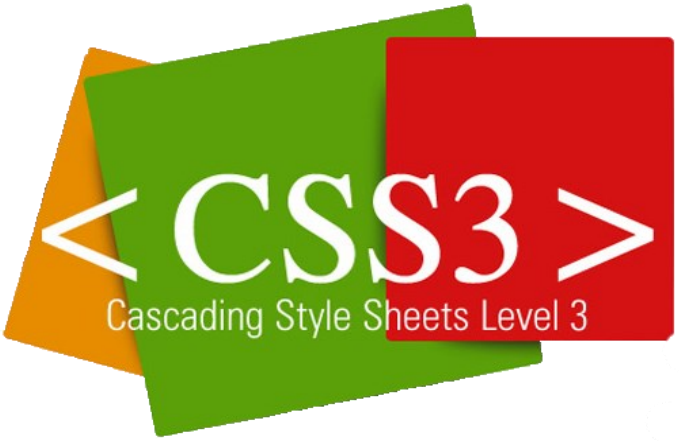
Song song với việc quy định vị trí cho thẻ content, bây giờ chúng ta sẽ tạo một bộ chọn css khác để nếu khi ta di chuột lên trên 'box' thì vị trí của thẻ content sẽ thay đổi:

```
.shakira-box:hover .shakira-content{
   top: -120px;
}
```

Bây giờ bạn thử dừng lại và quan sát kết quả của mình nào. Tất cả mọi thứ đã hoạt động tốt, trừ việc chúng ta vẫn chưa có được hiệu ứng di chuyển chậm lên phía trên. Để làm được điều này thì chúng ta nên sử dụng thuộc tính *transition* mà css3 cung cấp cho chúng ta, bộ chọn *.shakira-box .shakira-content* bây giờ được sửa lại :

```
.shakira-box .shakira-content{
   width:100%;
   height:120px;
   top:-40px;
   position:relative;
   background-color:rgba(0,0,0,0.6);
   color:#FFF;
   transition:top 0.5s;
   -moz-transition:top 0.5s;
   -webkit-transition:top 0.5s;
   -o-transition:top 0.5s;
   -ms-transition:top 0.5s;
}
```

Các bạn cũng đã biết rằng css3 vẫn đang trong quá trình phát triển, do đó có nhiều trình duyệt chưa hỗ trợ hết các thuộc tính *transition*, việc ta thêm các tiền tố như *-moz-*, *-webkit-*, *-o-* là cần thiết để thuộc tính này hoạt động tốt trên các loại trình duyệt khác nhau (ngoại trừ anh chàng *Internet Explorer* từ trước cho đến phiên bản 9). Thuộc tính *transition* trong trường hợp này quy định rằng thuộc tính top sẽ không thay đổi ngay tức thì khi chúng ta hover chuột lên '*box*', mà nó sẽ thay đổi dần dần trong vòng 0.5 giây.



Lời kết

Để có được những hiệu ứng như trên thì ngoài việc sử dụng CSS chúng ta có thể dùng javascript, hoặc đơn giản hơn là dùng jQuery, nhưng nếu so về số lượng dòng code thì chắc chắn việc dùng css sẽ ít hơn nhiều, còn nếu so về performance thì việc dùng css cũng tối ưu hơn (sao thế nhỉ?), chúng ta cần phải cảm ơn css3 vì đã cung cấp nhiều thuộc tính mới và rất tiện lợi để chúng ta sử dụng.

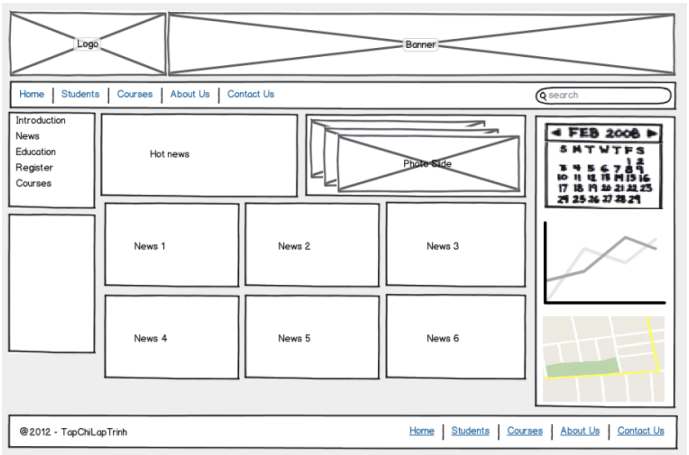
W3C đưa ra bản dùng thử đầu tiên của CSS3 vào tháng 2 năm 2008, sau hơn một thập kỷ kể từ khi CSS2 ra đời.

Thời điểm phát hành của HTML5 được ấn định vào năm 2022, còn CSS3 thì phải mất hơn 10 năm mới có thể trở thành một tiêu chuẩn được khuyên dùng. Nhưng ngày phát hành các tiêu chuẩn này không quan trọng bằng việc các trình duyệt nhanh chóng hỗ trợ nó.

Tạo Layout đơn giản và hiệu quả với DIV và CSS

~ Nguyễn Việt Khoa~

Việc sử dụng DIV kết hợp với CSS để làm layout (bố cục) cho một trang web đã trở lên phổ biến và đạt hiệu quả cao. Chẳng hạn bạn cần thiết kế một trang có giao diện được thiết kế với layout như sau:



Với layout như trên bạn hoàn toàn bạn có thể sử dụng table (bảng) cho việc thiết kế, tuy nhiên table lại bộc lộ khá nhiều nhược điểm khi sử dụng làm layout cho một trang web có cấu trúc như trên. Các nhược điểm có thể gặp phải với table với những dạng layout này là chậm, khó tùy chỉnh và khó kết hợp với CSS\Javascript để tạo lên sự linh hoạt cho trang web.

Sau đây tôi sẽ hướng dẫn các bạn các bước căn bản nhất để có thể tạo được các layout với DIV và CSS. Trước hết bạn cần tạo mới một trang HTML với cấu trúc chuẩn (gồm đầy đủ các phần html, head, title, body). Tiếp theo bạn code đoạn mã HTML sau vào phần body của trang.

```
<div id="main">
   <div id="head">
   </div>
   <div id="head-link">
   </div>
   <div id="left">
   </div>
   <div id="content">
   </div>
   <div id="right">
   </div>
   <div id="footer">  </div>
</div>
```

Các cặp thẻ trong đoạn mã trên giúp bạn tạo ra các phân vùng khác nhau trên trang HTML của mình, mỗi phân vùng được đánh ID riêng biệt (các ID này sẽ dùng để định nghĩa CSS), chi tiết như sau:

1. **main**: Phân vùng chứa toàn bộ nội dung của trang.
2. **head**: Phân vùng chứa nội dung đầu tiên của trang (theo VD trên đó là phần logo + banner)
3. **head-link**: Phân vùng chứa các liên kết đầu trang (ở VD trên: Home, Students, Course, v.v.)
4. **left**: Phân vùng chứa các nội dung bên trái (ở VD trên là các liên kết trái)
5. **content**: Phân vùng chứa các nội dung chính của trang (ở VD trên là các phân vùng thông tin như: Hot news, Photo Slide, News 1, v.v.)
6. **right**: Phân vùng chứa các thông tin bên phải trang (ở VD trên là lịch, biểu đồ, bản đồ)
7. **footer**: Phân vùng chứa các thông tin cuối trang (ở VD trên là thông tin về chủ sở hữu và các liên kết)

Bây giờ là đến công đoạn code các bộ chọn CSS (selector) để phân vùng rõ ràng cho các DIV đã tạo ở trên:

1. Tạo bộ chọn **body** để cố định phông chữ cho cả trang và bộ chọn **#main** để cố định độ rộng vùng thông tin và trang sẽ chiếm và căn chỉnh vùng này nằm giữa màn hình.

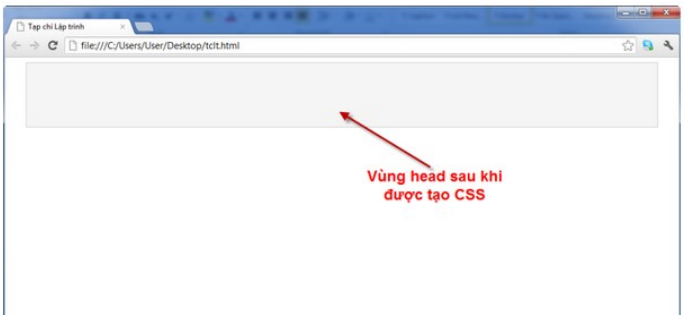
```
body{
   font-family: Arial, Tahoma;
   font-size: 12px;
}

#main{
   width: 1000px;
   padding: 0;
   margin-left: auto;
   margin-right: auto;
}
```

2. Tạo bộ chọn #head xác định chiều cao, màu nền, đường viền và khoảng cách so với các vùng khác (khoảng cách so với vùng bên dưới – margin-bottom).

```
#head{
   height: 100px;
   background-color: #F5F5F5;
   border: 1px solid #CDCDCD;
   margin-bottom:5px;
}
```

3. Kiểm tra lại trang trên trình duyệt bạn sẽ có kết quả sau (phần div với id là head đã được xác định):



4. Tiếp tục với định nghĩa bộ chọn **#head-link**, để xác định vùng đặt các liên kế đầu trang:

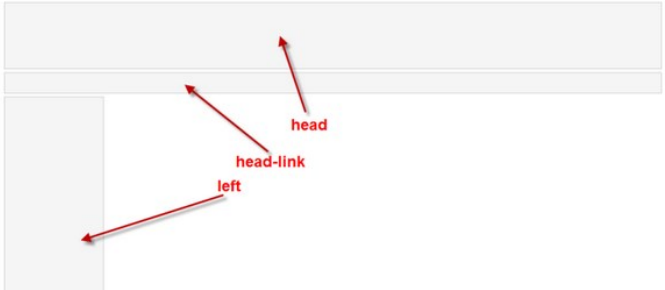
```
#head-link{
   height: 30px;
   line-height: 30px;
   padding-left: 10px;
   padding-right: 10px;
   border: 1px solid #CDCDCD;
   background-color: #F5F5F5;
   margin-bottom:5px;
   clear: both;
}
```

5. Quan sát lại trang trên trình duyệt, bạn sẽ thấy vùng liên kết đầu trang hiển thị ngay dưới phần head.

6. Định nghĩa tiếp bộ chọn #left để xác định vùng cho nội dung bên trái, với độ rộng là 200px, chiều cao tối thiểu là 400px và đặc biệt chú ý tới mô tả **float: left;** mô tả này làm cho vùng bên trái (DIV với ID là left) sẽ dạt sang bên trái nhường chỗ cho các phân vùng khác: content và right).

```
#left{
   width: 150px;
   min-height: 400px;
   border: 1px solid #CDCDCD;
   float:left;
   background-color: #004C00;
   margin-bottom: 5px;
}
```

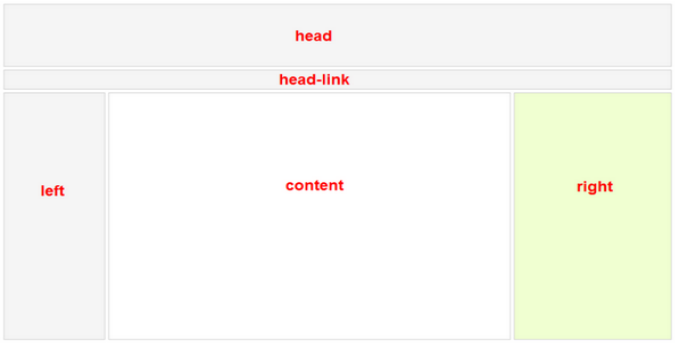
7. Quan sát lại sự thay đổi về các vùng của trang trên trình duyệt:



8. Bổ sung tiếp các selector cho vùng giữa và vùng phải lần lượt là **#content** và **#right**:

```
#content{
   width: 600px;
   min-height: 400px;
   border: 1px solid #CDCDCD;
   float:left;
   margin-left: 5px;
   margin-right: 5px;
   margin-bottom: 5px;
}
#right{
   width: 234px;
   min-height: 400px;
   border: 1px solid #CDCDCD;
   float:right;
   margin-bottom: 5px;
}
```

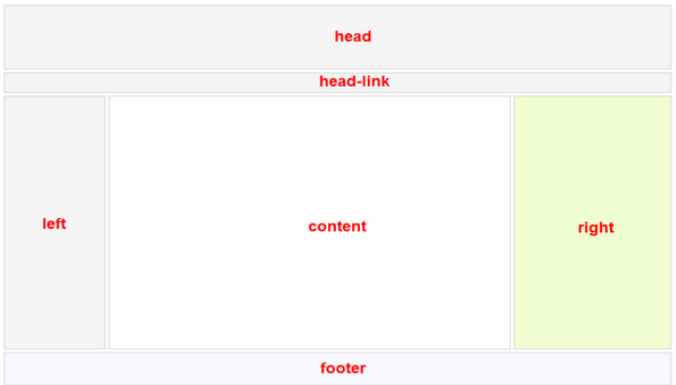
9. Quan sát lại trang:



10. Cuối cùng, bạn định nghĩa bộ chọn cho phân vùng cuối trang với id là **#footer:**

```
#footer{
   height: 50px;
   clear: both;
   border: 1px solid #CDCDCD;
   background-color: #F8F8FF;
}
```

11. Quan sát lại trang trên trình duyệt bạn sẽ thấy các phân vùng cần thiết đã vào đúng vị trí cần đặt:



Như vậy qua các bước ở trên chúng ta đã thử nghiệm với các phương pháp cơ bản để dàn trang sử dụng thẻ Div có sự kết hợp với CSS. Điều quan trọng nhất là bạn biết cách áp dụng những kỹ thuật cơ bản này để tạo ra các trang web có layout mà bạn\khách hàng của bạn mong muốn. Bây giờ bạn tiếp tục bổ sung thông tin vào các phân vùng và định nghĩa thêm các bộ chọn cần thiết để có thể có một trang HTML giống như mẫu thiết kế ở đầu bài viết.

Chúc bạn thành thạo DIV và CSS để có được những trang web vừa mắt người dùng!

Tùy biến List đẹp hơn với CSS

Nguyễn Việt Khoa

Bạn đã được học HTML, bạn hiểu rất rõ về các thẻ tạo List (danh sách) trong HTML. List cũng cung cấp cho bạn nhiều công dụng trong thiết kế web, chẳng hạn bạn có thể biến chúng thành các "Trình đơn" (Menu – ngang, dọc), dùng chúng để liệt kê một sê-ri ảnh, mẩu tin, danh sách các hạng mục, v.v... Bạn có thể thay đổi danh sách với các kiểu khác nhau như bullet tròn, vuông, v.v.. tuy nhiên tất cả những kiểu định dạng sẵn có này đều không đáp ứng được nhu cầu về mỹ thuật. Với CSS bạn có thể tùy biến cho List của mình trở lên sinh động hơn. Bài viết này tôi sẽ giới thiệu 02 cách thức đơn giản để bạn có thể làm được điều này.

Trước hết bạn chuẩn bị một trang HTML chứa đoạn code sau:

```
<h4>Tạp chí Lập trình</h4>
<ul>
   <li>Tin mới</li>
   <li>Array trong Javascript</li>
   <li>Hiệu ứng Shakira</li>
   <li>Tìm hiểu Cấu trúc dữ liệu #1: "Chết vì thiếu hiểu biết"</li>
</ul>
```

Trang này của chúng ta sẽ có kết quả như thế nào chắc bạn đã tưởng tượng ra rồi nhỉ?

Tạp chí Lập trình

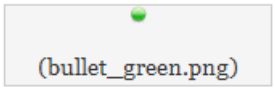
- Tin mới
- Array trong Javascript
- Hiệu ứng Shakira
- Tìm hiểu Cấu trúc dữ liệu #1: "Chết vì thiếu hiểu biết"

Giờ chúng ta sẽ bắt tay vào để làm cho danh sách này đẹp hơn.

Cách thứ nhất:

Sử dụng thuộc tính list-style-image của CSS để thay đổi bullet cho danh sách.

Để tiến hành cách này bạn tìm một icon mà mình thấy phù hợp nhất cho danh sách kể trên, còn đây là icon tôi chọn:



Icon này tôi đặt vào thư mục icons, là thư mục con của thư mục chứa trang HTML kể trên. Bây giờ ta sẽ định nghĩa một selector (bộ định kiểu) cho danh sách của mình, đơn giản như sau:

```
ul{
   list-style-image: url('icons/
bullet_green.png');
}
```

Kết quả như ảnh minh họa dưới đây, chúng ta sẽ có một danh sách với bullet sinh động hơn so với ban đầu.

Tạp chí Lập trình

- Tin mới
- Array trong Javascript
- Hiệu ứng Shakira
- Tìm hiểu Cấu trúc dữ liệu #1: "Chết vì thiếu hiểu biết"

Vậy đấy, chỉ với một chút cải tiến ta đã có danh sách với các bullet khác nhau thay vì sử dụng những icon mặc định của nó.

Ưu điểm: Đơn giản, nhanh chóng đạt được kết quả.

Nhược điểm: Một số trình duyệt không hỗ trợ thuộc tính **list-style-image** do đó có thể bạn sẽ không nhận được kết quả như mong muốn.

Cách thứ hai:

Sử dụng thuộc tính background-image của CSS để thiết lập bullet cho danh sách.

Chúng ta vẫn tiếp tục sử dụng icon ở trên cho trường hợp này. Tuy nhiên selector dành cho thẻ ul được định nghĩa lại như sau:

```
ul{
list-style-type: none;
padding: 0px;
margin: 0px;
}
```

Selector ở trên sẽ tiến hành bỏ các bullet của danh sách (tròn đen, vuông, tròn trắng) và thiết lập các thông số canh lề, nội dung (margin, padding) về giá trị 0.

Lúc này danh sách của chúng ta sẽ có hình dáng như sau:

Tạp chí Lập trình

Tin mới
Array trong Javascript
Hiệu ứng Shakira
Tìm hiểu Cấu trúc dữ liệu #1: "Chết vì thiếu hiểu biết"

Bây giờ là công đoạn cuối trước khi xem xét kết quả, ta cần một selector nữa cho thẻ li, cụ thể như sau:

```
li{
   background-image: url('icons/
bullet_green.png');
   background-repeat: no-repeat;
   background-position: 0 5px; padding-left:
15px;
}
```

Nào chúng ta cũng xem xét thành quả của mình:

Tạp chí Lập trình

- Tin mới
- Array trong Javascript
- Hiệu ứng Shakira
- Tìm hiểu Cấu trúc dữ liệu #1: "Chết vì thiếu hiểu biết"

Để danh sách này thêm sinh động hơn, bạn tìm thêm một icon nữa dành cho trường hợp đưa con trỏ qua mỗi phần tử trong danh sách, chẳng hạn tôi dùng icon sau:

(bullet_yellow.png)

Chúng ta thêm một selector nữa cho thẻ li mỗi khi có hành động đưa con trỏ qua chúng:

```
li:hover{
   background-image: url('icons/
bullet_yellow.png');
   cursor: pointer;
}
```

Và đây là kết quả mà bạn nhận được với danh sách của mình:

Tạp chí Lập trình

- Tin mới
- Array trong Javascript
- Hiệu ứng Shakira
- Tìm hiểu Cấu trúc dữ liệu #1: "Chết vì thiếu hiểu biết"

Ưu điểm: Hỗ trợ bởi tất cả các trình duyệt, nhiều tùy chỉnh hơn đối với bullet.

Nhược điểm: Mất nhiều thời gian và dòng lệnh hơn so với cách thứ nhất

Kết luận:

Sau bài viết này bạn lại trang bị thêm cho mình một "vũ khí" mới để làm cho trang web của mình ngày càng sinh động hơn. Bạn cũng thấy được sự hiệu quả khi kết hợp HTML với CSS trong phát triển website. Chúc bạn thành công :o)

CSS3: Một số thuộc tính thú vị

Một số thuộc tính thú vị đã được giới thiệu trong CSS3, hầu hết các trình duyệt hiện nay đã hỗ trợ các thuộc tính này. Với các phiên bản cũ thì cần bổ sung thêm các tiền tố trước các thuộc tính:

Mozilla: `-moz-`
WebKit: `-webkit-`
Opera: `-o-`
IE: `-ms-`

1. **Border Radius** (góc của đường viền):
   `border-radius`

2. **Border Images** (đường viền sử dụng ảnh):
   `border-image`

3. **Box Shadow** (đổ bóng dành cho các khối):
   `box-shadow`

4. **Multi-Column Layout** (dàn trang với nhiều cột)
   `column-gap` (khoảng cách giữa các cột)
   `column-rule` (đường kẻ giữa các cột)
   `column-count` (số lượng cột)

5. **Multiple Backgrounds** (nhiều ảnh nền)
   Cho phép đưa nhiều ảnh nền vào khu vực nào đó.
   VD:

```
background:
url(topbg.jpg) top left no-repeat,
url(middlebg.jpg)center left no-
repeat,
url(bottombg.jpg) bottom left no-
repeat;
```

6. **Opacity** (Làm mờ)
   `opacity`

2013:
Năm của Responsive Web Design

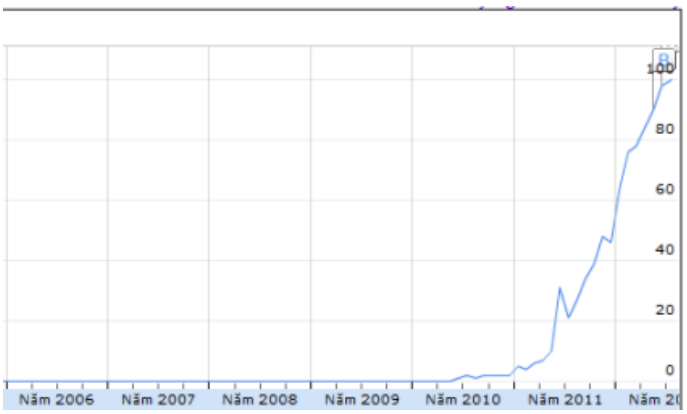
~ Nguyễn Khắc Nhật ~

Việc truy cập các website bằng các thiết bị di động đã phổ biến từ lâu nay, nhưng phải cho đến gần đây, với sự ra đời hàng loạt của các loại điện thoại thông minh và máy tính bảng thì nó mới trở thành một xu hướng thực sự. Theo một dự đoán của Gartner thì đến năm 2013 việc sử dụng các thiết bị di động để duyệt web sẽ trở nên phổ biến hơn là sử dụng PC.

Với xu hướng đó, hầu hết các các công ty hiện nay đều tính đến việc chuyển đổi website của mình để có thể hỗ trợ tốt việc duyệt web trên nhiều thiết bị khác nhau. Một giải pháp được đưa ra đó là tạo ra nhiều phiên bản khác nhau của website cho các thiết bị khác nhau: iPhone, iPad, netbook, Kindle, BlackBerry... và tất nhiên là cho cả pc. Thế nhưng, với sự phong phú về chủng loại của các thiết bị di động, cũng như đa dạng về kích thước màn hình như hiện nay thì chúng ta sẽ phải tạo ra bao nhiêu phiên bản cho đủ? Đó là chưa kể những trường hợp các thiết bị "thông minh" như iPad hiện nay có khả năng quay ngang quay dọc màn hình; ngoài ra, theo một thống kê khác, có một số lượng lớn người dùng thường không để cửa sổ trình duyệt của mình ở dạng cực đại; làm thế nào để các website có thể hiển thị tốt trong tất cả những trường hợp trên?

Tháng 5 năm 2010 Ethan Marcotte đã đăng một bài viết với tiêu đề "Responsive Web Design" đề cập đến một hướng tiếp cận mới cho việc thiết kế giao diện web, theo đó thì chúng ta không nhất thiết phải tạo ra vô số các phiên bản khác nhau cho website của mình, mà thay vào đó chúng ta chỉ nên tập trung vào một thiết kế duy nhất, và thiết kế này phải có khả năng tự động điều chỉnh, tự động thích nghi với với các kích thước màn hình khác nhau. Nhưng Responsive Web Design không chỉ dừng lại ở việc tạo ra các web-site có khả năng thích nghi với nhiều lại kích thước màn hình, mà nó còn là một cách suy nghĩ mới về việc thiết kế. Trong loạt bài này, chúng ta sẽ cùng nhau điểm qua những tính năng chủ chốt của Responsive Web Design, và chúng ta cũng sẽ có các ví dụ demo nhỏ để ta có thể dễ dàng hơn trong việc làm quen và tiến tới áp dụng Responsive Web Design vào trong các sản phẩm thực tế.

Để tìm hiểu xem sự quan tâm của người dùng đối với Responsive Web Design là như thế nào, tôi đã thử dùng công cụ Google Insights for Search và nhập vào từ khóa "Responsive Web Design" thì nhận được kết quả như sau:



Biểu đồ trên phản ánh số lượng tìm kiếm được thực hiện của từ khóa "Responsive Web Design", ta có thể dễ dàng nhận thấy rằng kể từ nửa cuối năm 2010 mới bắt đầu xuất hiện sự quan tâm đến khái niệm này, và đến nửa cuối năm 2011 cho đến nay thì nó đã tạo nên một xu hướng thực sự trên toàn thế giới (Chú ý rằng con số từ 0-100 được chia theo tỷ lệ chứ không phải là giá trị thực của số lượt tìm kiếm). Và chúng ta cũng có được kết quả thống kê theo từng vùng, theo đó thì sự quan tâm lớn nhất đến từ các quốc gia Bắc Mỹ, Bắc Âu, Ấn Độ và Úc, điều đó cũng không có gì là ngạc nhiên, bởi vì đây chính là các quốc gia có nền CNTT phát triển đứng vào top đầu của thế giới.

Các khái niệm cơ bản trong Responsive Web Design

Có 3 đặc điểm kỹ thuật cốt lõi để tạo nên Responsive Web Design:

- Media queries và media query listeners
- Một bố cục dạng lưới linh hoạt đi đôi với việc sử dụng các cách đo kích thước bằng các đơn vị đo tương đối.
- Sử dụng các ảnh với kích thước linh hoạt bằng cách thay đổi kích thước động hoặc là thông qua CSS

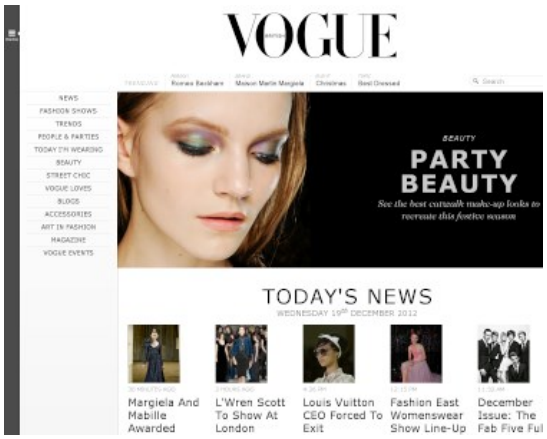
Để đạt được hiệu quả cao nhất thì nên sử dụng kết hợp cả 3 kỹ thuật trên với nhau. Điểm mấu chốt trong Responsive Web Design đó là sự thích nghi tốt nhất với khả năng của từng thiết bị và nhu cầu của từng người dùng. Có thể bạn sẽ cần phải thay đổi trật tự hiển thị của các khối thông tin đối với từng nhóm người dùng khác nhau, song song với việc thay đổi kích thước chữ, kích thước của từng vùng tương tác, tất cả nhằm mang lại trải nghiệm tốt nhất cho người dùng. Tất cả những yếu tố đó cũng ảnh hưởng đến việc trang web của bạn có được đánh giá là "Responsive" hay không.

Media Queries

Mặc dù Media Queries đã được sử dụng kể từ phiên bản CSS 2.1, tuy nhiên phải đợi đến CSS3 thì Media Queries mới được trang bị một sức mạnh vô cùng lớn lao. Với Media Queries bạn có thể áp dụng các *style* của mình cho từng trường hợp khác nhau, hay nói cách khác, bạn có thể dùng mệnh đề "if" ở trong CSS.

Media Queries Listeners

Sử dụng Media Queries Listeners bạn có thể đưa ra các phản hồi khi có một sự thay đổi nào đó của Media Queries. Ví dụ, bạn có thể tải về một tấm ảnh với kích thước nhất định khi có sự kiện thay đổi độ rộng của khung chứa nó.



Lưới linh hoạt (Flexible Grids)

Lưới linh hoạt có nghĩa là chúng ta sẽ sử dụng CSS để định vị các thành phần, tạo ra một số dạng bố cục theo một cách mới. Để có được Responsive Web Design chúng ta sẽ phải quyên đi một đơn vị đo mà ta rất yêu thích đó là pixel, thay vào đó sẽ là % hoặc là em. Bố cục dạng lưới cho phép chúng ta trình bày các thành phần dựa trên một bố cục bao gồm các hàng và cột.

Ảnh linh hoạt

Tính năng này cho phép chúng ta hiển thị các ảnh với kích thước phù hợp trong từng trường hợp, bằng cách co giãn ảnh hoặc là sử thuộc tính *overflow* của CSS.

Cấu trúc dữ liệu: "Chết" vì thiếu hiểu biết.

~ Dương Trọng Tấn ~

Nếu bạn đã học môn "cấu trúc dữ liệu và giải thuật", hoặc đọc hết (nghiêm chỉnh) một cuốn sách về chủ đề này, thì bạn có thể bỏ qua vệt bài tôi sắp đăng lên "**Tạp chí Lập trình**" có tên "*Tìm hiểu Cấu trúc dữ liệu*" này.

Mục đích chính của vệt bài này là cung cấp các hiểu biết "chả có gì mới" về các cấu trúc dữ liệu cơ bản (bao gồm các danh sách, hàng đợi, ngăn xếp, cây, bảng băm và đồ thị) bằng cách viết giản lược và những ví dụ minh họa. Để hiểu được các bài viết trong vệt bài này, bạn cần có kiến thức cơ bản về ngôn ngữ lập trình (Java, C, C#) trước. Tôi sẽ dùng Java làm ngôn ngữ minh họa, nhưng các vấn đề được đề cập thì cố gắng trừu tượng hóa nhất có thể, vì CTDL là một vấn đề nền tảng, không phụ thuộc ngôn ngữ – một vấn đề mà mọi lập trình viên đều phải quan tâm không kể là chị ta đang sử dụng ngôn ngữ nào. Và tôi sẽ cố gắng dùng ngôn ngữ "bình dân', thay vì ngôn ngữ "học thuật" như các sách về CTDL hay dùng.

***

Chúng ta cùng bắt đầu với một câu chuyện thường gặp trong các tình huống lập trình. Giả sử bạn nhận được một class được viết bởi người khác (từ một API nào đó, hoặc từ một thành viên khác trong team) với một hàm quan trọng getData() có nguyên mẫu như sau:

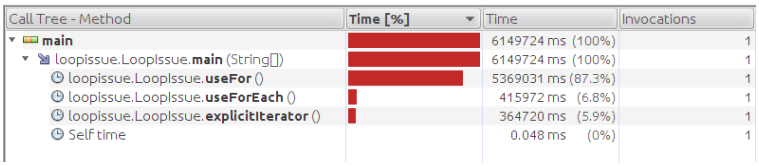
```
public List<Integer> getData();
```

Hàm này trả về tất cả các dữ liệu quan trọng cho các chức năng mà bạn sẽ code ngay sau đó. Dữ liệu trả về là một danh sách (List), và bạn bắt đầu viết đoạn code đầu tiên để duyệt (traverse) toàn bộ dữ liệu trong đó như sau:

```
//duyet danh sach
for (int i = 0; i < aList.size(); i++) {
   System.out.println(aList.get(i));
}
```

Đơn giản, đúng không? Có bao giờ bạn đặt câu hỏi "đoạn code trên có vấn đề gì không"? Nếu chưa, thì bạn vừa được đặt câu hỏi rồi đấy. Có vấn đề gì không?

Đoạn code đó về bản chất là "không vấn đề gì" nếu nó không rơi vào trường hợp oái oăm như sau: cái Collection mà người viết hàm getData() đã dùng để tổ chức dữ liệu là LinkedList. Khi đó, việc dùng không đúng cách như trên có thể dẫn đến hậu quả rất tồi tệ: bạn phải ngồi chờ hàng tiếng đồng hồ để chờ chương trình kết thúc, trong khi nếu biết cách dùng cho đúng, bạn chỉ mất vài phút để duyệt hết một danh sách một triệu phần tử (thường thi khi bạn code, bạn chỉ chạy thử với vài chục phần tử nên có thể không phát hiện ra, nhưng một chương trình chạy thì vài chục nghìn cho tới một vài triệu bản ghi nằm trong một collection là chuyện ... thường ngày ở huyện), như trong hình ghi lại từ Profiler (nếu bạn chưa dùng Profiler, hãy xem qua bài "Dùng Profiler đo hiệu năng ứng dụng Java") như sau:

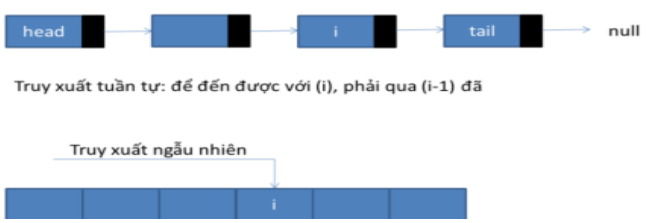


Hàm useFor() với đoạn code ở trên cần tới 5369031 ms (gần 90 phút) để hoàn tất việc duyệt qua tập hợp dữ liệu. Thử tưởng tượng nếu bạn code một ứng dụng web e-commerce, khách hàng sẽ bỏ bạn vì không thể kiên nhẫn đến thế. Nhìn vào hình trên, bạn thấy hàm useForEach() có thể thực hiện công việc tương tự mà chỉ cần tới 415972 ms (chưa tới 7 phút), đây là lõi của hàm useForEach():

```
//duyet qua danh sach
for (Integer i : aList) {
   System.out.println(i);
}
```

Sự khác nhau giữa hai đoạn code chỉ ở một điểm mấu chốt: lựa chọn for hay for-each để duyệt aList(). Đến đây bạn có thể đặt câu hỏi: vậy cái hàm useFor() chạy như rùa kia sai ở chỗ nào?

Câu trả lời rất đơn giản: việc duyệt theo kiểu random access (ngẫu nhiên) như trong useFor() (gọi hàm get(i) để duyệt phần tử thứ i) là sai nguyên tắc. Vì LinkedList được tổ chức đặc biệt, nên chỉ có thể được truy xuất tuần tự chứ không phải là truy xuất ngẫu nhiên thông qua chỉ số như là một danh sách dạng mảng. Để truy xuất phần tử thứ i của một danh sách liên kết, bạn sẽ phải bắt đầu từ phần tử đầu tiên (head), tuần tự đi qua các phần tử kế tiếp (thứ hai, thứ ba, v.v.) cho tới khi đến phần tử thứ i. Do đó, mỗi lần viếng thăm phần tử i, bạn tiêu tốn đúng "i" lần bước, do vậy bạn "đi" rất chậm, đặc biệt là khi "i" lớn.



Với việc dùng for-each, danh sách sẽ được dùng theo cách tối ưu với danh sách liên kết. Để hiểu rõ cơ chế duyệt theo for-each, chúng ta thử nhìn một đoạn code minh họa khác như sau đây:

```
for (Iterator<Integer> it = aList.iterator();
it.hasNext();) {
   System.out.println(it.next());
}
```

Trong Java, mỗi một tập hợp (collection) đều được tổ chức với một iterator (một đối tượng phụ trợ cho việc duyệt qua các phần tử bên trong tập hợp). Khi duyệt qua aList, iterator của một LinkedList đã đánh dấu phần tử đang duyệt (current), như đang đặt con trỏ (cursor) ở đó vậy; đo đó, khi gọi hàm it.next(), thì iterator đó không phải mất công dò từ đầu (head) cho tới phần tử thứ i, mà chỉ cần lần theo liên kết để đến với phần tử tiếp theo, tính từ vị trí đang đứng (current), mất thêm đúng một lần di chuyển.

Đó chính là lí do tại sao hàm useFor() lại mất thì giờ đến vậy, trong khi hàm useForEach() và explicitIterator() thì lại rất tiết kiệm thì giờ.

Bạn thấy đấy, chỉ khác có mỗi cái lệnh lặp mà chuyện xem ra phức tạp. "Sai một li đi một dặm". Trong lập trình, đôi khi tốc độ của chương trình chỉ do một dòng code quyết định. Và vệt bài "Tìm hiểu Cấu trúc dữ liệu" sẽ cố gắng cung cấp các hiểu biết cơ bản để bạn không phải mất "dặm" nào, với các tình huống liên quan đến cấu trúc dữ liệu. Nhưng đây là câu chuyện dài kì, hãy tạm dừng ở đây đã. Ở bài tới (sẽ rất sớm thôi) chúng ta sẽ trò chuyện về các cấu trúc tuần tự: danh sách.

ARRAY TRONG JAVASCRIPT

Có lẽ các bạn đã quen thuộc với Javascript qua loạt bài "Javascript và lập trình hướng đối tượng". Bài viết này tác giả Nguyễn Hiển sẽ giới thiệu một đối tượng quan trọng trong Javascript: Array.

Array có mặt trong hầu hết các ngôn ngữ lập trình, là một cấu trúc dữ liệu cho phép lưu trữ và truy xuất các phần tử ngẫu nhiên dựa trên vị trí. Trong Javascript, Array có gì khác?

Một lớp

Và do đó, chúng ta có thể tạo một đối tượng Array để lưu trữ các phần tử. Có 2 cách thường dùng để khởi tạo 1 đối tượng Array:

```
var arr = new Array("Bob", "Jobs", "Bill");
```

hoặc:

```
var arr = ["Bob", "Jobs", "Bill"];
```

sẽ khởi tạo 1 đối tượng mảng arr chứa các phần tử "Bob", "Jobs" và "Bill".

No-type

Như bạn đã biết, trong Javascript, nói chung các biến không cần chỉ định rõ kiểu. Do đó, chúng ta có thể chứa những phần tử có kiểu khác nhau trong cùng 1 mảng như:

```
var arr = new Array("Bobs","has","$",1000);
```

Điều này rất khác so với những ngôn ngữ như C++, Java... khi khai báo mảng với int arr[10]; sẽ giới hạn mảng arr chỉ chứa các số kiểu int.

Mutable

Điều gì xảy ra nếu chúng ta thực hiện đoạn mã sau?

```
var arr = new Array(3);
for (i = 0; i < 10; i++) {
   arr[i] = i;
}
```

Sẽ chẳng có lỗi OutOfRange nào như chúng ta mong đợi. Đơn giản vì Javascript tự điều chỉnh kích thước của mảng cho phù hợp, hay khả năng "co giãn", còn gọi là mutable. Có lẽ Array mang hình ảnh của List?

Và...

Ngoài việc lưu trữ những phần tử có kiểu khác nhau, đối tượng Array còn cho phép dùng nhiều kiểu dữ liệu khác nhau làm "key":

```
var arr = new Array("Bob", "Jobs", "Bill");
arr[1] = 1;
arr['hello'] = "World";
for (i in arr) {
   document.write(i + ': ' + arr[i]);
}
```

Đến đây chắc bạn cũng nhận ra, đối tượng Array không giống như mảng thông thường, chỉ cho phép truy xuất qua chỉ số là số nguyên, mà còn cho phép truy xuất qua chỉ số là một đối tượng bất kỳ. Thực ra, Javascript lưu trữ các đối tượng dưới dạng key-value-based, tức là mỗi đối tượng key (khoá) sẽ tương ứng (ánh xạ) với một đối tượng value (giá trị) theo cặp. Cấu trúc dữ liệu nào lưu trữ dưới dạng key-value-based? Có lẽ là Map.

Ngoài ra..

Hãy để ý các phương thức có sẵn của đối tượng Array, có lẽ bạn sẽ ấn tượng với 2 phương thức hay được sử dụng nhất: pop() và push().

Cấu trúc dữ liệu nào sở hữu những phương thức này? Stack và Queue. Vậy đối tượng Array mang hình ảnh của Stack hay Queue? Cách kiểm chứng đơn giản nhất là dựa trên cơ chế của 2 kiểu cấu trúc dữ liệu này: Stack hoạt động theo phương thức LIFO (Last In First Out – Vào sau ra trước), Queue thì ngược lại LILO (Last In Last Out – Vào sau ra sau) (Có người thích gọi cơ chế hoạt động của Stack là: FILO và Queue là: FIFO). Hãy đưa một phần tử vào một Array đã có dữ liệu qua phương thức push(), tiếp đó, lấy phần tử trong Array qua phương thức pop(). Nếu hai phần tử này giống nhau, chắc chắn Array hoạt động theo phương thức LIFO tức là Stack, ngược lại Array hoạt động theo phương thức LILO hay Queue:

```
var arr = new Array("Bob", "Jobs", "Bill");
arr.push('new item');
item = arr.pop();
alert(item);
```

Chắc bạn đã có câu trả lời.

~ Nguyễn Văn Hiển ~

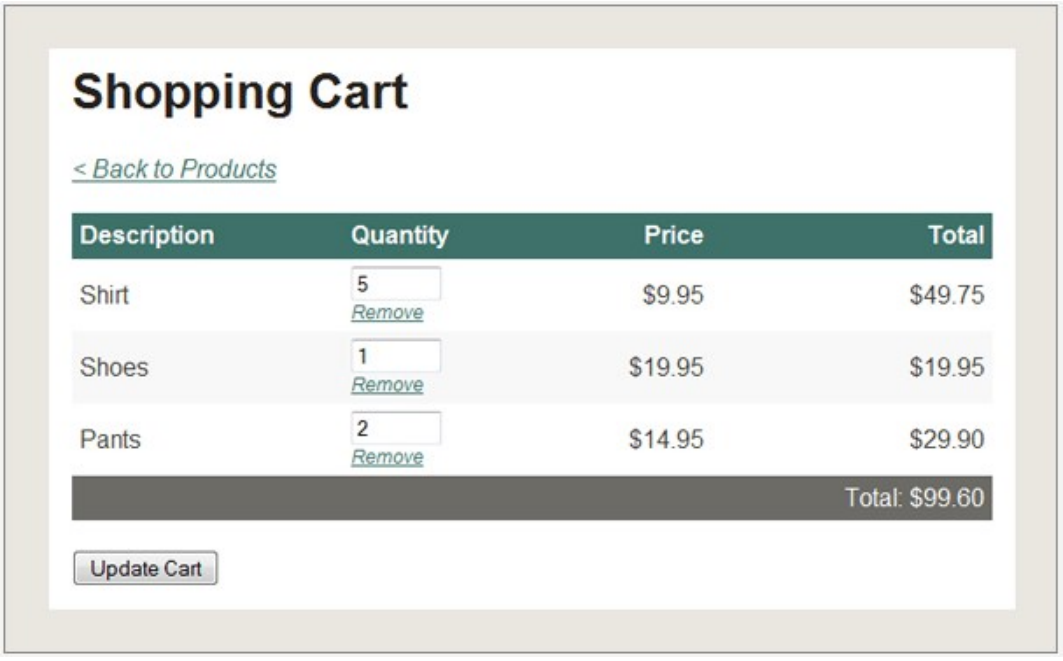
TẠO GIỎ HÀNG TRONG ASP.NET

Đặng Kim Thi

Câu chuyện về giỏ hàng trong website thương mại điện tử, nó cũng giống như chiếc xe đẩy chở hàng trong siêu thị. Tức là, bạn phải làm sao cho người dùng lựa chọn được sản phẩm, đưa vào giỏ hàng, thay đổi số lượng sản phẩm, xóa sản phẩm trong giỏ hàng. Và bạn nhớ rằng giỏ hàng chỉ là nơi lưu trữ tạm thời các sản phẩm mà bạn cần mua trước khi thanh toán.

Ngày nay, việc mua hàng qua website đã trở nên quen thuộc với nhiều người và nó mang lại một thị trường mở cho các nhà kinh doanh. Để khách hàng có thể mua hàng qua website của mình, thì họ cần phải xây dựng một website thương mại điện tử. Với vai trò là một lập trình viên, bạn thường xuyên gặp phải bài toán giỏ hàng trong những dự án ấy. Đây là một trong những nghiệp vụ quan trọng bậc nhất với những website thương mại điện tử.

Để hiểu về giỏ hàng thì chúng ta hãy tưởng tượng chúng ta đang đi siêu thị nhé. Việc đầu tiên khi bạn vào siêu thị là bạn làm gì? Tôi chắc rằng bạn sẽ cần tới một chiếc xe đẩy để chở hàng, hay một cái giỏ để đựng hàng, tôi xin tạm gọi hai thứ đó là "giỏ hàng". Công việc tiếp theo là lựa chọn sản phẩm để mua, bạn có thể nhặt sản phẩm định mua, đưa vào giỏ hàng, mua bao nhiêu sản phẩm thì bạn nhặt bấy nhiêu. Sau khi nhặt một hồi, bạn có thể thay đổi ý định, bỏ bớt một số sản phẩm, hoặc giảm hoặc tăng số lượng sản phẩm mình mua trong giỏ hàng. Những sản phẩm trong giỏ hàng là những sản phẩm bạn chọn để mua, và nó chỉ có tính chất lưu trữ tạm thời trước khi thanh toán. Tức là, khi bạn thanh toán xong giỏ hàng lại trở về trạng thái rỗng.



Câu chuyện về giỏ hàng trong website thương mại điện tử, nó cũng giống như chiếc xe đẩy chở hàng trong siêu thị. Tức là, bạn phải làm sao cho người dùng lựa chọn được sản phẩm, đưa vào giỏ hàng, thay đổi số lượng sản phẩm, xóa sản phẩm trong giỏ hàng. Và bạn nhớ rằng giỏ hàng chỉ là nơi lưu trữ tạm thời các sản phẩm mà bạn cần mua trước khi thanh toán.

Tôi thấy có khá nhiều bạn sinh viên đau đầu với bài toán kinh điển này. Để giúp các bạn bớt đau đầu, tôi sẽ hướng dẫn các bạn những thao tác cơ bản để xây dựng được giỏ hàng trong ASP.NET.

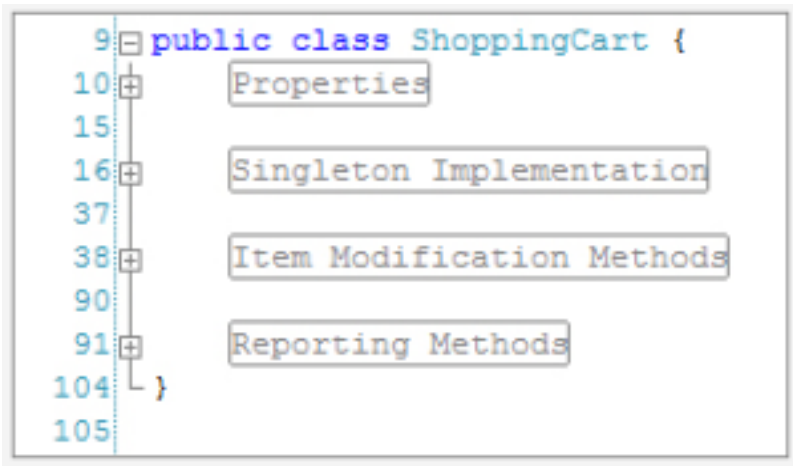
Để tạo ra được giỏ hàng, chúng ta cần có 3 class cơ bản là Shopping-Cart, CartItem, Product được mô tả như class diagram ở dưới đây.

1. Tạo Class ShoppingCart

Chúng ta cần một nơi để lưu trữ trữ các mục (item) trong giỏ hàng cũng như cung cấp chức năng để thao tác với những phần tử đó. Chúng ta sẽ tạo ra một class để làm được việc này. Class này sẽ quản lý việc lưu trữ session (phiên làm việc).

Đầu tiên, chúng ta phải tạo ra thư mục App_Code. Để làm được việc này, vào menu "Website", sau đó chọn "Add ASP.NET Folder", rồi chọn "App_Code". Đây chính là nơi chúng ta đặt toàn bộ class do mình viết ra. Từ bất kỳ trang web nào cũng có thể truy cập tới code trong App_Code (chúng ta không cần phải tham chiếu tới nó bằng cách sử dụng những lệnh kiểu như "include" hay bất cứ cái gì). Sau đó, chúng ta có thể thêm class vào thư mục này bằng cách kích chuột phải vào folder rồi chọn "Add New Item".

Mẹo nhanh: Regions (Khu vực) trong ASP.NET thực sự là một điều tuyệt vời để tổ chức và nhóm code lại với nhau. Điều tốt đẹp nhất về chúng là bạn có thể mở và đóng các khu vực để thu gọn số lượng mã mà bạn đang tìm kiếm hoặc nhanh chóng tìm thấy theo cách của bạn xung quanh một tập tin.



2. Tạo CartItem và Product

Bên cạnh việc tạo ra một nơi để lưu trữ các phần tử, chúng ta cũng cần tạo ra nơi để lưu trữ thông tin về mỗi mục (item). Chúng ta sẽ tạo ra class CartItem để làm việc này. Chúng ta cũng sẽ tạo ra một class Product đơn giản, nó sẽ chứa thông tin cơ bản về sản phẩm chúng ta đang bán.

Một "property" (thuộc tính) là một biến trong một class mà nó kèm theo setter, getter hoặc cả hai. Trong class Product chúng ta khai báo các thuộc tính của sản phẩm thông qua các property: Id, Price, Description. Ngoài ra, class Product còn khởi tạo một vài sản phẩm (3 sản phẩm) để chúng ta dễ dàng kiểm tra phần giỏ hàng.

Đây là Product class

```
/**
 * The Product class
 *
 * This is just to simulate some way of accessing data about  our products
 */
public class Product
{
    public int Id { get; set; }
    public decimal Price { get; set; }
    public string Description { get; set; }
    public Product(int id)
    {
        this.Id = id;
        switch (id) {
            case 1:
                this.Price = 19.95m;
                this.Description = "Shoes";
                break;
            case 2:
                this.Price = 9.95m;
                this.Description = "Shirt";
                break;
            case 3:
                this.Price = 14.95m;
                this.Description = "Pants";
                break;
        }
    }
}
```

3. Thêm vào giỏ hàng

Sau khi code rất nhiều, giờ là lúc chúng ta làm cái gì đó thật trực quan. Chúng ta sẽ tạo ra một page đơn giản để thêm các mục vào trong giỏ hàng. Tất cả, chúng ta có một vài mục cùng với link "Add to Cart" . Nào chúng ta hãy cùng code trang Default.aspx.

4. Hiển thị giỏ hàng

Tất cả những gì chúng ta làm ở trên chỉ là để chuẩn bị cho shopping cart! Giờ chúng ta hãy nhìn trang View-Cart.aspx.

Về cơ bản chúng ta sẽ sử dụng GridView để hiển thị thông tin trong giỏ hàng. Cột Description trong GridView chúng ta sẽ sử dụng BoundFiled. Còn cột Quantity chứa số lượng sản phẩm sẽ mua, ta cần sử dụng TemplateFiled, để có thể đưa TextBox và link Remove luôn vào đó. Để hiển thị thông tin như Price, De-scription .... trên GridView chúng ta cần dùng tới <%# Eval ("PropertyName") %>.

Kết quả

Bây giờ, chúng ta đã có một shop-ping cart khá là đẹp mắt !

Bài này mới chỉ dừng lại ở việc hướng dẫn bạn xây dựng giỏ hàng bằng ASP.NET sử dụng C#, nhưng từ đây bạn hoàn toàn có thể thể triển khai cách làm này với các ngôn ngữ khác. Chúc bạn thành công với những website thương mại điện tử của mình !

Bài viết có tham khảo thông tin của net.tutspluts.com

Để xem và download SourceCode hãy lên blog Tạp Chí Lập Trình.

Để Không Mất Động Lực Học Tập

10 cách để bạn không bị mất hứng trong lúc học.

1. **Xác định một mục đích rõ ràng và thực tế** mà bạn có thể làm được, phải chắc rằng đó thực sự là mục-đích-của-bạn chứ không phải là mục đích của bố mẹ, người xung quanh hay của số đông. Có thái độ và suy nghĩ tích cực để theo đuổi mục tiêu mà mình đề ra trong việc học, cũng như khi thực hiện kế hoạch.

Bắt tay vào nào: Dán một tờ stick note (tờ giấy dán, thường để ghi chú lên đó) ghi thời hạn chót nộp bài lên lịch, sau đó đánh dấu ngày bạn sẽ bắt đầu tiến hành làm bài cũng trên tờ lịch đó.

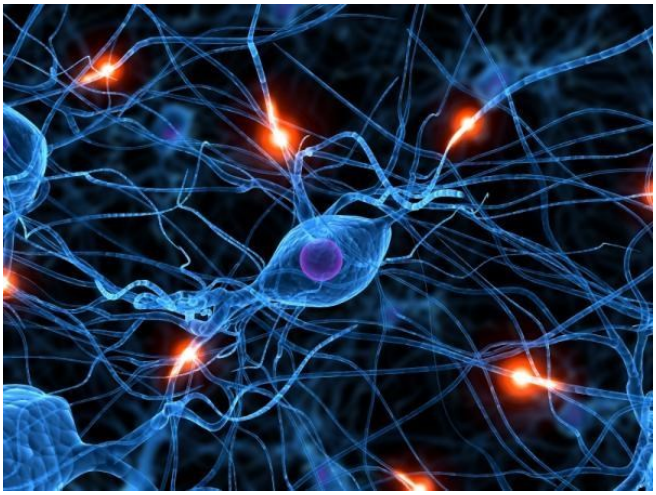
2. **Lên danh sách những yếu tố thúc đẩy** bạn học hành: khách quan (nhận được lời khen của bố mẹ, quà thưởng, học bổng...), chủ quan (đạt được trình độ cao cấp trong lĩnh vực mình đang học, thoả mãn sự ham mê tìm hiểu của bản thân...).

3. **Tạo một áp lực thời gian** cho bản thân khi làm bài tập, nếu không có áp lực về thời gian, bạn sẽ dễ lãng quên nhiệm vụ chính của mình và dần dần mất hứng thú khi bắt tay vào làm. Có một cách cực tốt: Dán tờ stick note (tờ giấy dán, thường để ghi chú lên đó) ghi thời hạn chót nộp bài lên lịch, sau đó đánh dấu ngày bạn sẽ bắt đầu tiến hành làm bài cũng trên tờ lịch đó.

4. Nếu bạn thấy bài tập quá nhiều và nặng, **hãy chia nhỏ ra làm nhiều phần**. Mỗi ngày làm một chút, nhưng phải chắc chắn là mình làm xong chứ không để dồn sang hôm sau.

5. Bạn muốn hoàn thành sớm bài tập thì chọn phần **dễ trước, khó sau,** chọn những phần nào bạn cảm thấy hứng thú hoặc những đề mục nhỏ trước. Việc hoàn thành một cách nhanh chóng những phần như thế sẽ khiến bạn tự tin hơn về khả năng của mình.

6. Nếu cảm thấy khó khăn hoặc khó hiểu ở điểm nào trong bài tập, **đừng ngại hỏi** giáo viên hoặc người hướng dẫn. Sự giảng giải ngắn gọn của họ sẽ giúp bài tập trở nên dễ hiểu hơn, do đó bạn có thể tiếp tục phát triển bài làm nếu đi đúng hướng, cũng như hạn chế được những sai sót trong quá trình thực hiện.



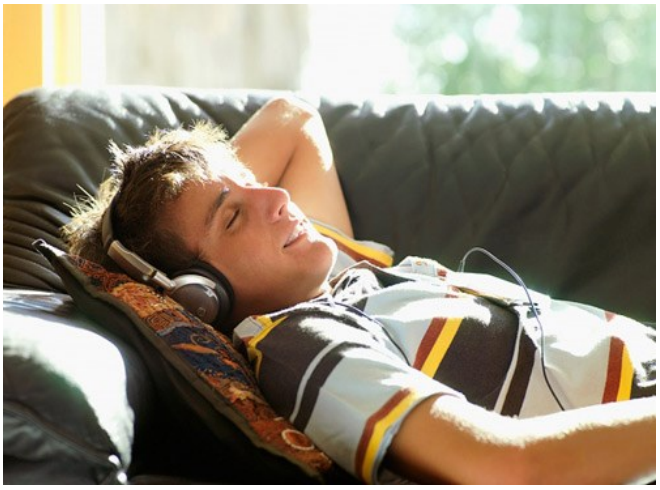
7. **Tìm mối liên hệ** giữa những gì bạn đang học/đang làm với những gì bạn sẽ thực hiện trong tương lai.

8. Cố gắng **giải quyết những vấn đề cá nhân** có thể làm ảnh hưởng đến sự tập trung của bạn, nếu không, hãy điều tiết sao cho nó không can thiệp sâu vào việc học.

9. **Hạn chế những suy nghĩ hoặc thái độ thiếu tích cực** như: chần chừ, chờ đợi may mắn mỉm cười, tự ti... khi học. Nhìn vào những thành công hoặc kết quả mà bạn đạt

được, tuy nhỏ thôi, nhưng nó có thể thay đổi thái độ của bạn đấy.

10. Mỗi khi hoàn thành xong một phần bài tập đề ra, **bạn hãy tự thưởng** cho mình nhé. Một que kem, thanh kẹo, một giờ nghe nhạc hoặc xem phim... vừa khiến đầu óc bạn thoải mái hơn, vừa duy trì được sự nhiệt tình trong bạn. Đừng nghĩ đến những gì chưa hoàn thành, hãy hài lòng với những gì mình đã hoàn thành bạn nhé!



Nguồn: ione.net

Coding Dojo là gì?

Coding Dojo là một buổi làm việc mà ở đó nhóm lập trình viên sẽ cùng nhau giải quyết một bài toán với mục tiêu nâng cao kỹ năng nào đó trong không khí vui vẻ. Đây là một hoạt động thực hành có chủ ý.

ParisDojo tập trung vào việc biểu diễn lập trình trước những người khác, họ thường làm một chương trình hoàn chỉnh trong khoảng thời gian ngắn từ 1 đến 1,5 giờ và dùng nhiều ngôn ngữ, công cụ và các dạng bài tập khác nhau. Một bài tập được coi là thành công khi nó được hoàn thành trong thời gian đã định và những người khác có thể tự làm lại bài tập đó.

Tiên đề: Việc tiếp thu các kĩ năng lập trình phải là một quá trình liên tục.

Đặc điểm: Cộng tác, tạo không khí làm việc vui vẻ, không cạnh tranh. Các đối tượng với mọi trình độ, kĩ năng đều có thể tham gia. Thoải mái đề xuất, thử ý tưởng mới.

Cơ sở vật chất: Phòng họp phải có đủ chỗ ngồi. Cần phải có tối thiểu một máy tính xách tay hoặc để bàn. Có một máy chiếu.

Cấu trúc của một phiên làm việc: Phiên bản ParisDojo tổ chức theo cấu trúc sau:

- 2 phút: Để quyết định thời gian cho buổi gặp mặt tiếp theo.
- 25-30 phút: Xem nhanh lại những gì của buổi trước, những gì tốt, thú vị, những gì còn chưa tốt.
- 10 phút: Quyết định chủ đề của buổi gặp này.
- 40 phút hoặc hơn: Lập trình! Cách thức làm việc tùy theo là PreparedKata hoặc RandoriKata.
- 5-10 phút: Nghỉ giữa phiên làm việc để thảo luận cách mọi việc đang diễn ra.
- 40 phút: Lập trình thêm.

Các hình thức:

- PreparedKata: Một thuyết trình viên sẽ chỉ cách giải quyết thử thách dùng TDD hoặc BabySteps. Mỗi bước làm phải có ý nghĩa cho các thành viên. Mọi người chỉ ngắt khi không hiểu điều gì đang diễn ra.
- RandoriKata: Thử thách được giải quyết bằng cách lập trình theo cặp (bao gồm driver và navigator). Mọi người có mặt đều có thể giúp đỡ. Mỗi cặp có 1 khoảng thời gian ngắn (5 đến 7 phút) để viết mã nguồn, dùng TDD và BabySteps. Kết thúc thời gian trên, người vừa làm driver sẽ về chỗ ngồi làm khán giả, người navigator trở thành driver và 1 khán giả sẽ lên làm hoa tiêu.

***Nguồn**: http://codingdojo.org. **Biên tập**: Phạm Anh Đới cùng nhóm sinh viên.*

Thư Tình của Lập Trình Viên

"Gửi em yêu,

Anh nhận ra em khi đang lướt trên sân ga và thấy rằng em là site duy nhất để anh truy cập. Em biết không, từ lâu lắm rồi anh rất cô đơn và đang phân tích, test kỹ lại lỗi lầm của chính mình. Có thể em sẽ là người gỡ lỗi và kiểm thử thực sự cho anh.

Không có em, đời anh như một phần mềm dang dở, không tạo được mã thực thi, không có người sử dụng đích thực và trở nên vô dụng, chỉ tốn tài nguyên mà thôi.

Em không chỉ có giao diện đẹp, tiện dụng mà còn có cả tư thế phòng vệ ActiveX thật là dễ thương trước bọn người dùng con trai lố nhố. Nụ cười tươi sáng, tông màu giao diện áo quần em thật nhẹ nhàng càng làm anh thấy mình nên thay đổi có gam màu mạnh mẽ, một sức mạnh mới tương đương với hàng ngàn con vi xử lý Dual Core hiện nay gộp lại.

Anh viết thư này bằng mã Unicode tiếng Việt chỉ là thổ lộ một điều rằng chúng ta 2 chip Nam, chip Nữ hãy nên liên kết lại với nhau. Anh sẽ mang lại cho em trong thời đại truyền thông tích hợp này mọi thứ, mọi thông tin và năng lực xử lý cần thiết để hai ta cùng sống một cuộc đời không có lỗi hay trục trặc gì hết.

Em cũng đừng quá lo về firewall mà bố mẹ em hay cộng đồng dựng lên vì anh crack và hack rất giỏi. Anh sẽ cày đêm và phá bằng được password hay những cản trở truy cập của họ để họ chấp nhận đám cưới của chúng ta như một entry miễn phí của cộng đồng. Và anh sẽ cùng em tự do như bao blog nóng hổi của bầu trời ảo lung linh mỗi ngày và rồi page views của mình sẽ cực cao. (St)

Thơ Tình cho Lập Trình Viên

Đừng dại yêu dân IT

Đừng dại mà quen bọn IT
Chúng nó khô khan, lãng mạn gì?
Viết thư tán gái thì kinh dị.
Chúng viết bằng gì? Ngôn ngữ C.

Đừng dại mà yêu bọn IT.
Chúng nó tài năng, mỗi tội kỳ.
Bạn gái chúng đòi đi đăng ký
"Bẻ khóa được rồi đăng ký chi?".

Đừng dại mà yêu bọn IT.
Chúng nó 35 đến lạ kỳ.
Gặp nhau anh í tòan năn nỉ.
Em mở mã nguồn cho anh đi.

Đừng dại mà yêu bọn IT
Chúng nó yêu đương cái kiểu gì
Gặp thì đòi cắm u ét bí (usb)
Lúc về nó bảo "Format đi"

```
if (this expression is true) {
   code block
} else {
   another code block
}
```

Hai ly nước

Một lập trình viên trước khi đi ngủ đặt lên tủ 2 cái ly:

Một ly có nước để phòng nếu đêm muốn uống nước.

Một ly không có nước để dùng trong trường hợp không muốn uống.

Chỉ Lập trình viên mới hiểu

```
int array[2]=new int[]{1,2};
int n=array.length-1;
for(int i=0;i<n;i++){
   array[0]=i;
   array[1]=i+1;
}
```

```
int getRandomNumber() {
   //chosen by fair dice roll
   //guaranteed to be random
   return 4;
}
```

Chúc Mừng Năm Mới

Thông tin Ban biên tập

Ban biên tập Tạp Chí Lập Trình bao gồm các thành viên: Nguyễn Việt Khoa, Dương Trọng Tấn, Nguyễn Ngọc Tú, Phạm Anh Đới, Đặng Kim Thi, Nguyễn Khắc Nhật.

Website: http://tapchilaptrinh.wordpress.com

Facebook: http://www.facebook.com/tapchilaptrinh

Email: lienhe.tapchilaptrinh@gmail.com